

# What is Express.js?

- Web application framework for Node.js
- Light-weight and minimalist
- Provides boilerplate structure & organization for your web-apps

# Installing Express.js

- Install node and node package manager (npm)
  - ‘sudo apt-get install nodejs’ on Debian/Ubuntu.
  - Installer available for Mac [1]
  - Installing node should also install npm
  - Running ‘node’ on a shell would now open an interactive node session (just like python)

[1] <http://nodejs.org/download/>

# Installing Express.js

- We can now create a project (say a blog), initialize it as a node package & install express.js
  - ‘mkdir blog && cd blog’
  - ‘npm init’
    - Creates a node package (touching package.json)
    - package.json: lists all dependencies (node modules)
  - ‘npm install express --save’
    - Install express and add it as a dependency to our app

# Hello World - Example

Let's write a simple hello world program using node and express.

We create a file 'app.js' inside the blog directory, with the following contents:

```
1  var express = require('express');  
2  var app = express();  
3  
4  app.listen(3000);  
5
```

- The app is invoked with 'node app.js' on the console. At this point, node listens on port 3000, but doesn't really do much.
- We can check this out, by accessing <http://localhost:3000> on the browser.

# Example - Continued

With a few more lines:

```
1  var express = require('express');
2  var app = express();
3
4  app.get('/', function(request, response){
5      response.send('Yay!');
6  });
7
8  app.listen(3000);
9
```



- Declare an instance of express called 'app'
- Accept requests on '/' (the root path on the URL – localhost:3000) and pass it to function (more on this later).
- The processing function or 'callback' receives the request and is expected to present with a response.

# Example - Continued

With a few more lines:

```
1  var express = require('express');  
2  var app = express();  
3  
4  app.get('/', function(request, response){  
5    response.send('Yay!');  
6  });  
7  
8  app.listen(3000);  
9
```

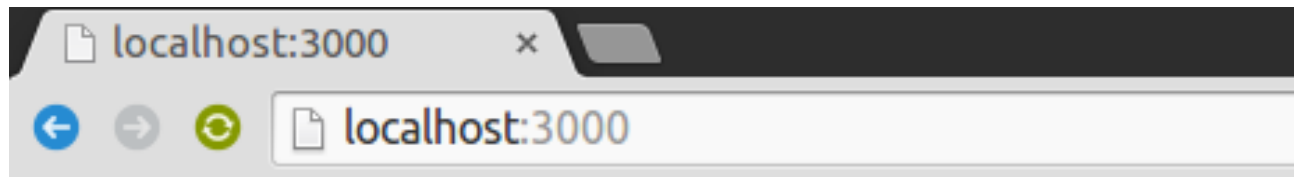


- Send a string as the response.

# Example - Continued

We can obviously add some 'dynamically' generated content:

```
1 var express = require('express');
2 var app = express();
3
4 app.get('/', function(request, response){
5   var d = new Date();
6   response.send("Yay! <br/> Today's date is <b>" + d + "</b>");
7 });
8
9 app.listen(3000);
10
```



Yay!

Today's date is: **Mon Feb 02 2015 23:45:57 GMT-0500 (EST)**

# Routing

- Apps needs to know what exactly to do when a request is made to a particular endpoint.
  - `http://localhost:3000/` -- `'/'` is the endpoint
  - `http://localhost:3000/admin` -- `'/admin'` is the endpoint
- We do this by defining routes. In the previous example, we defined one route `'/'` which returned a string as the response.



# Routing - structure

- `app.METHOD(ENDPOINT, HANDLER)`
  - Equivalent to `app.action(where, what-to-do)`
- **ENDPOINT:** What path is the request directed to.
  - This can be a regular expression.
  - `app.get('/ab*cd', .....);` will match:
    - `localhost:3000/abcd`, `localhost:3000/abXYZcd` and so on.

# Routing - structure

- `app.METHOD(ENDPOINT, HANDLER)`
  - Equivalent to `app.action(what, where, what-to-do)`
- **ENDPOINT:** What path is the request directed to.
  - This can be a regular expression.
  - `app.get(/.*cat$/, .....);` will match:
    - `localhost:3000/happycat`, `localhost:3000/sadcat` and so on.

# Routing - structure

- `app.METHOD(ENDPOINT, HANDLER)`
  - Equivalent to `app.action(where, what-to-do)`
- **HANDLER:** Or a 'callback' – function that determines how to process the request.
  - Takes in a few parameters (request and response as an example).

# Routing - structure

- Handler: 

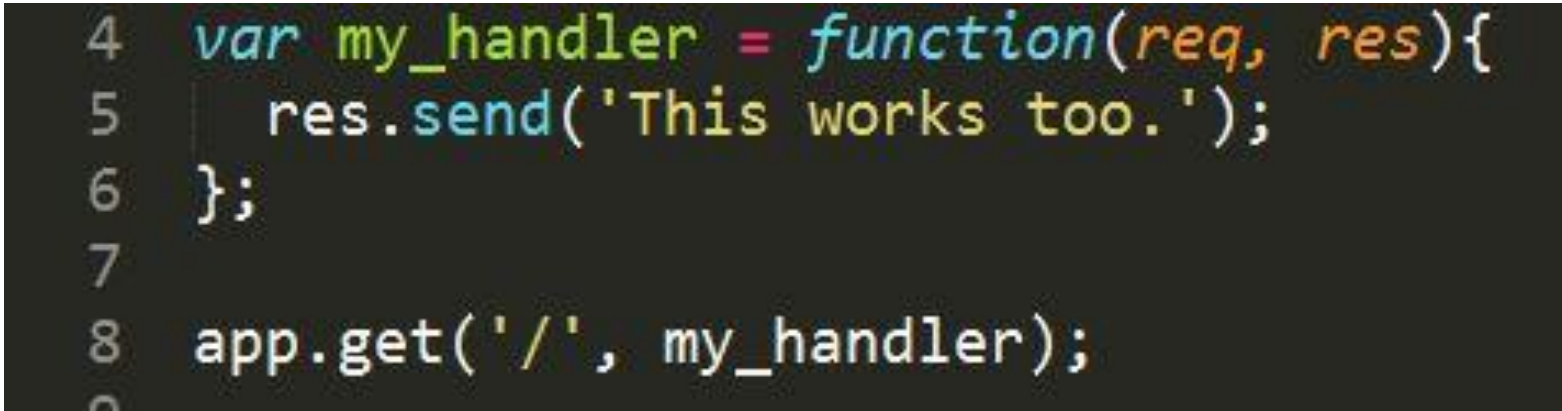
```
4 app.get('/', function(request, response)){
```

  - Sent in an anonymous function that would receive the request and return a response.
  - Data pertaining to this instance of the request can be obtained through the 'request' variable.
  - Data pertaining to the response should be handled through the 'response' variable.

# Routing - structure

- Handler:
  - Handler functions can also be passed as variables for brevity.

```
4  var my_handler = function(req, res){  
5      res.send('This works too.');
```



```
6  };  
7  
8  app.get('/', my_handler);  
9
```

# Routing – chaining handlers

- Handlers can be chained together.
  - Route handlers take a third parameter called 'next'.  
Invoking this variable as a function, delegates the request to the next handler (if there is one).

```
4  var secret_check = function(req, res, next){
5      // Some logic for checking access to secret resource
6      // eventually fails, user isn't allowed
7      next();
8  };
9
10 var error = function(req, res){
11     res.send("Access denied.")
12 };
13
14 app.get('/', secret_check, error);
```

# Routing – chaining handlers

```
4 var secret_check = function(req, res, next){
5   // Some logic for checking access to secret resource
6   // eventually fails, user isn't allowed
7   next();
8 };
9
10 var error = function(req, res){
11   res.send("Access denied.")
12 };
13
14 app.get('/', secret_check, error);
```

- Ordering of the handlers matter.
- If `secret_check()` sent in a response (instead of calling `next`) the delegation to `error()` won't happen and the handler chain would stop with `secret_check()`

# Response methods

Method	Description
<code>res.download()</code>	Prompt a file to be downloaded.
<code>res.end()</code>	End the response process.
<code>res.json()</code>	Send a JSON response.
<code>res.jsonp()</code>	Send a JSON response with JSONP support.
<code>res.redirect()</code>	Redirect a request.
<code>res.render()</code>	Render a view template.
<code>res.send()</code>	Send a response of various types.
<code>res.sendFile</code>	Send a file as an octet stream.
<code>res.sendStatus()</code>	Set the response status code and send its string representation as the response body.



# Aside: Logging

- 'console.log(...)' is extremely useful for logging information server side (by default would print out to the console).
  - Helps with debugging.

```
8 app.get('/', function(req, res){  
9   console.log("Someone made a request from: " + req.connection.remoteAddress)  
10  res.send('Yay!');  
11 });
```

- Prints out the client's IP address for every request they make -- to the console (where you invoked 'node app.js')

# Express generator

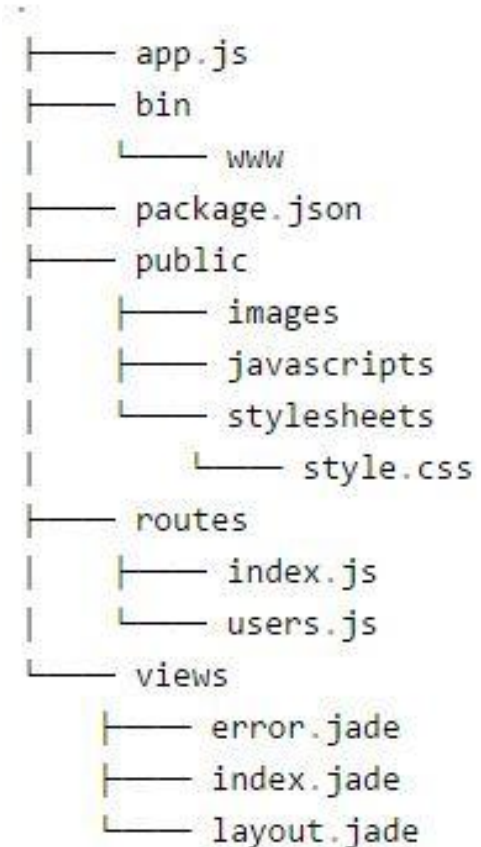
- Express provides a tool that can create and initialize an application skeleton.
  - Sets up a directory structure for isolating your business and presentation logic (among others).
  - Not a norm in any way, can be used as a starting point.

# Express generator

- How to use it?
  - Installing using `'npm install express-generator -g'`
  - Can then be invoked with `'express'`
- Create an app (for instance, a blog)
  - `'express blog'` – will generate the directories/files
  - `'cd blog && npm install'` – will install the dependencies
  - Invoke the server using `'DEBUG=blog ./bin/www'`

# Express generator

- Provides & sets up some boilerplate code for getting started faster
- Adds some commonly used dependencies
- Pretty good starting point for digging deeper into Node.js/Express.js
- Creates a structure that allows separation of concerns on business/presentation logic (more on this next week)



# Resources

- <http://expressjs.com> (Sections: Guide, API ref.)
- <http://expressjs.com/starter/faq.html>
- <http://code.tutsplus.com/tutorials/introduction-to-express--net-33367>
- Some resources on the slides are based on 'Getting Started' section from <http://expressjs.com>

# Using template engine

- First you need to tell express to use JADE template engine.
- Then you can create JADE template files inside view directory.

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

# Using template engine

index.jade

html

head

title!= title

body

h1!= message

# Using template engine

app.js

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