

# WEB TECHNOLOGIES & PROGRAMMING (CSC350)

Lecture 11, 12  
Instructor: [Humaira Waqas](#)

# EXPRESS

Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of middleware function calls.

- It is Node.js web application framework
- It facilitates the rapid development of Node based web applications
- It acts as a middleware to respond to HTTP requests
- Facilitates the rendering of dynamic HTML views
- Easier to organize applications functionalities with the middleware.
- It helps to abstract away lot of Node.js complexities to simple code
- It lets you refactor one monolithic request handler function into many smaller request handlers that handle only specific bits and pieces, which is more maintainable and modular.

# EXPRESS

- **Express**

- `npx express-generator` command (available in Node.js 8.2.0)
- or `npm install -g express-generator`
- `Express myapp`
- `cd myapp`
- `npm install`
- `npm start`
- <http://localhost:3000/>
- `npm install nodemon`
- `npm i -d nodemon` or `npm i -g nodemon`

You can run the application generator with the `npx`

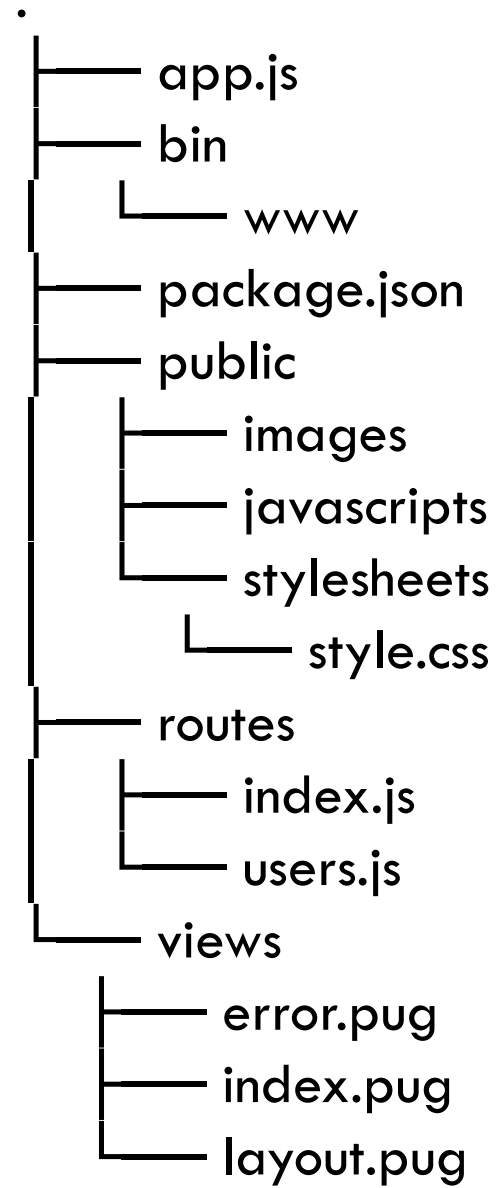
For earlier Node versions, install the application generator as a global npm package and then launch creates an Express app named `myapp`

install dependencies

start the server

access the browser

# EXPRESS



7 directories, 9 files

# EXPRESS - MIDDLEWARE

## Middleware

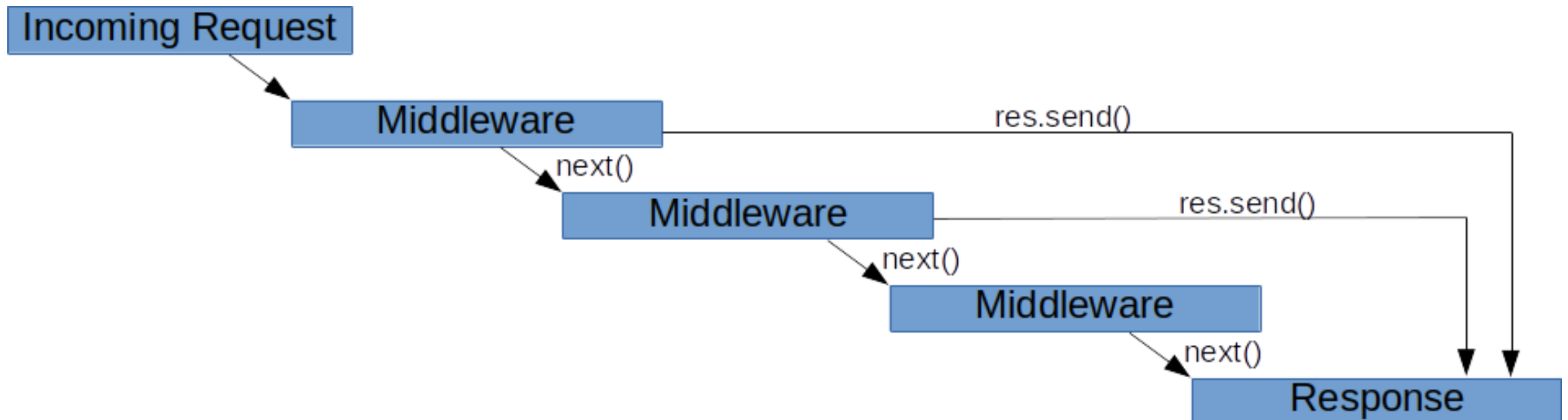
Functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- Call the next middleware function in the stack.
- End the request-response cycle.

# EXPRESS - MIDDLEWARE

- When a request is received by Express, each middleware that matches the request is run in the order it is initialized until there is a terminating action (like a response being sent).
- If an error occurs, all middleware that is meant to handle errors will be called in order until one of them does not call the `next()` function call.



# EXPRESS - MIDDLEWARE

Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the applications request-response cycle.

**Request:** it is the HTTP request that reaches the Express application when a client makes HTTP request (PUT, GET, DELETE etc)

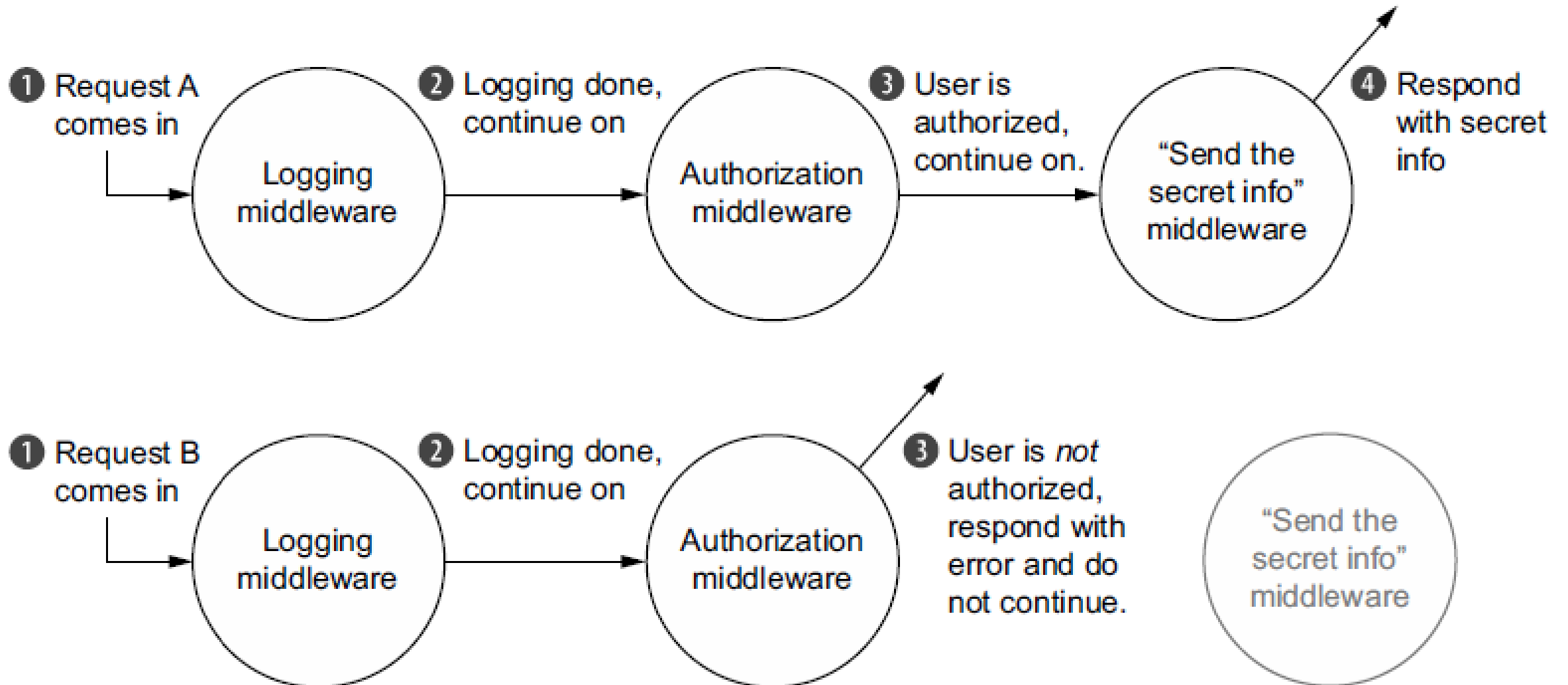
**Response:** it represents the HTTP response that an Express application sends when it get an HTTP request

**Next:** next function in the middleware stack.

**Request-response cycle:** the cycle of operations that get executed starting with a request hitting the Express application till the response leaves the application

**Middleware stack:** Stack of middleware functions that get executed for a request-response cycle.

# EXPRESS - MIDDLEWARE





# EXPRESS - MIDDLEWARE

## Define middleware function

```
function logger(req, res, next){  
    next() //calls the next function in the middleware stack  
}
```

**req** is the HTTP request object.

**res** is the response object

**next** is the next function in request-response cycle.

## Call Middleware

In express application, you call middleware using **use** function on application object.

```
app.use(logger)
```

# EXPRESS - ROUTING

Routing is the process of selecting a path for traffic in a network or between or across multiple networks.

- Like middleware, it breaks the one monolithic request handler function into smaller pieces.
- Unlike middleware, these request handlers are executed conditionally, depending on what **URL and HTTP** method a client sends.
  - For example, you might build a web page with a homepage and a guestbook.
  - When the user sends an HTTP GET to the homepage URL, Express should send the homepage. But when they visit the guestbook URL, it should send them the HTML for the guestbook, not for the homepage!
  - And if they post a comment in the guestbook (with an HTTP POST to a particular URL), this should update the guestbook.
  - Routing allows you to partition your application's behavior by route.

# EXPRESS - ROUTER

Express router is a class (`Express.Router`) which helps us to create **router handlers**

## Define routes

```
app.method(path, handler)
```

The `method` can be applied to any one of the HTTP verbs – `get`, `set`, `put`, `delete`.

An alternate method also exists, which executes independent of the request type.

**Path** is the route at which the request will run

## Router handler

Router handler is a callback function that executes when a matching request type is found on the relevant route.

# EXPRESS - MIDDLEWARE

`app.use(middleware)`

called every time a request is sent to the server

or used to mount the middleware function to a specified path  
(the middleware function is executed when the base path matches)

`app.all()`

special routing methods which is used to load middleware functions at a path for all HTTP request methods.

`app.get()`

`app.post()`

`app.put()`

`app.delete()`

`app.set('title', 'Express Server')`

gives a common place to read and write application wide settings. This code will assign the server setting title to express server

`app.get()`

get these values back using `app.get('title')`

# EXPRESS - MIDDLEWARE

```
var express = require('express'); HTTP method for which the middleware function applies.  
var app = express();
```

Path (route) for which the middleware function applies.

The middleware function.

```
app.get('/', function(req, res, next) {  
  next();  
})
```

Callback argument to the middleware function, called "next" by convention.

```
app.listen(3000);
```

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

# EXPRESS — ROUTING

## Static Routes

```
App.get('/', requestProcessFunction)
```

```
App.get('/student', requestProcessFunction)
```

## Dynamic Routes

Express path with route parameters like:

- `/:id` , `/:id?`
- `/student/:status/current/:id`
- `/student/:status/alumini/:batch?`

The route parameters `:status` and `:id` will be parsed from the URI and made available via `req.params.status` and `req.params.id`

Optional parameter is defined by `?` as a postfix on a parameter e.g. `:batch?`

Express also allows regular expressions to be used as URI's

- `app.get('/things/:id([0-9]{5})', function(req, res){ }`

# EXPRESS — REQUEST OBJECT

Express request object represents the HTTP request and has properties for the request

- query string
- Parameters
- Body
- HTTP headers

`req.path`                      it contains the path part of the request URL

`req.query`                    object containing a property for each query string parameter in the route

`req.body`                    contains key-value pairs of data submitted in the request body

`req.get`

# EXPRESS — RESPONSE OBJECT

<code>res.write</code>	build up the response body with content
<code>res.status</code>	set the HTTP status code of the reply
<code>res.end</code>	end the request by responding to it
<code>res.send</code>	do a write and end



# EXPRESS — ROUTING

```
express.static(root)
```

To use multiple static assets directories, call the `express.static` middleware function

```
app.use("/", express.static(__dirname));
```

- `__dirname` is a global object that contains the name of the directory that the executing script resides from. For example, if you are running `node script.js` from `/home/user/env`, then `__dirname` will contain `/home/user/env`.

# EXPRESS - ROUTING

```
app.all('*', function(req, res, next) {  
    res.writeHead(404, { 'Content-Type': 'text/html' });  
    res.end('404 - Page Not Found');  
});
```

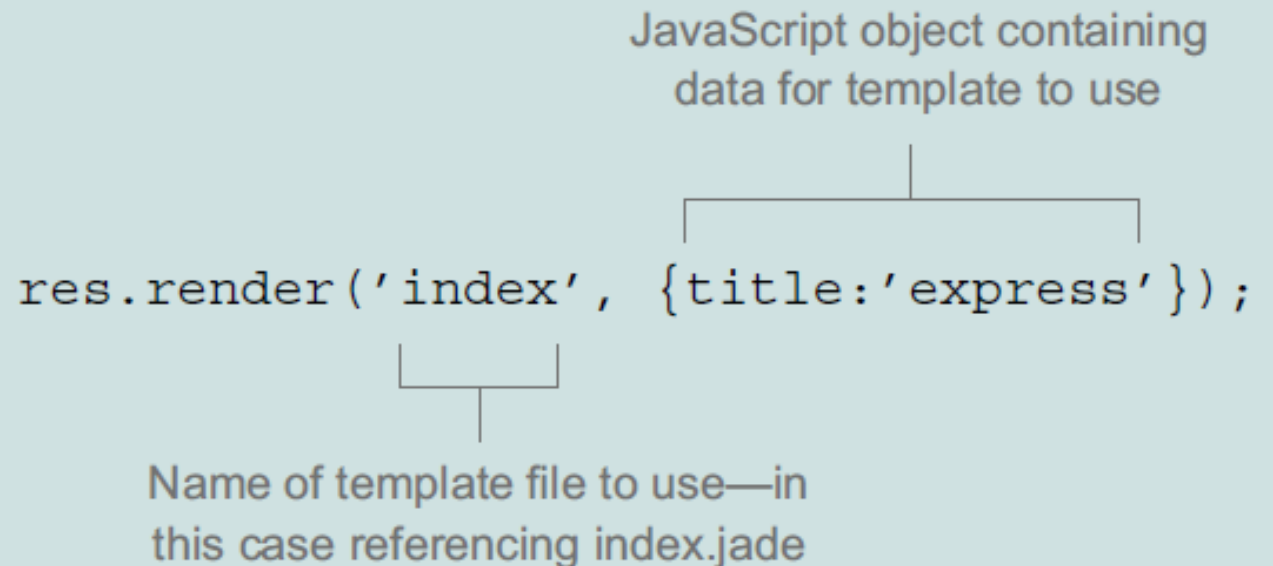
Here \* is a wild card and app.all means all methods.

# EXPRESS - ROUTING

Template file doesn't need to have the file extension suffix, so `index.jade` can just be referenced as `index`. You also don't need to specify the path to the view folder because you've already done this in the main Express setup.

`res.render` is the Express function for compiling a view template to send as the HTML response that the browser will receive.

The render method takes the name of the view template and a JavaScript data object in the following construct:



```
res.render('index', {title:'express'});
```

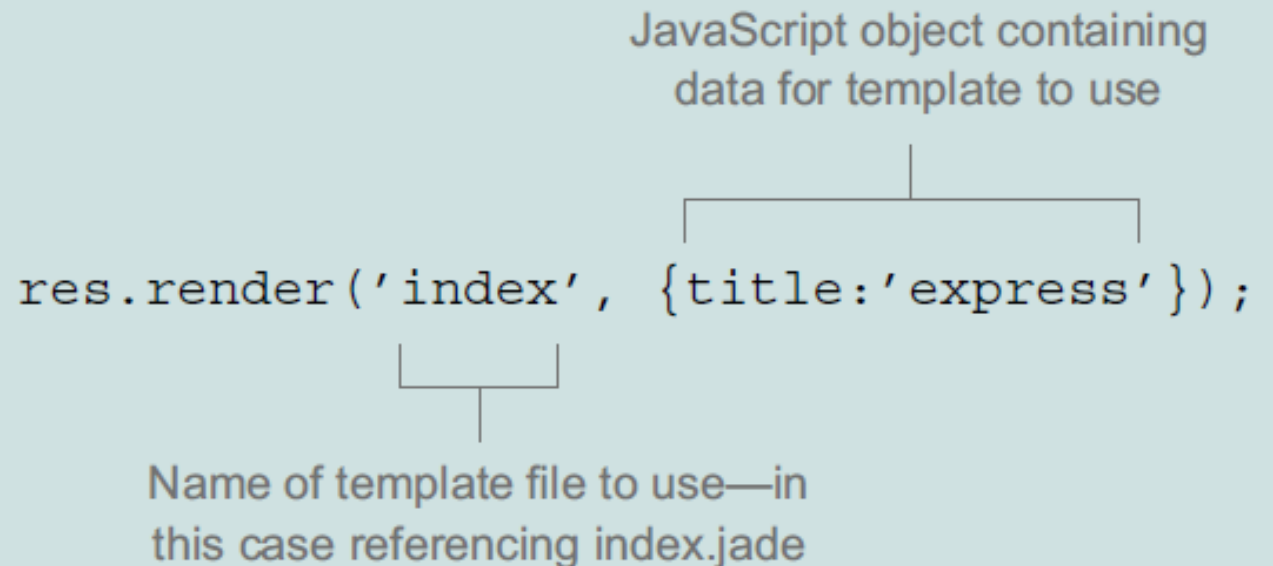
The diagram illustrates the two arguments of the `res.render` method. A bracket above the second argument, `{title:'express'}`, points to the text "JavaScript object containing data for template to use". A bracket below the first argument, `'index'`, points to the text "Name of template file to use—in this case referencing index.jade".

# EXPRESS - ROUTING

Template file doesn't need to have the file extension suffix, so `index.jade` can just be referenced as `index`. You also don't need to specify the path to the view folder because you've already done this in the main Express setup.

`res.render` is the Express function for compiling a view template to send as the HTML response that the browser will receive.

The render method takes the name of the view template and a JavaScript data object in the following construct:



```
res.render('index', {title:'express'});
```

The diagram illustrates the components of the `res.render()` method call. A bracket above the second argument, `{title:'express'}`, points to the text "JavaScript object containing data for template to use". Another bracket below the first argument, `'index'`, points to the text "Name of template file to use—in this case referencing index.jade".



# DISCUSSION