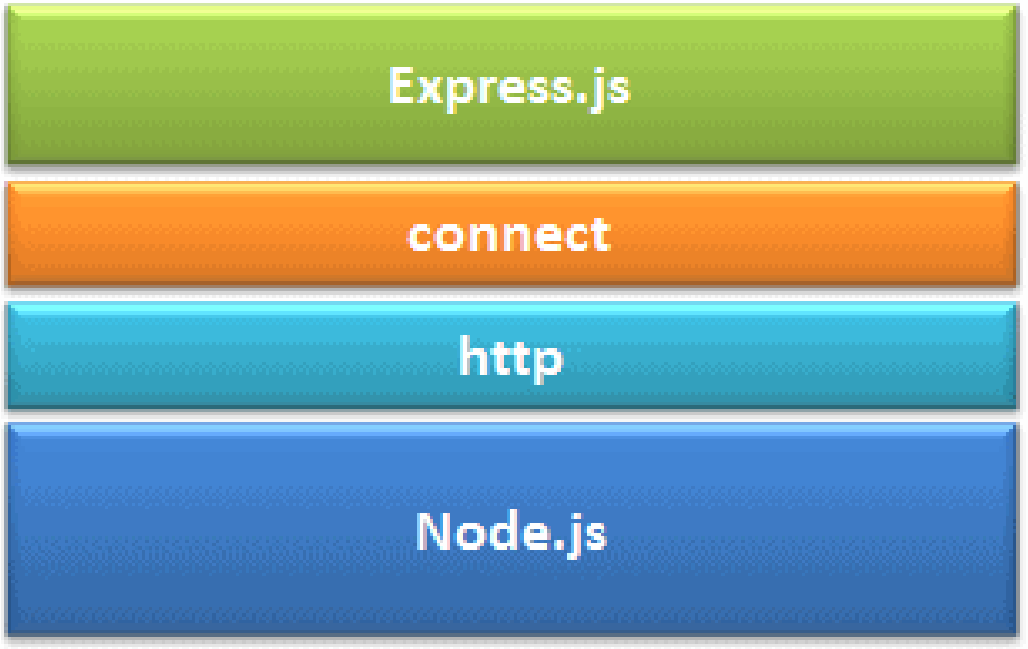
Express.js

- Introduction
- Express Generator, Static files
- Routing, HTTP Methods
- Writing Middleware, Using Middleware
- Error handling
- Cookies, Session

Express.js

- Express.js is a web application framework for Node.js.
- It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.
- Express.js is based on the Node.js middleware module called connect which in turn uses http module.
- So, any middleware which is based on connect will also work with Express.js.



Express.js

Advantages

- Makes Node.js web application development fast and easy.
- Easy to configure and customize.
- Allows you to define routes of your application based on HTTP methods and URLs.
- Includes various middleware modules which you can use to perform additional tasks on request and response.
- Easy to integrate with different template engines like Jade, Vash, EJS etc.
- Allows you to define an error handling middleware.
- Easy to serve static files and resources of your application.
- Allows you to create REST API server.
- Easy to connect with databases such as MongoDB, Redis, MySQL

Express.js

Features of Express.js

- Express quickens the development pace of a web application.
- It also helps in creating mobile and web application of single-page, multi-page, and hybrid types
- Express can work with various templating engines such as Pug, Mustache, and EJS.
- Express follows the Model-View-Controller (MVC) architecture.
- It makes the integration process with databases such as MongoDB, Redis, MySQL effortless.
- Express also defines an error-handling middleware.
- It helps in simplifying the configuration and customization steps for the application.

Express.js

- Example:

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

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

app.listen(3000);
```

Express.js

- Installation

- To install it globally, you can use the below command:

```
npm install -g express
```

- Or, if you want to install it locally into your project folder, you need to execute the below command:

```
npm install express --save
```

Express.js Routing

app.get(route, callback)

- This function tells what to do when a **get** request at the given route is called.
- The callback function has 2 parameters, ***request(req)*** and ***response(res)***.
- The request **object(req)** represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc.
- Similarly, the response object represents the HTTP response that the Express app sends when it receives an HTTP request.

res.send()

- This function takes an object as input and it sends this to the requesting client.
- Here we are sending the string *"Hello World!"*.

app.listen(port, [host], [backlog], [callback]])

- This function binds and listens for connections on the specified host and port.
- Port is the only required parameter here.

Express.js Routing

app.get(route, callback)

- This function tells what to do when a **get** request at the given route is called.

res.send()

- This function takes an object as input and it sends this to the requesting client.

app.listen(port, [host], [backlog], [callback]])

- This function binds and listens for connections on the specified host and port.

Express.js Routing

S.No.	Argument & Description
1	**port** A port number on which the server should accept incoming requests.
2	**host** Name of the domain. You need to set it when you deploy your apps to the cloud.
3	**backlog** The maximum number of queued pending connections. The default is 511.
4	**callback** An asynchronous function that is called when the server starts listening for requests.

Express.js Routing

Routing and HTTP Methods

- Routing refers to the process of determining a specific behavior of an application.

- It is used for defining how an application should respond to a client request to a particular route, path or URI along with a particular HTTP Request method. Each route can contain more than one handler functions, which is executed when the user browses for a specific route.

- Structure of Routing in Express:
  app.METHOD(PATH, HANDLER)

app is just an instance of Express.js. You can use any variable of your choice.

- METHOD is an HTTP request method such as get, set, put, delete, etc.

- PATH is the route to the server for a specific webpage

- HANDLER is the callback function that is executed when the matching route is found.

Express.js Routing

Routing and HTTP Methods

- Structure of Routing in Express:

  app.METHOD(PATH, HANDLER)

Express.js Routing

HTTP Methods

Method	Description
1. GET	The HTTP GET method helps in requesting for the representation of a specific resource by the client. The requests having GET just retrieves data and without causing any effect.
2. POST	The HTTP POST method helps in requesting the server to accept the data that is enclosed within the request as a new object of the resource as identified by the URI.
3. PUT	The HTTP PUT method helps in requesting the server to accept the data that is enclosed within the request as an alteration to the existing object which is identified by the provided URI.
4. DELETE	The HTTP DELETE method helps in requesting the server to delete a specific resource from the destination.

Express.js Middleware

- Express.js Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler.

- As the name specified, Middleware appears in the middle between an initial request and final intended route.

- In stack, middleware functions are always invoked in the order in which they are added.

- Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.

Express.js Middleware

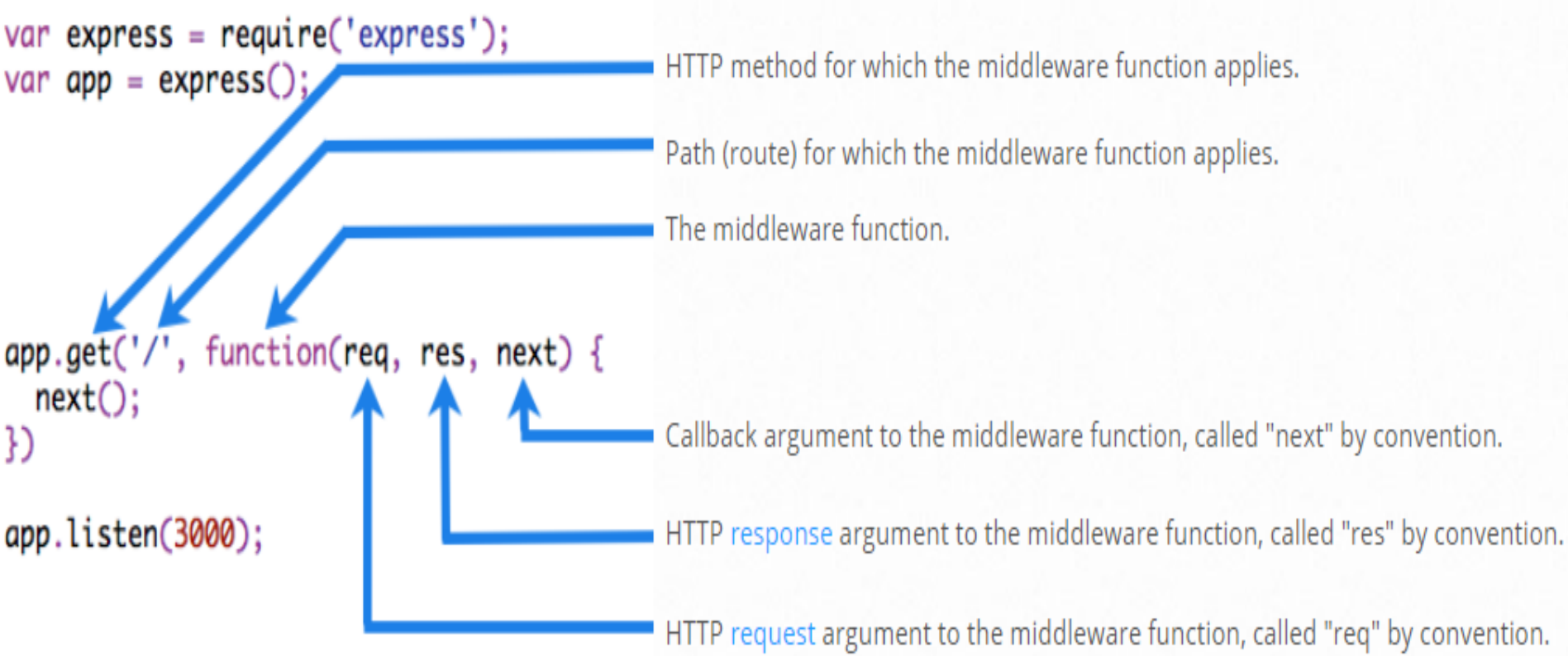
A middleware function can perform the following tasks:

- It can execute any code.
- It can make changes to the request and the response objects.
- It can end the request-response cycle.
- It can call the next middleware function in the stack.

Following is a list of possibly used middleware in Express.js app:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

Express.js Middleware



Express.js Middleware

- express.static(root, [options])

- This is a built-in middleware function in Express.

- It serves static files and is based on serve-static.

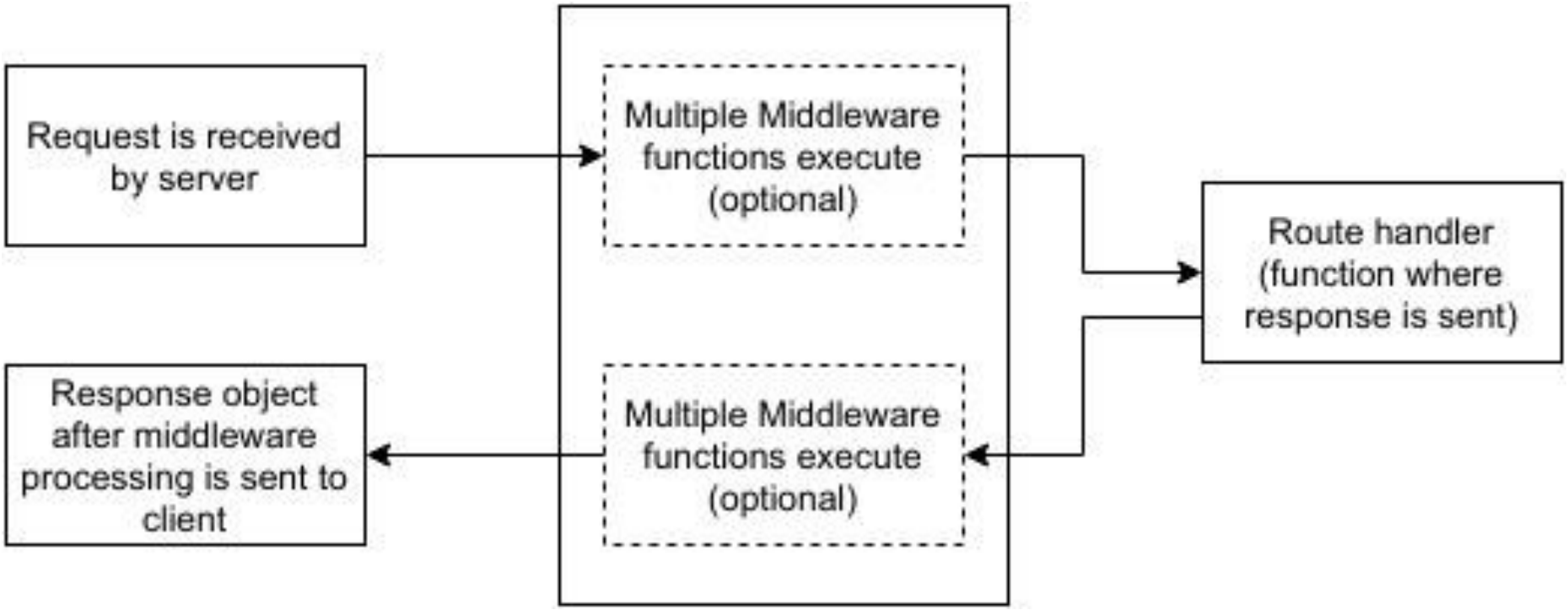
- The root argument specifies the root directory from which to serve static assets.

- The function determines the file to serve by combining req.url with the provided root directory.

- When a file is not found, instead of sending a 404 response, it instead calls next() to move on to the next middleware, allowing for stacking and fall-backs.

Express.js Middleware

The following diagram summarizes what we have learnt about middleware −



Express.js Middleware

Third Party Middleware

- body-parser:

- This is used to parse the body of requests which have payloads attached to them.

- To mount body parser, we need to install it using npm install --save body-parser and to mount it, include the following lines in your index.js −

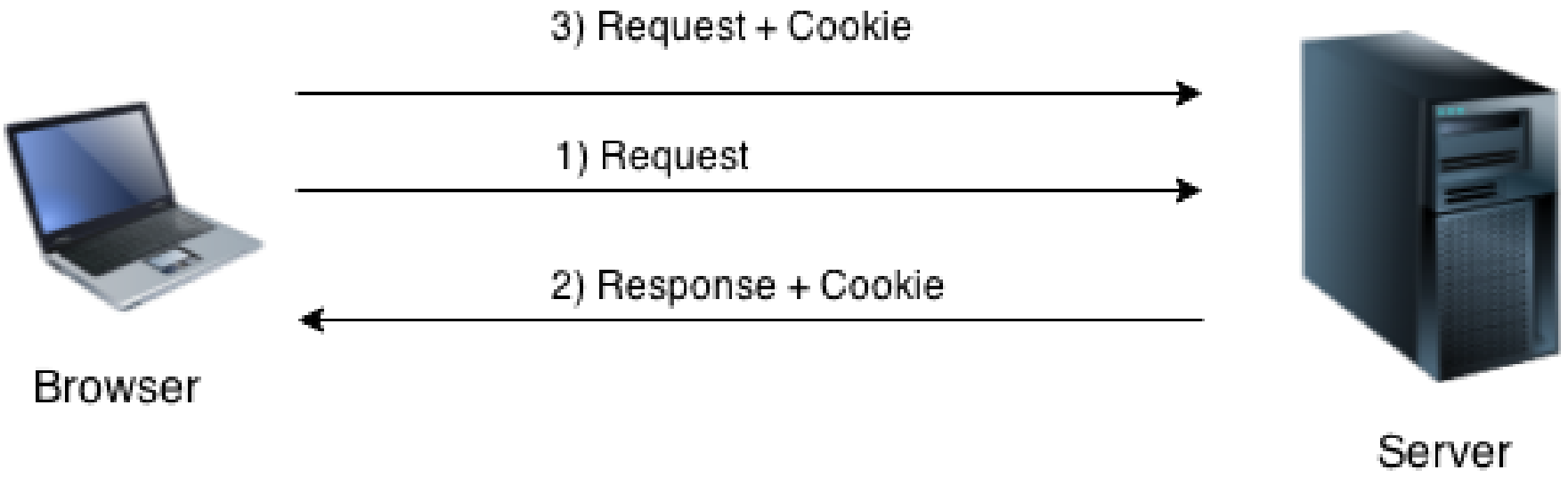
```
var bodyParser = require('body-parser');

//To parse URL encoded data
app.use(bodyParser.urlencoded({ extended: false }))

//To parse json data
app.use(bodyParser.json())
```

Express.js Cookies Management

What are cookies

Cookies are small piece of information i.e. sent from a website and stored in user's web browser when user browses that website. Every time the user loads that website back, the browser sends that stored data back to website or server, to recognize user.

cookies in express.js



Express.js Cookies Management

Third Party Middleware

- cookie-parser

- It parses Cookie header and populate req.cookies with an object keyed by cookie names.

- To mount cookie parser, we need to install it using npm install --save cookie-parser and to mount it, include the following lines in your index.js −

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

- express-session

- It creates a session middleware with the given options. We will discuss its usage in the Sessions section.

Express.js Cookies Management

Import cookie-parser into your app.

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

- cookie-parser:

Cookie-parser parses Cookie header and populate req.cookies with an object keyed by the cookie names.

```
app.get('/cookie',function(req, res){
    res.cookie('cookie_name' , 'cookie_value').send('Cookie is set');
});
app.get('/', function(req, res) {
  console.log("Cookies :  ", req.cookies);
});
```

Express.js Cookies Management

To check if your cookie is set or not, just go to your browser, fire up the console, and enter −

```
console.log(document.cookie);
```

You will get the output like (you may have more cookies set maybe due to extensions in your browser) −

```
"name = express"
```

Express.js Session Management

- Cookies and URL parameters are both suitable ways to transport data between the client and the server.
- But they are both readable and on the client side. Sessions solve exactly this problem.
- You assign the client an ID and it makes all further requests using that ID. Information associated with the client is stored on the server linked to this ID.

- npm install --save express-session

- The session middleware handles all things for us, i.e., creating the session, setting the session cookie and creating the session object in req object.

- Whenever we make a request from the same client again, we will have their session information stored with us (given that the server was not restarted).

Express.js Session Management

- **Example**:

```
app.get('/', function(req, res){
  if(req.session.page_views){
    req.session.page_views++;
    res.send("You visited this page " + req.session.page_views + " times");
  } else {
    req.session.page_views = 1;
    res.send("Welcome to this page for the first time!");
  }
```

Express.js Error Handling

- Error handling in Express is done using middleware. But this middleware has special properties.

- The error handling middleware are defined in the same way as other middleware functions, except that error-handling functions MUST have four arguments instead of three – err, req, res, next.

- For example, to send a response on any error, we can use −

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

Express.js Error Handling

- The error handling middleware allows us to separate our error logic and send responses accordingly.

- The next() method we discussed in middleware takes us to next middleware/route handler.

- For error handling, we have the next(err) function. A call to this function skips all middleware and matches us to the next error handler for that route.

- This error handling middleware can be strategically placed after routes or contain conditions to detect error types and respond to the clients accordingly.

Express.js Error Handling

```
app.get('/', function(req, res){
   //Create an error and pass it to the next function
   var err = new Error("Something went wrong");
   next(err);
});

/*
 * other route handlers and middleware here
 * ....
 */

//An error handling middleware
app.use(function(err, req, res, next) {
   res.status(500);
   res.send("Oops, something went wrong.")
});
```

Express.js Debugging

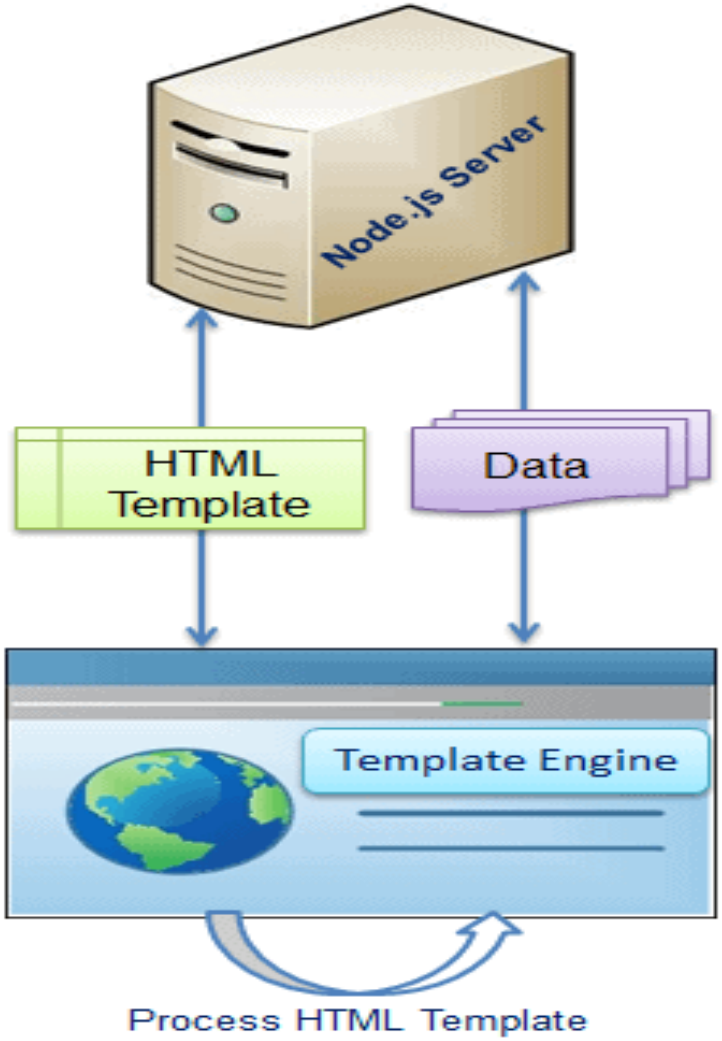
Express uses the Debug module to internally log information about route matching, middleware functions, application mode, etc.

To see all internal logs used in Express, set the DEBUG environment variable to Express:* when starting the app −

DEBUG = express:* node index.js

Express.js Template Engine for Node.js

Template engine helps us to create an HTML template with minimal code. Also, it can inject data into HTML template at client side and produce the final HTML



Express.js Template Engine for Node.js

- Client-side browser loads HTML template, JSON/XML data and template engine library from the server.

- Template engine produces the final HTML using template and data in client's browser. However, some HTML templates process data and generate final HTML page at server side also.

There are many template engines available for Node.js. Each template engine uses a different language to define HTML template and inject data into it.

The following is a list of important (but not limited) template engines for Node.js

Jade
Vash
EJS
Mustache
Dust.js
Nunjucks
Handlebars

Express.js Template Engine for Node.js

Advantages of Template engine in Node.js

- Improves developer's productivity.
- Improves readability and maintainability.
- Faster performance.
- Maximizes client side processing.
- Single template for multiple pages.
- Templates can be accessed from CDN (Content Delivery Network).

Express.js Template Engine for Node.js

Using template engines with Express:

Template engine makes you able to use static template files in your application.

To render template files you have to set the following application setting properties:

Views: It specifies a directory where the template files are located.
For example: app.set('views', './views').

view engine: It specifies the template engine that you use. For example, to use the Pug template engine: app.set('view engine', 'pug').

Express.js Template Engine for Node.js

- Jade Template Engine

- Jade is a template engine for Node.js. Jade syntax is easy to learn.
- It uses whitespace and indentation as a part of the syntax.
- Install jade into your project using NPM as below.

npm install jade

- Jade template must be written inside .jade file. And all .jade files must be put inside views folder in the root folder of Node.js application.

Note: By default Express.js searches all the views in the views folder under the root folder, which can be set to another folder using views property in express e.g. app.set('views','MyViews').

Express.js Template Engine for Node.js

- Jade Engine with Express.js

Express.js can be used with any template engine.
Here, we will use different Jade templates to create HTML pages dynamically.

- In order to use Jade with Express.js, create sample.jade file inside views folder and write following Jade template in it.

- Sample.jade :

```
doctype html
html
    head
        title Jade Page
    body
        h1 This page is produced by Jade engine
        p some paragraph here..
```

Express.js Template Engine for Node.js

- Jade Engine with Express.js

server.js:

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

//set view engine
app.set("view engine","jade")

app.get('/', function (req, res) {

    res.render('sample');

});

var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

Express.js Template Engine for Node.js

- Jade Engine with Express.js

- first we import express module and then set the view engine using app.set() method.

- The set() method sets the "view engine", which is one of the application setting property in Express.js.

- In the HTTP Get request for home page, it renders sample.jade from the views folder using res.render() method.

References:

- https://www.tutorialspoint.com/expressjs/expressjs_routing.htm
- https://www.tutorialspoint.com/nodejs/nodejs_first_application.htm
- https://expressjs.com/en/guide/writing-middleware.html