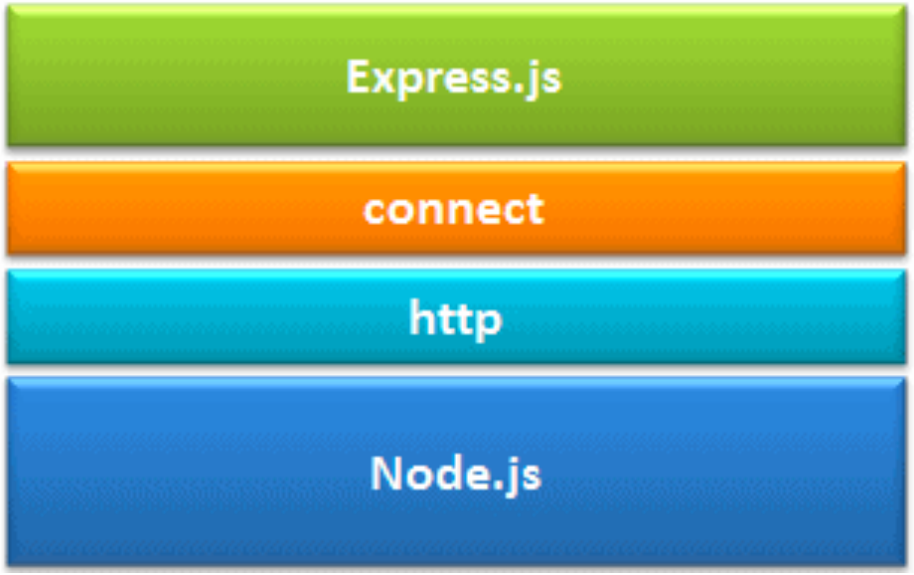
COMP 3123

Frameworks for Node JS

- There are various third party open-source frameworks available in Node Package Manager which makes Node.js application development faster and easy.
- You can choose an appropriate framework as per your application requirements.

Open-Source Framework	Description
Express.js	Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. This is the most popular framework as of now for Node.js.
Geddy	Geddy is a simple, structured web application framework for Node.js based on MVC architecture.
Locomotive	Locomotive is MVC web application framework for Node.js. It supports MVC patterns, RESTful routes, and convention over configuration, while integrating seamlessly with any database and template engine. Locomotive builds on Express, preserving the power and simplicity you've come to expect from Node.
Koa	Koa is a new web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.
Total.js	Totaljs is free web application framework for building web sites and web applications using JavaScript, HTML and CSS on Node.js

What is Express JS Framework



- Express is a minimal and flexible Node.js web application framework.
- It provides a robust set of features to develop web and mobile applications.
- It facilitates the rapid development of Node based Web applications.
- Following are some of the core features of Express framework −
  - Allows to set up middleware's to respond to HTTP Requests.
  - Defines a routing table which is used to perform different actions based on HTTP Method and URL.
  - Allows to dynamically render HTML Pages based on passing arguments to templates.

Installing Express

- Install the Express framework globally using NPM so that it can be used to create a web application using node terminal.

$ npm install –g express

OR

$ npm install express –save

The above command saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules.

Expressjs.com

Express-Generator

Advantages of Express JS

- 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

Other Modules with Express JS

You should install the following important modules along with express −

- **body-parser** − This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
- **cookie-parser** − Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- **multer** − This is a node.js middleware for handling multipart/form-data.

- **Installation**

```
$ npm install body-parser --save
$ npm install cookie-parser --save
$ npm install multer --save
```

Example - Express JS

```
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)
})
```

- Imported Express.js module using require() function.
- The express module returns a function.
- This function returns an object which can be used to configure Express application (***app in the above example***).
- *The app object includes methods for routing HTTP requests, configuring middleware, rendering HTML views and registering a template engine.*
- The app.listen() function creates the Node.js web server at the specified host and port.
- Run the above example using node app.js command and point your browser to ***http://localhost:8081***.

Introduction to RESTful API

Method	URI	Details	Function
GET	/movies	Safe, cachable	Gets the list of all movies and their details
GET	/movies/1234	Safe, cachable	Gets the details of Movie id 1234
POST	/movies	N/A	Creates a new movie with the details provided. Response contains the URI for this newly created resource.
PUT	/movies/1234	Idempotent	Modifies movie id 1234(creates one if it doesn't already exist). Response contains the URI for this newly created resource.
DELETE	/movies/1234	Idempotent	Movie id 1234 should be deleted, if it exists. Response should contain the status of the request.
DELETE or PUT	/movies	Invalid	Should be invalid. **DELETE** and **PUT** should specify which resource they are working on.

- An API is always needed to create mobile applications, single page applications to provide data to clients.
- An popular architectural style of how to structure and name these APIs and the endpoints is called **REST (Representational Transfer State)**.
- REST was introduced by **Roy Fielding** in 2000 in his Paper Fielding Dissertations.
- RESTful URIs and methods provide us with almost all information we need to process a request.
- The table given summarizes how the various verbs should be used and how URIs should be named.

S.No.	Method & Description
1	**GET** The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
2	**POST** The POST method requests that the server accept the data enclosed in the request as a new object/entity of the resource identified by the URI.
3	**PUT** The PUT method requests that the server accept the data enclosed in the request as a modification to existing object identified by the URI. If it does not exist then the PUT method should create one.
4	**DELETE** The DELETE method requests that the server delete the specified resource.

What are Routes?

- Routing determine the way in which an application responds to a client request to a particular endpoint.
- For example, a client can make a GET, POST, PUT or DELETE http request for various URL such as the ones shown below;
  - *http://localhost:3000/Books*
  - *http://localhost:3000/Students*
- If a GET request is made for the first URL, then the response should ideally be a list of books.
- If the GET request is made for the second URL, then the response should ideally be a list of Students.
- ***So based on the URL which is accessed, a different functionality on the webserver will be invoked, and accordingly, the response will be sent to the client. This is the concept of routing.***

Syntax : `app.METHOD(PATH, HANDLER)`

Web Reference

Configuring Routes using Express JS

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

app.get('/', function (req, res) {
    res.send('<html><body><h1>Hello World</h1></body></html>');
});

app.post('/submit-data', function (req, res) {
    res.send('POST Request');
});

app.put('/update-data', function (req, res) {
    res.send('PUT Request');
});

app.delete('/delete-data', function (req, res) {
    res.send('DELETE Request');
});

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

- Use app object to define different routes of your application.
- The app object includes get(), post(), put() and delete() methods to define routes for HTTP GET, POST, PUT and DELETE requests respectively.
- The first parameter is a path of a route which will start after base URL.

http://expressjs.com/en/4x/api.html

Using Routes (Get)

- We have seen a basic application which serves HTTP request for the homepage. Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Here is a simple example which passes two values using **HTML FORM GET** method. We are going to use **process_get** router inside server.js to handle this input.

```
<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">
      <input type = "submit" value = "Submit">
    </form>

  </body>
</html>
```

```
var express = require('express');
var app = express();
app.use(express.static('public'));
app.get('/index.htm', function (req, res) {
  res.sendFile( __dirname + "/" + "index.htm" );
})
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.end(JSON.stringify(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)
})
```

Extracting Query and Route Parameters

- **req.query:** directly access the parsed query string parameters
- **req.params:** directly access the parsed route parameters from the path

- A query *string* is the part of a URL (Uniform Resource Locater) *after* the question mark (?).

  https://stackabuse.com?page=2&limit=3

- The query *parameters* are the actual key-value pairs like page and limit with values of 2 and 3, respectively.

```
app.get('/', function(req, res)
{
        let page = req.query.page;
        let limit = req.query.limit;
});
```

- In any web application another common way to structure your URLs is to place information within the actual URL path, which are simply called route parameters in Express.

  https://stackabuse.com/tag/pritam

- We tell Express that our route is **/tag/:id**, where **:id** is a placeholder for anything. It could be a string or a number.
- So whatever is passed in that part of the path is set as the id parameter.

```
app.get('/tag/:id', function(req, res)
{
        let page = req.params.id;
});
```

Query-strings-and-parameters

Working with route params & query string

- The parameters of **router.param()** are **name** and a **function**.
- Where **name** is the actual name of parameter and **function** is the callback function.
- Basically router.param() function triggers the callback function whenever user routes to the parameter.
- This callback function will be called for only single time in request response cycle, even if user routes to the parameter multiple times.

Syntax:

router.param(name, function)

Parameters of callback function are:

- *req – the request object*
- *res – the response object*
- *next – the next middleware function*
- *id – the value of name parameter*

Web Reference Link

Thank you