

Instructor Notes:

Add instructor notes here.

Express JS

Lesson 03 :Working with
Express.js

Capgemini

Instructor Notes:

Add instructor notes here.

Lesson Objectives

Introduction
Introduction to Express.js
Connect Module
Express.js Installation
app.js
Steps for creating Express.js Application
application, request, response object properties & methods
Request-params,body,files,route,header,get
Response-render,locals,status,json,redirect
Types of middleware
Application level middleware



Instructor Notes:

Add instructor notes here.

Lesson Objectives

Express-json,session,logger,compress
Router level middleware
Built-in middleware
Third party middleware
Express 4.0 Router
Express.js Scaffolding
Working with MongoDB



Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Introduction**

- If we try to create apps by only using core Node.js modules we will end up by writing the same code repeatedly for similar tasks such as
 - Parsing of HTTP request bodies
 - Parsing of cookies
 - Managing sessions
 - Organizing routes with a chain of if conditions based on URL paths and HTTP methods of the requests
 - Determining proper response headers based on data types
- Developers have to do a lot of manual work themselves, such as interpreting HTTP methods and URLs into routes, and parsing input and output data.
- Express.js solves these and many other problems using abstraction and code organization.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Introduction to Express.js**

- Express.js is a web framework based on the core Node.js http module and Connect components
- Express.js framework provides a model-view-controller-like structure for your web apps with a clear separation of concerns (views, routes, models)
- Express.js systems are highly configurable, which allows developers to pick freely whatever libraries they need for a particular project
- Express.js framework leads to flexibility and high customization in the development of web applications.
- In Express.js we can define middleware such as error handlers, static files folder, cookies, and other parsers.
- Middleware is a way to organize and reuse code, and, essentially, it is nothing more than a function with three parameters: request, response, and next.

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
Connect module

➤ Connect is a module built to support interception of requests in a modular approach.

```
var connect = require('connect');
var app = connect();
var logger = function(req, res, next) {
  console.log(req.method, req.url);
  next();
};
var helloWorld = function(req, res, next) {
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
};
var byeWorld = function(req, res, next) {
  res.setHeader('Content-Type', 'text/plain');
  res.end('Bye World');
};
app.use(logger);
app.use('/hello',helloWorld);
app.use('/bye',byeWorld);
app.listen(3000);
console.log('Server running at localhost:3000');
```

July 31, 2018

Proprietary and Confidential

- 6 -

We can build a simple web server using the http module. Connect creates an API to write code that manages the different HTTP requests sent to your server, handles them properly, and responds to each request with the correct response.

Connect uses a modular component called middleware, which allows you to simply register your application logic to predefined HTTP request scenarios. Connect middleware are basically callback functions, which get executed when an HTTP request occurs

Connect middleware

Connect middleware is just JavaScript function with a unique signature.

Each middleware function is defined with the following three arguments:

req: This is an object that holds the HTTP request information

res: This is an object that holds the HTTP response information and allows you to set the response properties

next: This is the next middleware function defined in the ordered set of Connect middleware.

The way a Connect application works is by using an object called dispatcher. The dispatcher object handles each HTTP request received by the server and then decides, in a cascading way, the order of middleware execution.

Connect middleware function will be executed in first-in-first-out (FIFO) order using the next arguments until there are no more middleware functions to execute or the next middleware function is not called.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Express.js Installation**

- The Express.js package comes in two flavors:
 - *express-generator*: a global NPM package that provides the command-line tool for rapid app creation (scaffolding)
 - *express*: a local package module in your Node.js app's node_modules folder

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
app.js

➤ App.js is the main file in Express framework. A typical structure of the main Express.js file consists of the following areas

- 1. Require dependencies
- 2. Configure settings
- 3. Connect to database (optional)
- 4. Define middleware
- 5. Define routes
- 6. Start the server

➤ The order here is important, because requests travel from top to bottom in the chain of middleware.

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
app.js

- First the dependencies need to be included with require()

```
var express = require('express');
var http = require('http');
var path = require('path');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
```

- Then Express.js object is instantiated (Express.js uses a functional pattern):

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

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
app.js

- One of the ways to configure Express.js settings is to use `app.set()`, with the name of the setting and the value.

```
const express=require('express');
const path=require('path');
//init app
const app=express();

//Load View Engine
app.set('views',path.join(__dirname,'views'));
app.set('view engine','pug');
```

- Middleware is the backbone of the Express.js framework and it comes in two flavors.

- Defined in external (third-party) modules, such as `bodyParser.json` from Connect/Express.js body-parser: `app.use(bodyParser.json())`;
- Defined in the app or its modules, such as `app.use(function(req, res, next){...})`;

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
app.js

➤ Routes are processed in the order they are defined. Usually, routes are put after middleware, but some middleware may be placed following the routes. A good example of such middleware, found after a routes, is error handler.

➤ The way routes are defined in Express.js is with helpers app.VERB(url, fn1, fn2, ..., fn), where fnNs are request handlers, url is on a URL pattern in RegExp, and VERB values are as follows:

- all: catch every request (all methods)
- get: catch GET requests
- post: catch POST requests
- put: catch PUT requests
- del: catch DELETE requests

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
app.js

➤ Finally to start the server, we need to use `createServer` method from the core `http` module. In this method, the system passes the Express.js app object with all the settings and routes

```
http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js

Steps for creating Express.js Application

- Step – 1 : Create a folder named SampleApp
- Step – 2 : Create package.json with the following schema

```
{ "name": "node",
  "version": "1.0.0",
  "description": "", "main":
  "app.js", "scripts": {
    "start": "node app" },
  "author": "Rahul Vikash",
  "license": "ISC",
  "dependencies": { "body-
    parser": "^1.18.2",
    "express": "^4.16.3",
    "mongoose": "^5.0.16",
    "nodeman": "^1.1.2",
    "pug": "^2.0.3" }}
```

Instructor Notes:

Add instructor notes here.

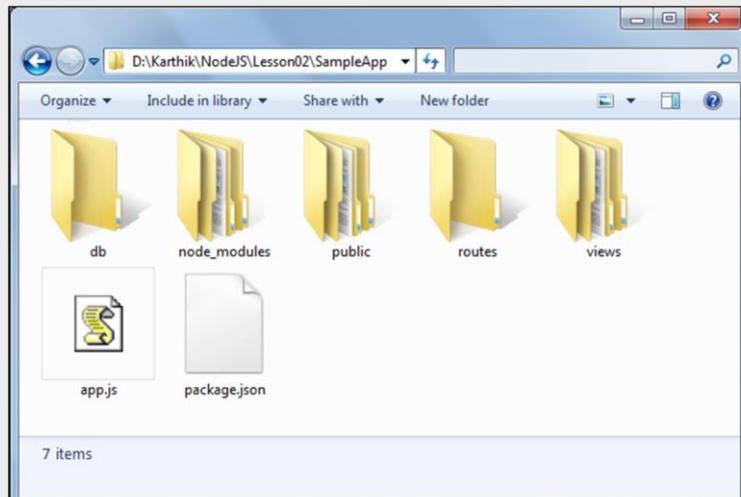
3.1 : Working with Express.js

Steps for creating Express.js Application

- Step – 3 : Install the dependencies using npm install command
- Step – 4 : Create the following folders under SampleApp folder
 - **public** : All the static (front-end) files like HTML
 - **public/css** : Stylesheet files
 - **public/img** : images
 - **public/js** : Scripts
 - **db** : Seed data and scripts for MongoDB
 - **views** : Jade/pug (or any other template engine) files
 - **views/includes** : Partial / include files
 - **routes** : Node.js modules that contain request handlers
- Step – 5 : Create the main file named app.js

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
Steps for creating Express.js Application

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js

Steps for creating Express.js Application

➤ Step – 6 : Type the following contents in app.js

```
const express=require('express');
const path=require('path');
//init app
const app=express();

//Load View Engine
app.set('views',path.join(__dirname,'views'));
app.set('view engine','pug');
```

July 31, 2018

Proprietary and Confidential

- 16 -

Routes in Express.js

```
// respond with "Hello World!" on the homepage
app.get('/', function (req, res) {
  res.send('Hello World!');
}

// accept POST request on the homepage
app.post('/', function (req, res) {
  res.send('Got a POST request');
}

// accept PUT request at /user
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
}

// accept DELETE request at /user
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user');
})
```

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js

Steps for creating Express.js Application

- Step – 7 : Create index.jade under views folder and type the following contents

```
doctype html
html
  head
    title= title
  body
    h1= title
    p Welcome to #{title}
```

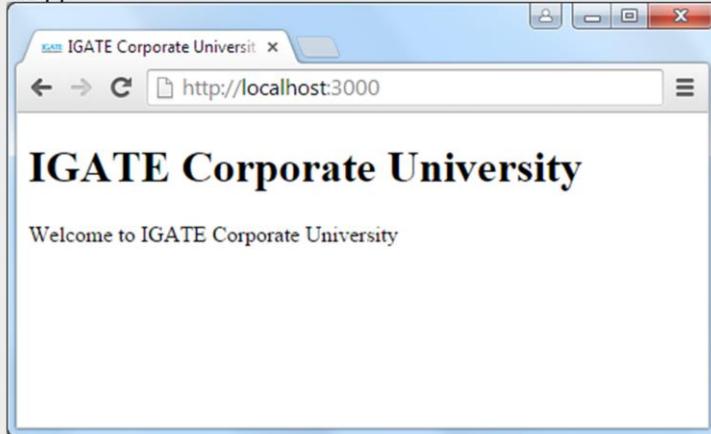
- Step – 8 : Start the app by typing *npm start* in command prompt.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Steps for creating Express.js Application**

➤ Step – 9 : Open browser and type `http://localhost:3000` to view the SampleApp



July 31, 2018

Proprietary and Confidential

- 18 -

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
application object properties & methods**

Property/Method	Description
app.set(name, value)	Sets app-specific properties
app.get(name)	Retrieves value set by app.set()
app.enable(name)	Enables a setting in the app
app.disable(name)	Disables a setting in the app
app.enabled(name)	Checks if a setting is enabled
app.disabled(name)	Checks if a setting is disabled
app.configure([env], callback)	Sets app settings conditionally based on the development environment
app.use([path], function)	Loads a middleware in the app
app.engine(ext, callback)	Registers a template engine for the app
app.param([name], callback)	Adds logic to route parameters
app.VERB(path, [callback...], callback)	Defines routes and handlers based on HTTP verbs
app.all(path, [callback...], callback)	Defines routes and handlers for all HTTP verbs
app.locals	The object to store variables accessible from any view
app.render(view, [options], callback)	Renders view from the app
app.routes	A list of routes defined in the app
app.listen()	Binds and listen for connections

July 31, 2018

Proprietary and Confidential

- 19 -

The application object

The application object is an instance of Express, conventionally represented by the variable named app. This is the main object of your Express app and the bulk of the functionality is built on it.

This is how you create an instance of the Express module:

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

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
request object properties & methods

Property/Method	Description
req.params	Holds the values of named routes parameters
req.params(name)	Returns the value of a parameter from named routes or GET params or POST params
req.query	Holds the values of a GET form submission
req.body	Holds the values of a POST form submission
req.files	Holds the files uploaded via a form
req.route	Provides details about the current matched route
req.cookies	Cookie values
req.signedCookies	Signed cookie values
req.get(header)	Gets the request HTTP header
req.accepts(types)	Checks if the client accepts the media types
req.accepted	A list of accepted media types by the client
req.is(type)	Checks if the incoming request is of the particular media type

July 31, 2018

Proprietary and Confidential

- 21 -

The request object

The HTTP request object is created when a client makes a request to the Express app. The object is conventionally represented by a variable named req, which contains a number of properties and methods related to the current request.

req.query: This is an object containing the parsed query-string parameters.

req.params: This is an object containing the parsed routing parameters.

req.body: This is an object used to retrieve the parsed request body. This property is included in the bodyParser() middleware.

req.param(name): This is used to retrieve a value of a request parameter. Note that the parameter can be a query-string parameter, a routing parameter, or a property from a JSON request body.

req.path, req.host, and req.ip: These are used to retrieve the current request path, host name, and remote IP.

req.cookies: This is used in conjunction with the cookieParser() middleware to retrieve the cookies sent by the user-agent.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
request object properties & methods**

Property/Method	Description
req.ip	The IP address of the client
req.ips	The IP address of the client, along with that of the proxies it is connected through
req.stale	Checks if the request is stale
req.xhr	Checks if the request came via an AJAX request
req.protocol	The protocol used for making the request
req.secure	Checks if it is a secure connection
req.subdomains	Subdomains of the host domain name
req.url	The request path, along with any query parameters
req.originalUrl	Used as a backup for req.url
req.acceptedLanguages	A list of accepted languages by the client
req.acceptsLanguage (langauge)	Checks if the client accepts the language
req.acceptedCharsets	A list of accepted charsets by the client
req.acceptsCharsets (charset)	Checks if the client accepts the charset
req.host	Hostname from the HTTP header

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
response object properties & methods**

Property/Method	Description
<code>res.status(code)</code>	Sets the HTTP response code
<code>res.set(field, [value])</code>	Sets response HTTP headers
<code>res.get(header)</code>	Gets the response HTTP header
<code>res.cookie(name, value, [options])</code>	Sets cookie on the client
<code>res.clearCookie(name, [options])</code>	Deletes cookie on the client
<code>res.redirect([status], url)</code>	Redirects the client to a URL, with an optional HTTP status code
<code>res.location</code>	The location value of the response HTTP header
<code>res.charset</code>	The charset value of the response HTTP header
<code>res.send([body status], [body])</code>	Sends an HTTP response object, with an optional HTTP response code
<code>res.json([status body], [body])</code>	Sends a JSON object for HTTP response, along with an optional HTTP response code

July 31, 2018

Proprietary and Confidential

- 23 -

The response object

The response object is created along with the request object, and is conventionally represented by a variable named res. While it may sound a little strange that both of them should be created together, it is a necessity to give all the middlewares a chance to work on the request and the response object, before passing the control to the next middleware.

- **`res.status(code)`:** This is used to set the response HTTP status code.
- **`res.set(field, [value])`:** This is used to set the response HTTP header.
- **`res.cookie(name, value, [options])`:** This is used to set a response cookie. The options argument is used to pass an object defining common cookie configuration, such as the `maxAge` property.
- **`res.redirect([status], url)`:** This is used to redirect the request to a given URL. Note that you can add an HTTP status code to the response. When not passing a status code, it will be defaulted to 302 Found.
- **`res.send([body|status], [body])`:** This is used for non-streaming responses. This method does a lot of background work, such as setting the Content-Type and Content-Length headers, and responding with the proper cache headers.
- **`res.json([status|body], [body])`:** This is identical to the `res.send()` method when sending an object or array. Most of the times, it is used as syntactic sugar, but sometimes you may need to use it to force a JSON response to non-objects, such as null or undefined.
- **`res.render(view, [locals], callback)`:** This is used to render a view and send an HTML response.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
response object properties & methods**

Property/Method	Description
<code>res.jsonp([status body], [body])</code>	Sends a JSON object for HTTP response with JSONP support, along with an optional HTTP response code
<code>res.type(type)</code>	Sets the media type HTTP response header
<code>res.format(object)</code>	Sends a response conditionally, based on the request HTTP Accept header
<code>res.attachment([filename])</code>	Sets response HTTP header Content-Disposition to attachment
<code>res.sendfile(path, [options], [callback])</code>	Sends a file to the client
<code>res.download(path, [filename], [callback])</code>	Prompts the client to download a file
<code>res.links(links)</code>	Sets the HTTP Links header
<code>res.locals</code>	The object to store variables specific to the view rendering a request
<code>res.render(view, [locals], callback)</code>	Renders a view

Express can send an HTTP response using one of its response methods: `res.send()`, `res.json()`, `res.jsonp()`, `res.sendFile()`, `res.download()`, `res.render()`, `res.redirect()`. If none of them is called, the request will be left hanging till the connection times out

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
How Express.js works**

➤ Express.js usually has an entry point aka, a main file. Most of the time, this is the file that we start with the node command or export as a module. In the main file we do the following

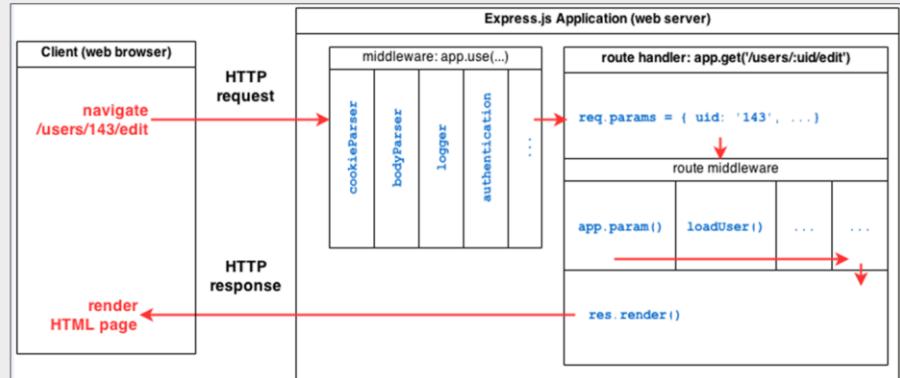
- Include third-party dependencies as well as our own modules, such as controllers, utilities, helpers, and models
- Configure Express.js app settings such as template engine and its file extensions
- Connect to databases such as MongoDB, Redis, or MySQL (optional)
- Define middlewares and routes
- Start the app and Export the app as a module (optional)

➤ When the Express.js app is running, it's listens to requests. Each incoming request is processed according to a defined chain of middleware and routes, starting from top to bottom.

➤ Routes / middleware that are higher in the file have precedence over the lower definitions.

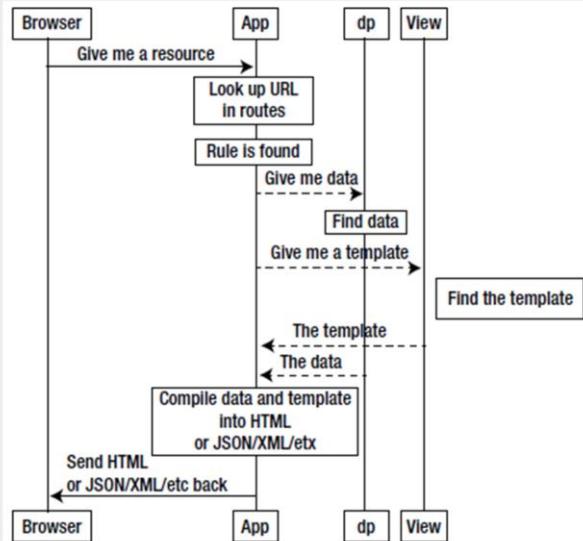
Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
How Express.js works

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
Request flow in Express

Instructor Notes:

Add instructor notes here.

**2.4 : Working with Express.js
Request flow in Express**

- In Express server the request flow will be :
 - Route → Route Handler → Template → HTML
- The route defines the URL schema. It captures the matching request and passed on control to the corresponding route handler
- The route handler processes the request and passes the control to a template.
- The template constructs the HTML for the response and sends it to the browser.
- The route handler need not always pass the control to a template, it can optionally send the response to a request directly.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Using middleware**

- A middleware is a JavaScript function to handle HTTP requests to an Express app.
- It can manipulate the request and the response objects or perform an isolated action, or terminate the request flow by sending a response to the client, or pass on the control to the next middleware.
- A middleware can:
 - Execute any code.
 - Make changes to the request and the response objects.
 - End the request-response cycle.
 - Call the next middleware in the stack.
- If the current middleware does not end the request-response cycle, it must call `next()` to pass control to the next middleware, otherwise the request will be left hanging.

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
Types of middleware

- An Express application can use the following kinds of middleware:
- Application-level middleware
- Router-level middleware
- Built-in middleware
- Third-party middleware

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Application level middleware**

- Application level middleware are bound to an instance of express, using app.use() and app.VERB().

```
var app = express();
/* a middleware with no mount path; gets executed for every request to the app */
app.use(function (req, res, next) {
    console.log('Time:', Date.now());
    next();
});
/* a middleware mounted on /user/:id; will be executed for any type of HTTP request to
 /user/:id */
app.use('/user/:id', function (req, res, next) {
    console.log('Request Type:', req.method);
    next();
});
/* a route and its handler function (middleware system) which handles GET requests to
 /user/:id */
app.get('/user/:id', function (req, res, next) {
    res.send('USER');
});
```

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Router level middleware**

- Router level middleware are loaded using router.use() and router.VERB()

```
var app = express();
var router = express.Router();
/* a middleware with no mount path, gets executed for every request to the router */
router.use(function (req, res, next) {
    console.log('Time!', Date.now());
    next();
});
/* a middleware sub-stack which handles GET requests to /user/:id */
router.get('/user/:id', function (req, res, next) {
    // if user id is 0, skip to the next router
    if (req.params.id == 0) next('route');
    // else pass the control to the next middleware in this stack
    else next(); //
}, function (req, res, next) {
    // render a regular page
    res.render('regular');
});
/* mount the router on the app */
app.use('/', router);
```

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
External middleware**

- The Express core is minimal, yet the team behind it provides various predefined middleware to handle common web development features.
- Those middleware vary in size and functionality and extend Express to provide a better framework support.
- The popular Express middleware are as follows:
 - **morgan**: This is an HTTP request logger middleware.
 - **body-parser**: This is a body-parsing middleware that is used to parse the request body, and it supports various request types.
 - **method-override**: This is a middleware that provides HTTP verb support such as PUT or DELETE in places where the client doesn't support it.
 - **cookie-parser**: This is a cookie-parsing middleware that populates the req.cookies object.
 - **express-session**: This is a session middleware used to support persistent sessions.

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Built-in middleware**

- Except for express.static, all of Express' previously included middleware are now in separate repo.
- express.static is based on serve-static, and is responsible for serving the static assets of an Express application. We can have more than one static directory per app.

```
var options = {  
  dotfiles: 'ignore',  
  etag: false,  
  extensions: ['htm', 'html'],  
  index: false,  
  maxAge: '1d',  
  redirect: false,  
  setHeaders: function (res, path, stat) {  
    res.set('x-timestamp', Date.now())  
  }  
};  
app.use(express.static('public', options));
```

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Third-party middleware**

- Express is a routing and middleware web framework with minimal functionality of its own. Functionality to Express apps are added via third-party middleware.
- Install the node module for the required functionality and loaded it in your app at the application level or at the router level.

```
$ npm install cookie-parser
```

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

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

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Express 4.0 Router**

- Express 4.0 comes with the new Router.
- Router is like a mini express application. It doesn't bring in views or settings, but provides us with the routing APIs like .use, .get, .param, and route.
- Creating instance of Router for application frontend routes

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

```
// home page route (http://localhost:3000)
router.get('/', function(req, res) {
  res.send('Home page!');
});

// about page route (http://localhost:3000/about)
router.get('/about', function(req, res) {
  res.send('About page!');
});

// apply the routes to our application
app.use('/', router);
```

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
Express 4.0 Router

- Route middleware in Express is a way to do something before a request is processed.

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

// route middleware that will happen on every request
router.use(function(req, res, next) {

    // log each request to the console
    console.log(req.method, req.url);

    // continue doing what we were doing and go to the route
    next();
});

// apply the routes to our application
app.use('/', router);
```

Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js
Express 4.0 Router

➤Route with parameters & Route Middleware for parameter

```
var router = express.Router();
// route middleware to validate :name
router.param('name', function(req, res, next, name) {
  // do validation on name here
  console.log('doing name validations on ' + name);

  // once validation is done save the new item in the req
  req.name = name;
  // go to the next thing
  next();
});

// route with parameters (http://localhost:3000/hello/:name)
router.get('/hello/:name', function(req, res) {
  res.send('hello ' + req.params.name + '!');
});

// apply the routes to our application
app.use('/', router);
```

Instructor Notes:

Add instructor notes here.

**3.1 : Working with Express.js
Express.js Scaffolding**

➤ To generate a application skeleton for Express.js app, we need to run a terminal command express [options] [dir | appname] the options for which are the following:

- -e, --ejs: add EJS engine support (by default, Jade is used)
- -c <engine>, --css <engine>: add stylesheet <engine> support, such as LESS, Stylus or Compass (by default, plain CSS is used)
- -f, --force: force app generation on a nonempty directory

```
b:\Karthik\NodeJS\Lesson02\SampleApp>express -e -f
create : ./package.json
create : ./server.js
create : ./views
create : ./views/index.ejs
create : ./views/error.ejs
create : ./bin
create : ./bin/www
create : ./routes
create : ./routes/index.js
create : ./routes/users.js
create : ./public/images
create : ./public
create : ./public/javascripts
create : ./public/stylesheets
create : ./public/stylesheets/style.css

install dependencies:
$ cd .
&& npm install

run the app:
$ DEBUG=SampleApp ./bin/www
```

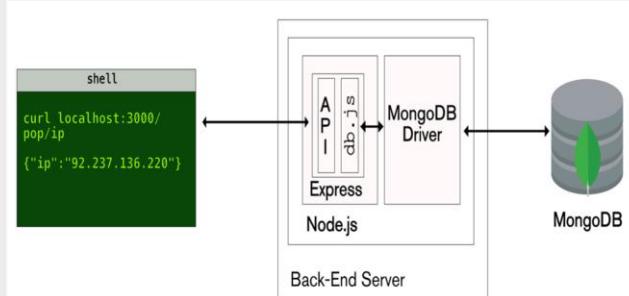
July 31, 2018

Proprietary and Confidential

- 39 -

Instructor Notes:

Add instructor notes here.

3.2 : Express with Mongo db
Express With Mongo Db**Working with Express with MongoDB**

MongoDB is a database. This is the place where you store information for your web websites (or applications).

Instructor Notes:

Add instructor notes here.

3.2 : Express with Mongo db
Express With Mongo Db

Connection with MongoDB

```
const mongoose=require('mongoose');
var bodyParser = require('body-parser');
//Connecting Mongodb server
mongoose.connect('mongodb://localhost/nodekb');
```

Instructor Notes:

Add instructor notes here.

3.2 : Express with Mongo db
Express With Mongo Db



CRUD is an acronym for Create, Read, Update and Delete. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE respectively). This is what each operation does:

Create (POST) - Make something
Read (GET) - Get something
Update (PUT) - Change something
Delete (DELETE) - Remove something

Instructor Notes:

Add instructor notes here.

Demo

Express02
express03-staticdb
Expresswithmongo
expresswithroute



July 31, 2018

Proprietary and Confidential

• 43 •

Add the notes here.

Instructor Notes:

Add instructor notes here.

Lab**Lab 3**

July 31, 2018

Proprietary and Confidential

- 44 -

Add the notes here.