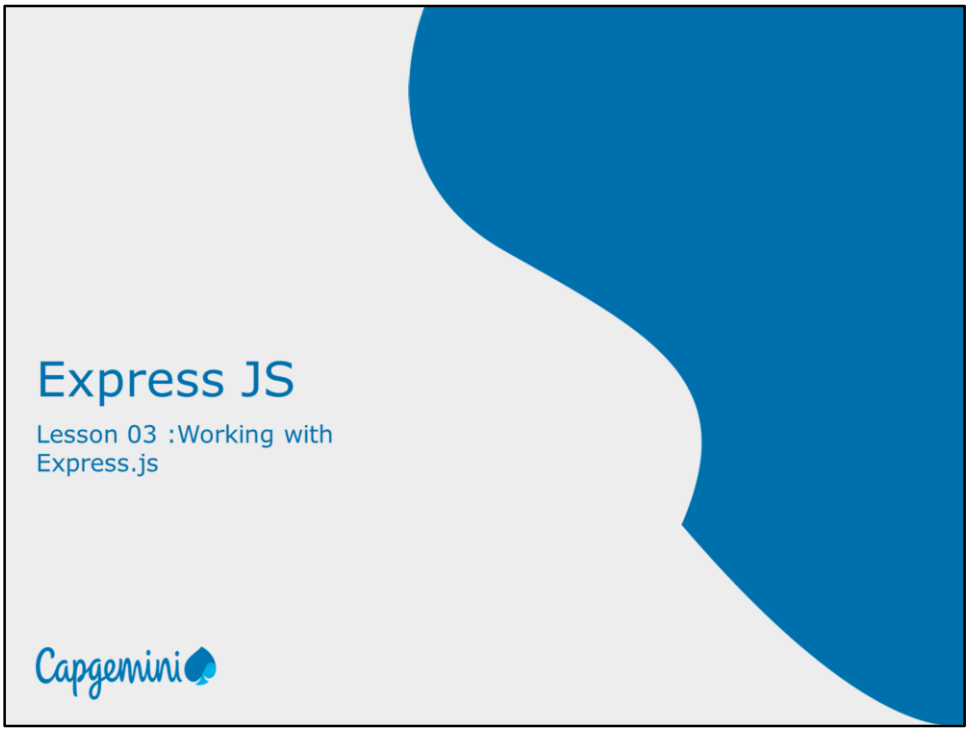
Instructor Notes:

Add instructor notes here.



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:

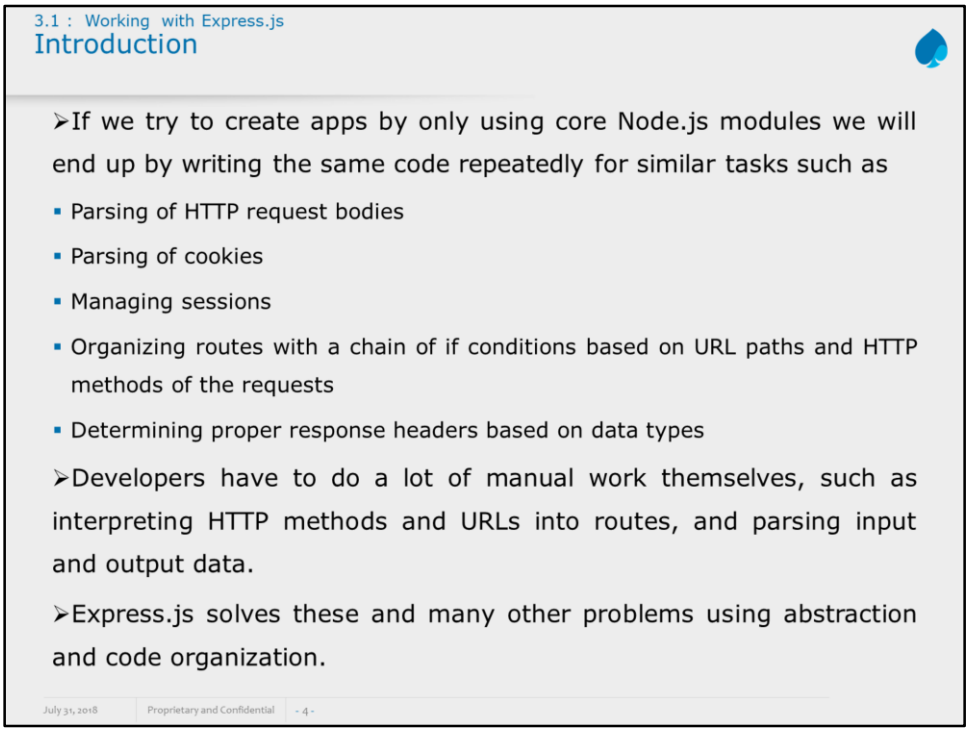
Add instructor notes here.

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

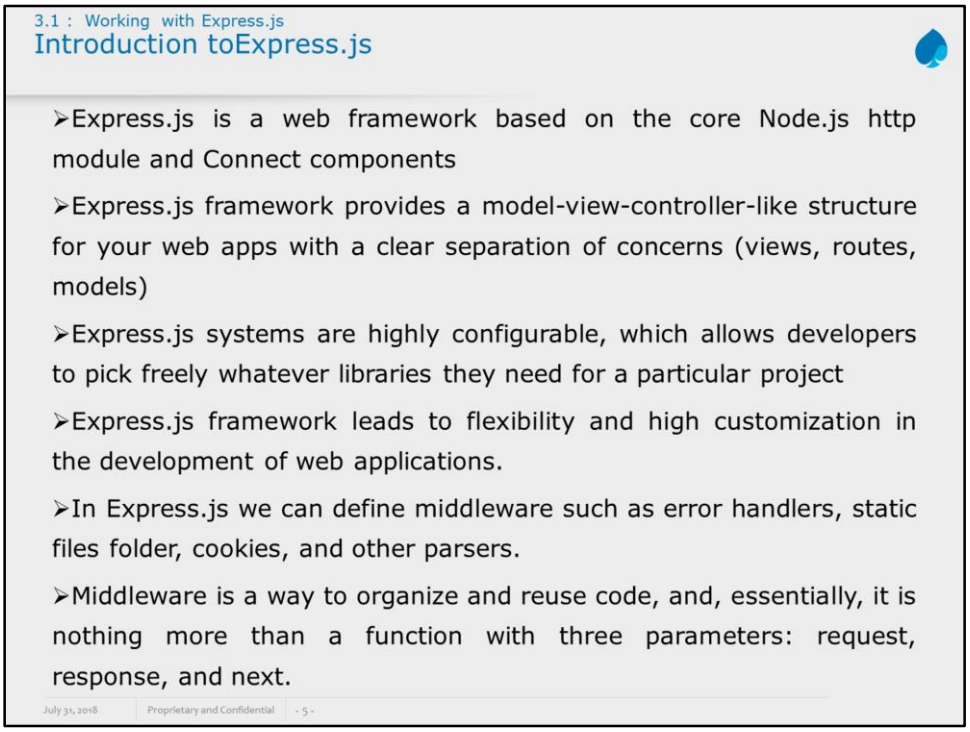
Instructor Notes:

Add instructor notes here.



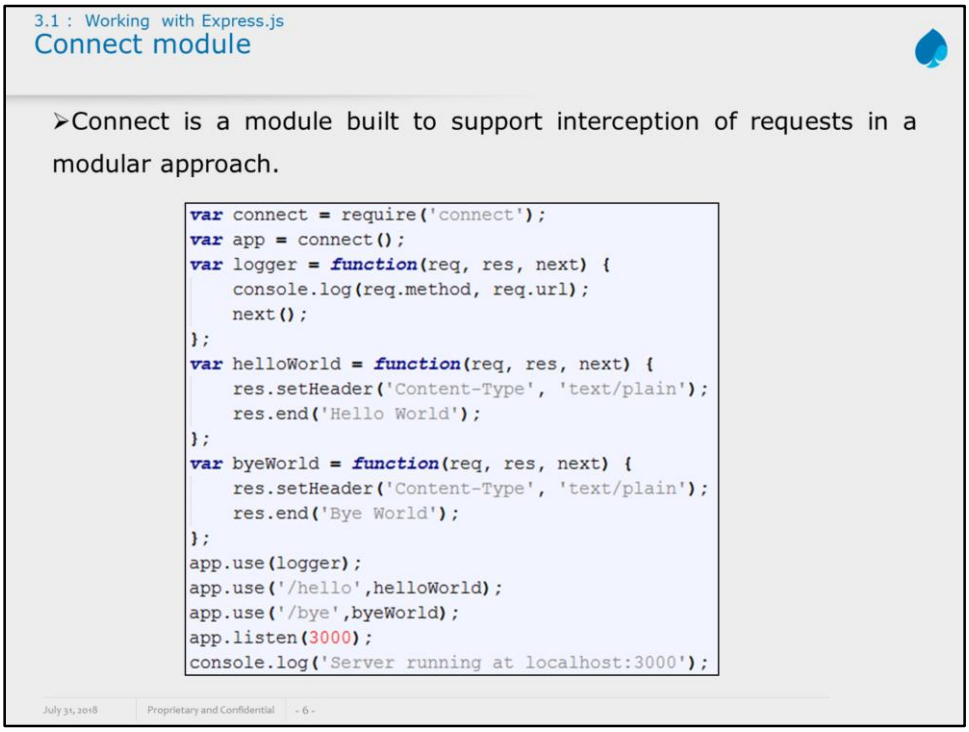
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



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:

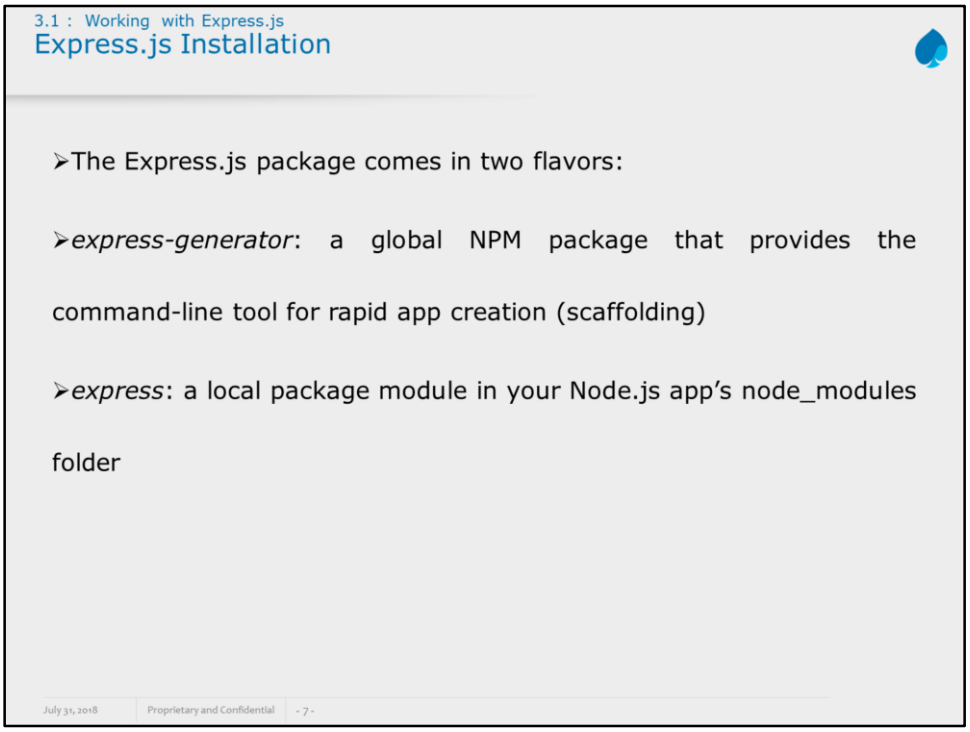
req: This is an object that holds the HTTP request information
res: This is an object that holds the HTTP response information and allows you to set the response properties
next: This is the next middleware function defined in the ordered set of Connect middleware.

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.

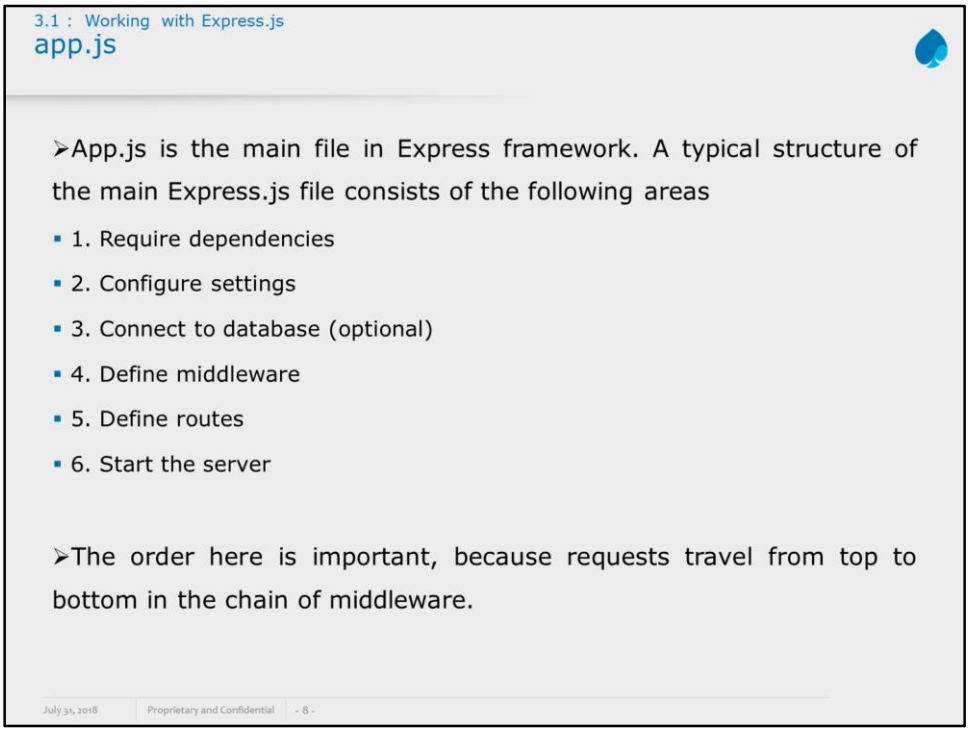
Instructor Notes:

Add instructor notes here.



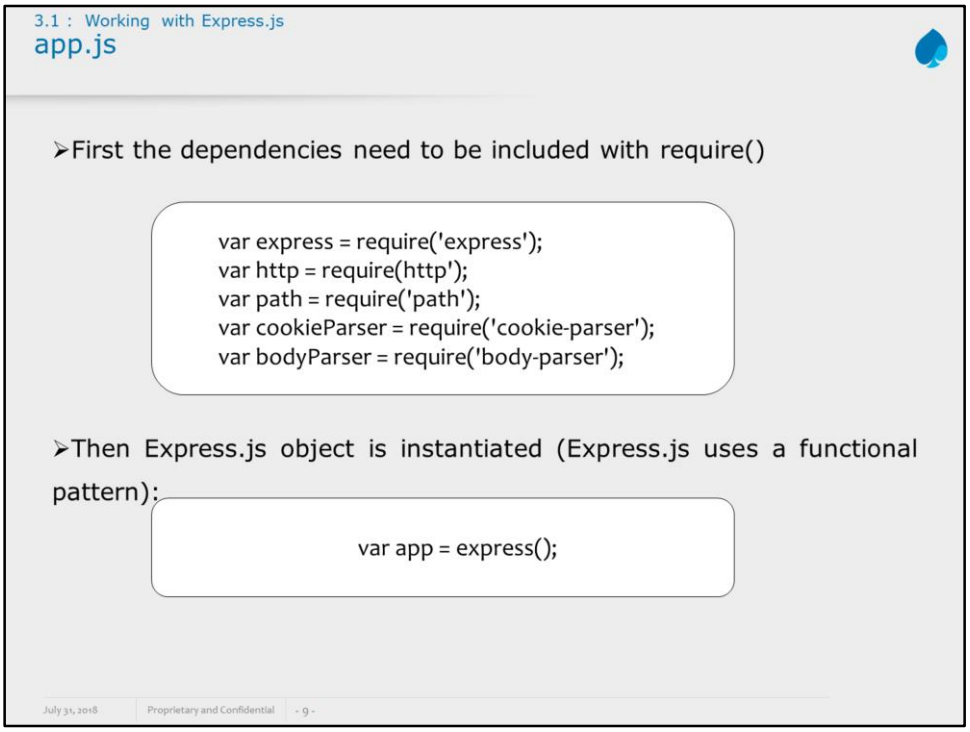
Instructor Notes:

Add instructor notes here.



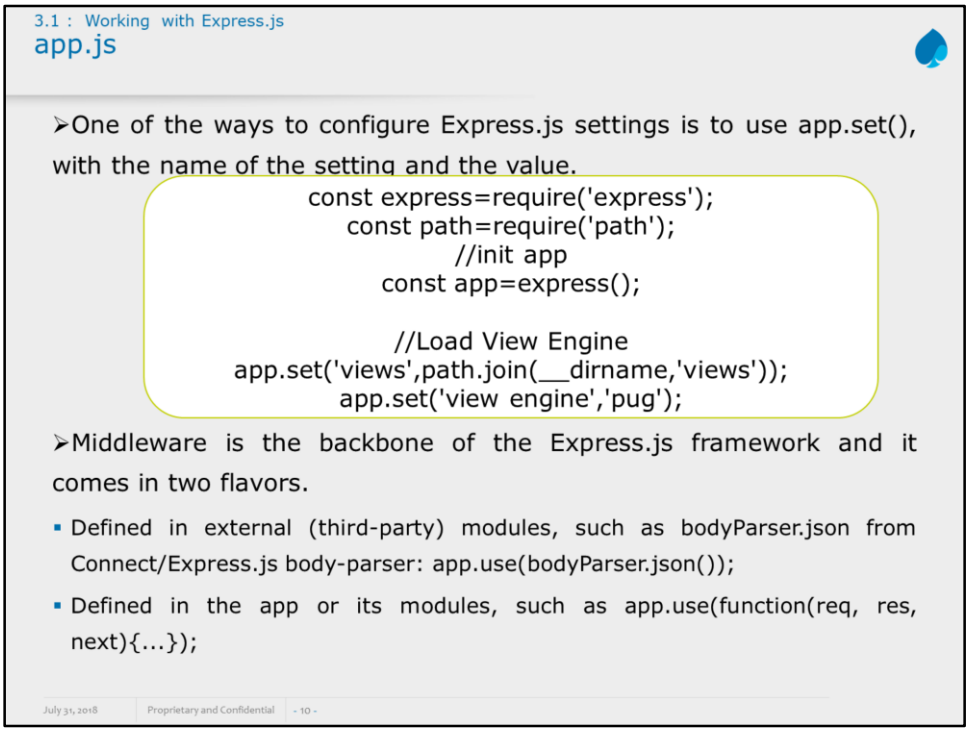
Instructor Notes:

Add instructor notes here.



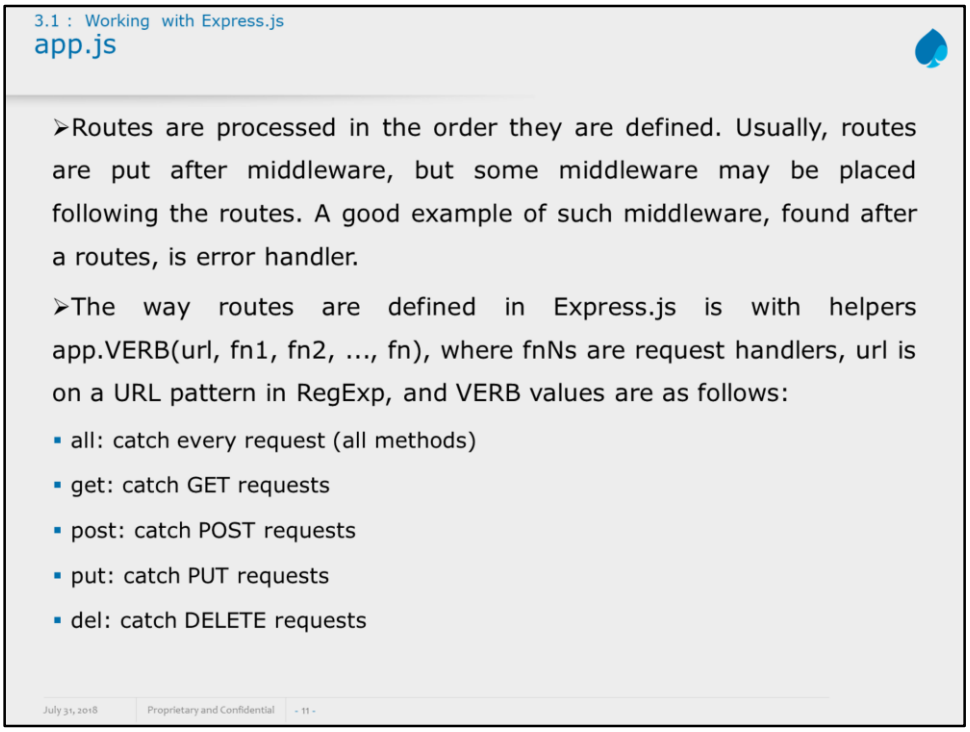
Instructor Notes:

Add instructor notes here.



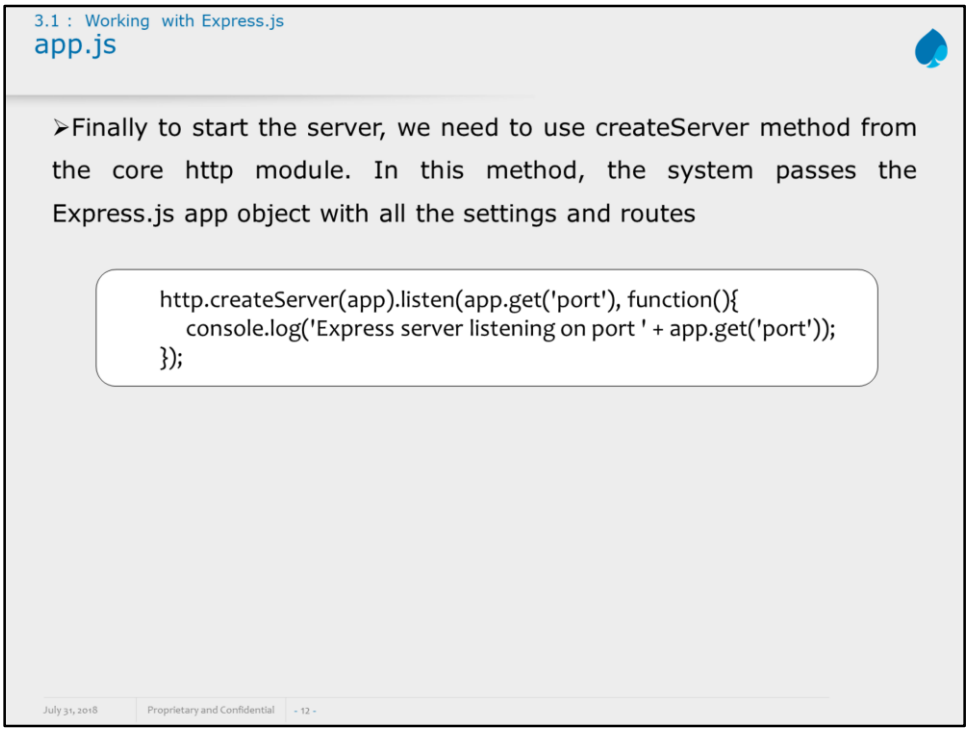
Instructor Notes:

Add instructor notes here.



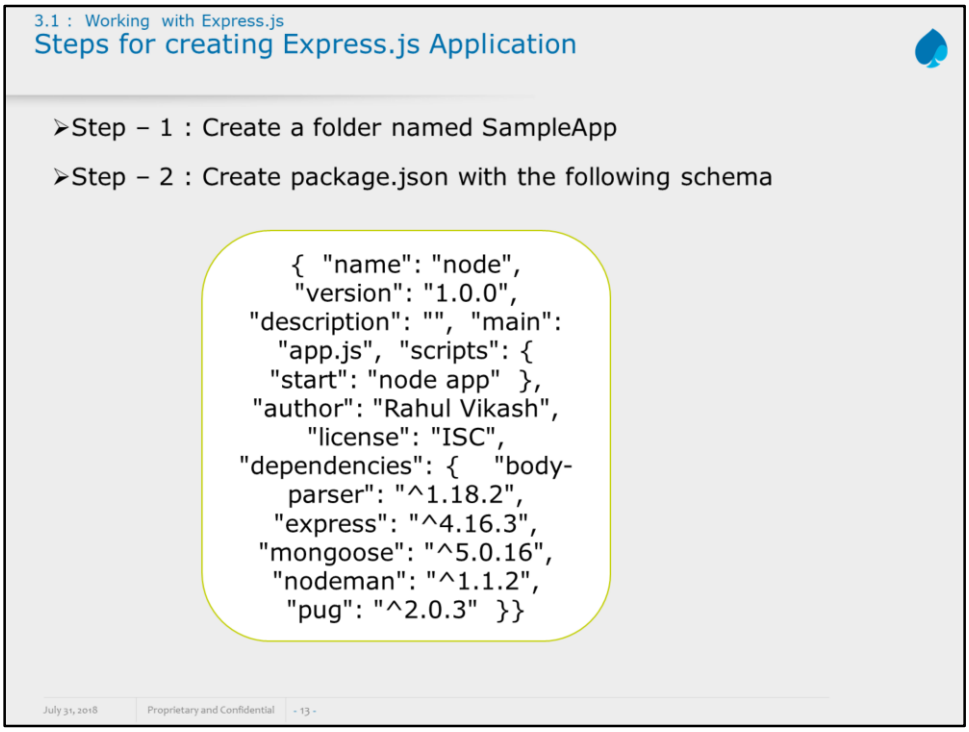
Instructor Notes:

Add instructor notes here.



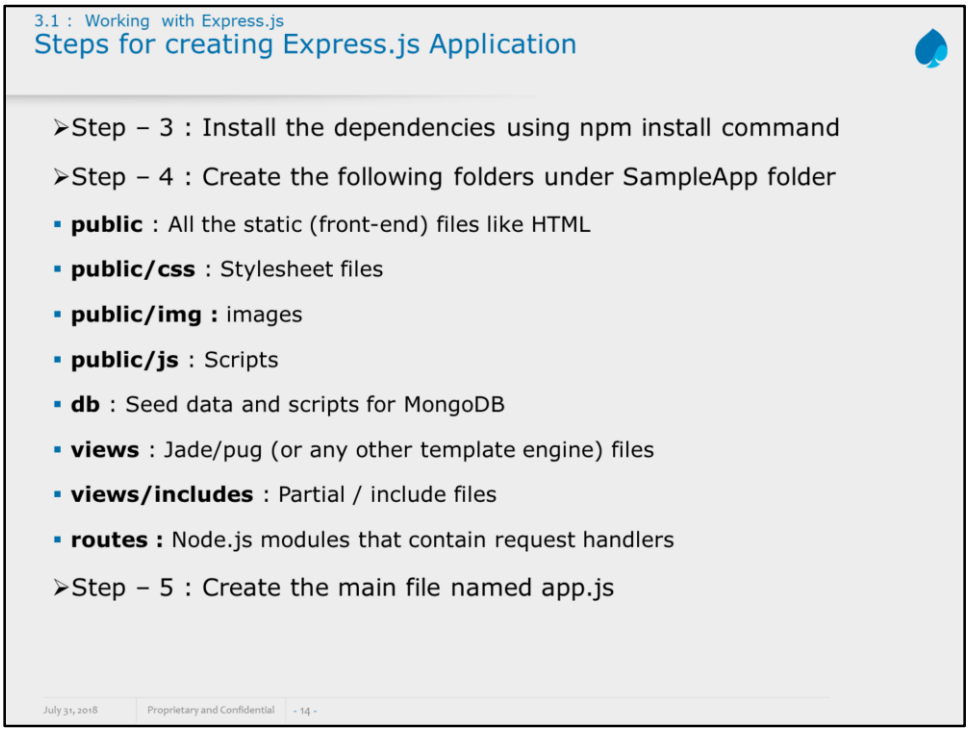
Instructor Notes:

Add instructor notes here.



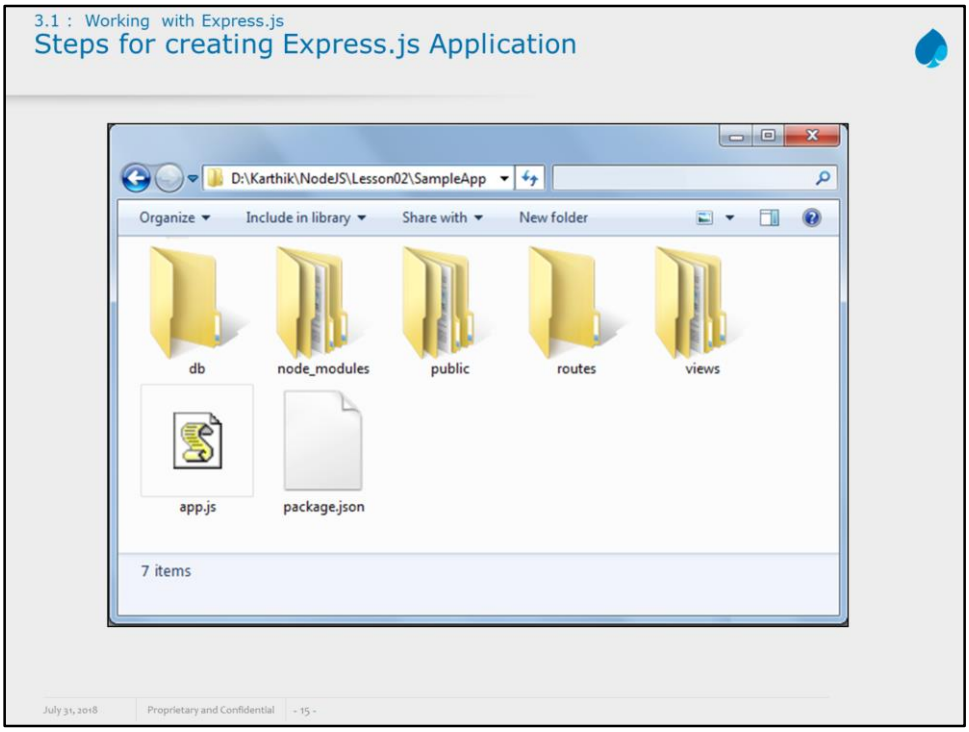
Instructor Notes:

Add instructor notes here.



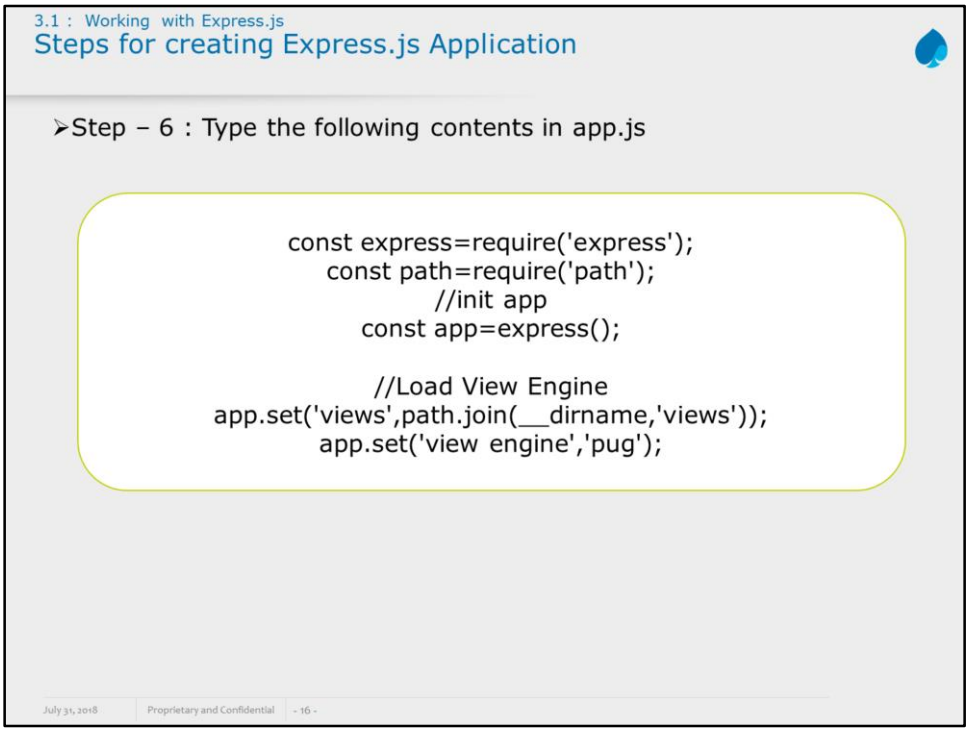
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

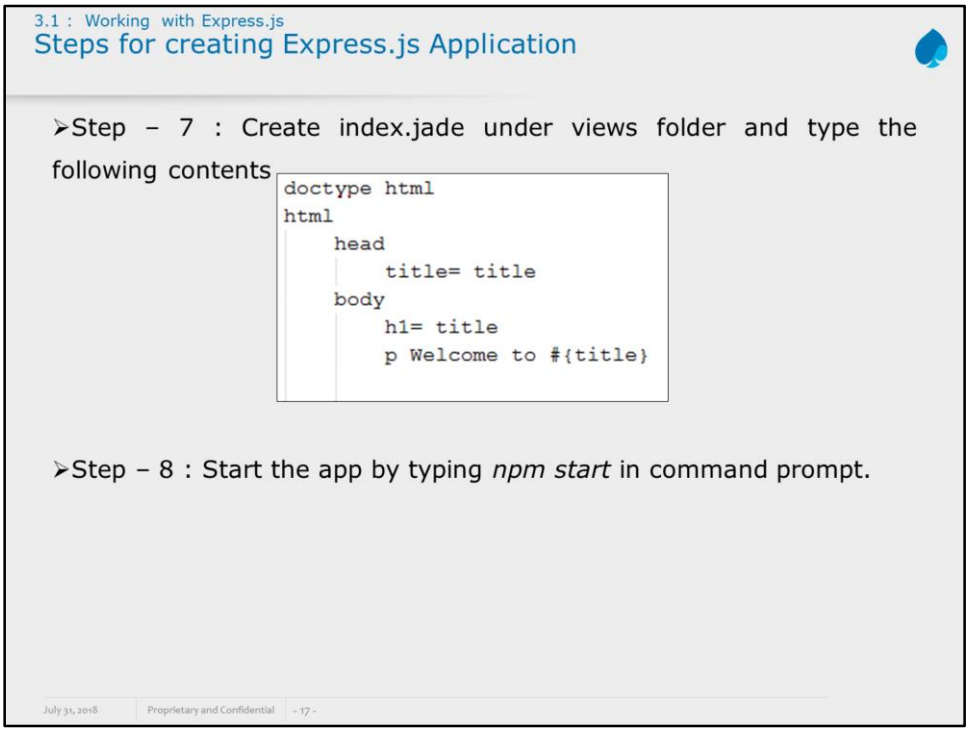


Routes in Express.js

```
// 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.



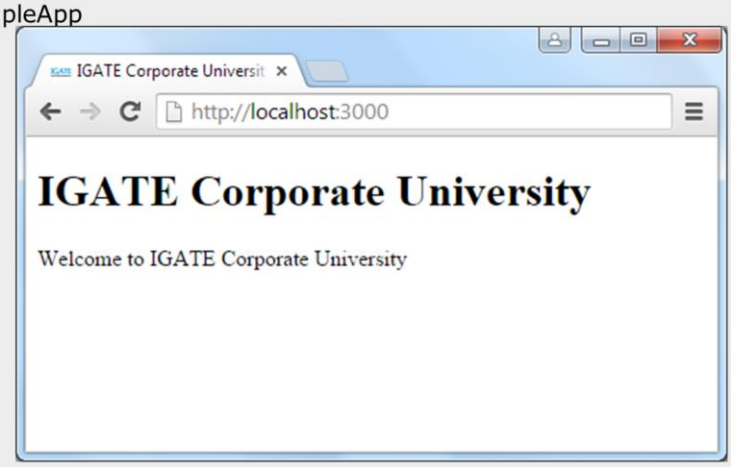
Instructor Notes:

Add instructor notes here.

3.1 : Working with Express.js

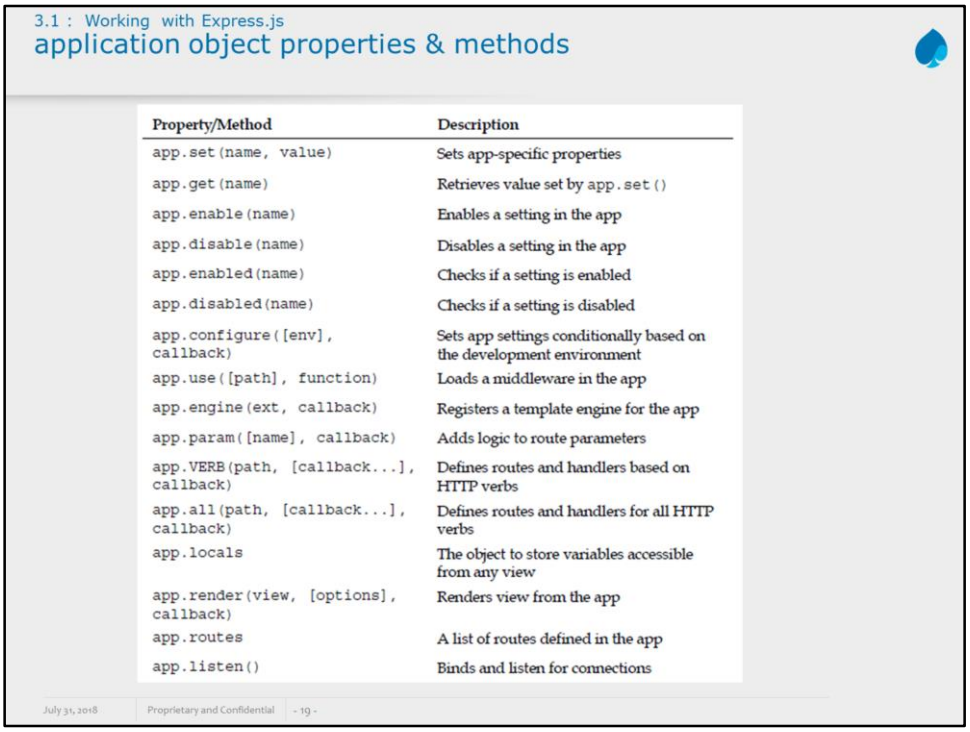
Steps for creating Express.js Application

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



Instructor Notes:

Add instructor notes here.



The application object

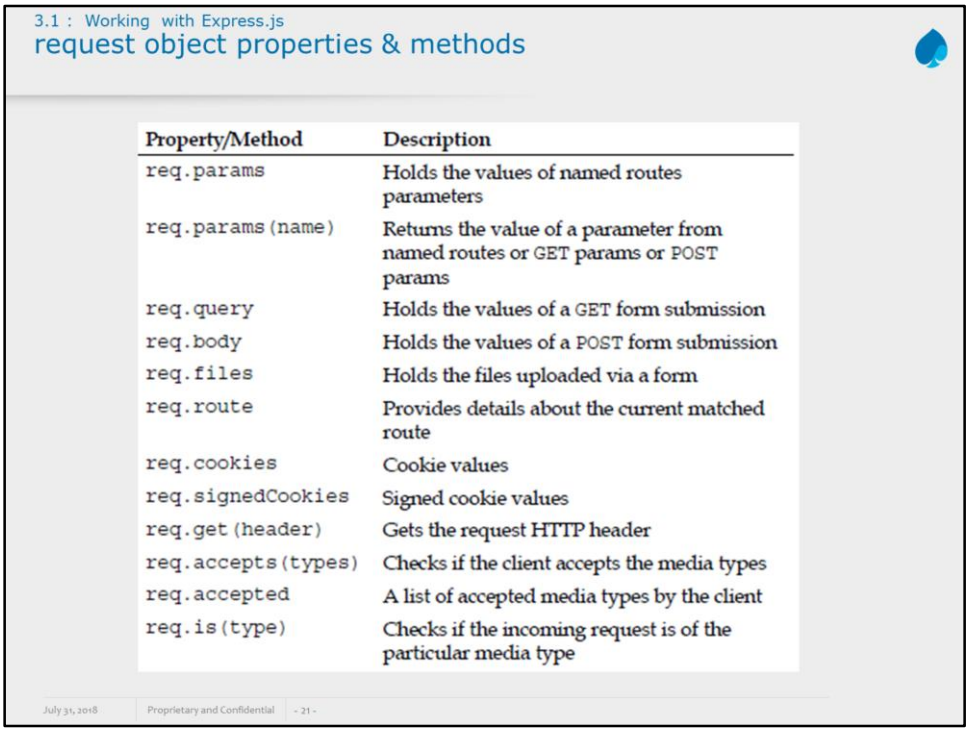
The application object is an instance of Express, conventionally represented by the variable named app. This is the main object of your Express app and the bulk of the functionality is built on it.

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.



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.

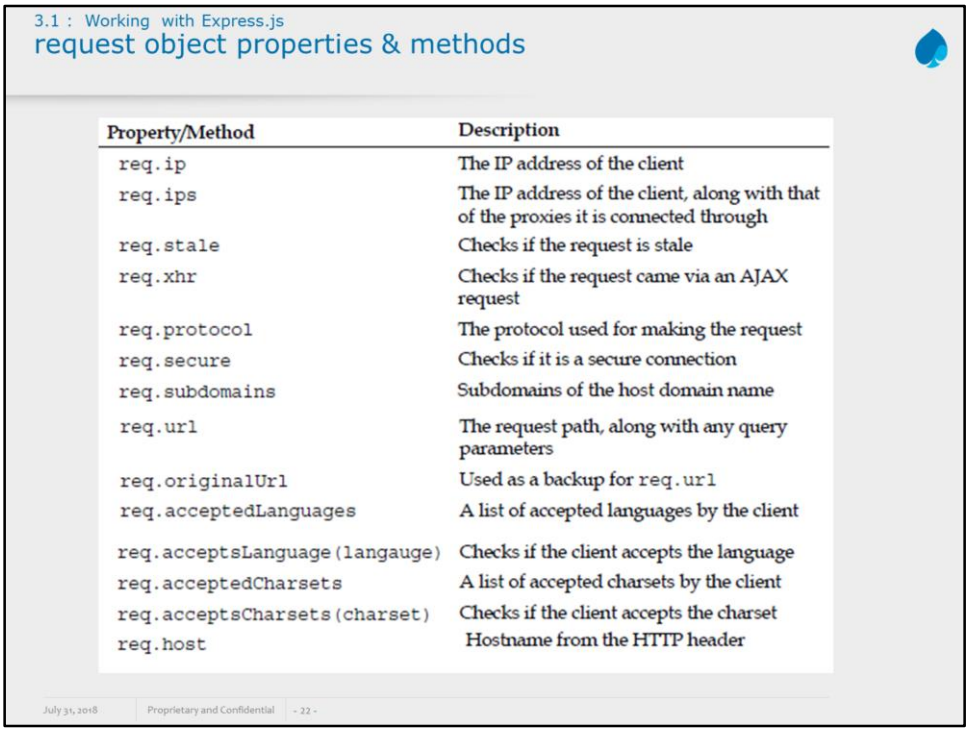
req.param(name): This is used to retrieve a value of a request parameter. Note that the parameter can be a query-string parameter, a routing parameter, or a property from a JSON request body.

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.

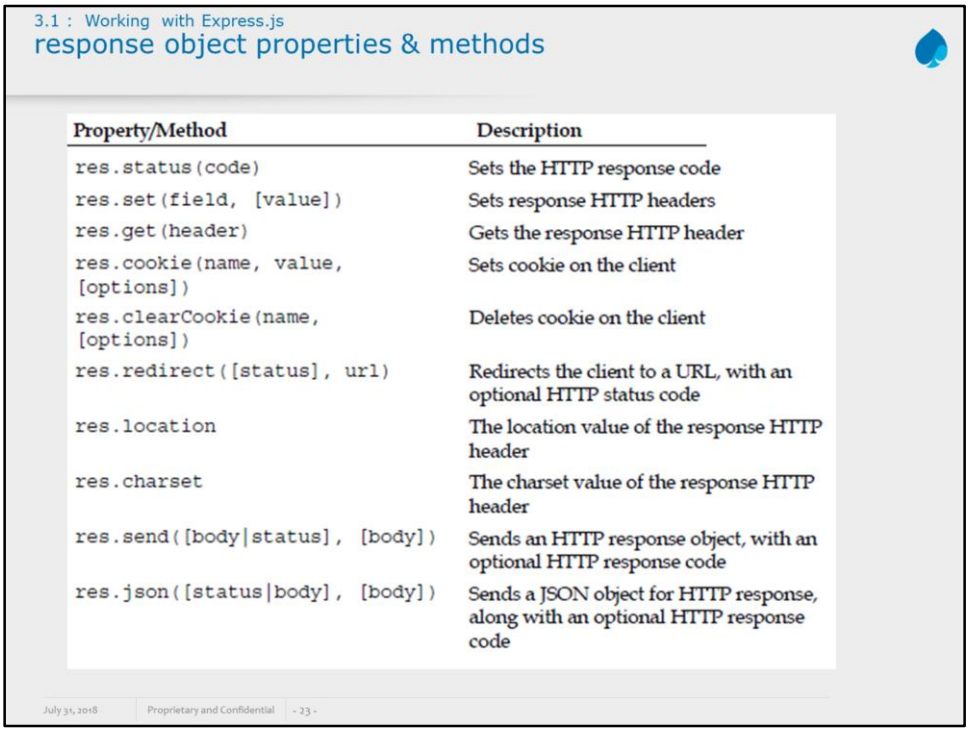
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



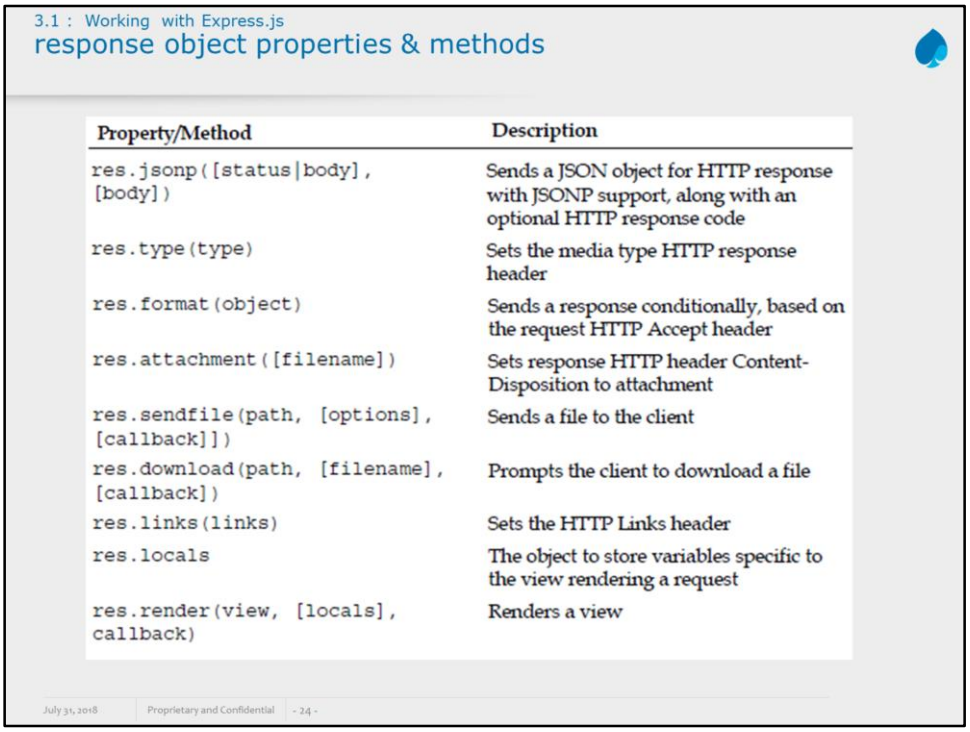
The response object

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:

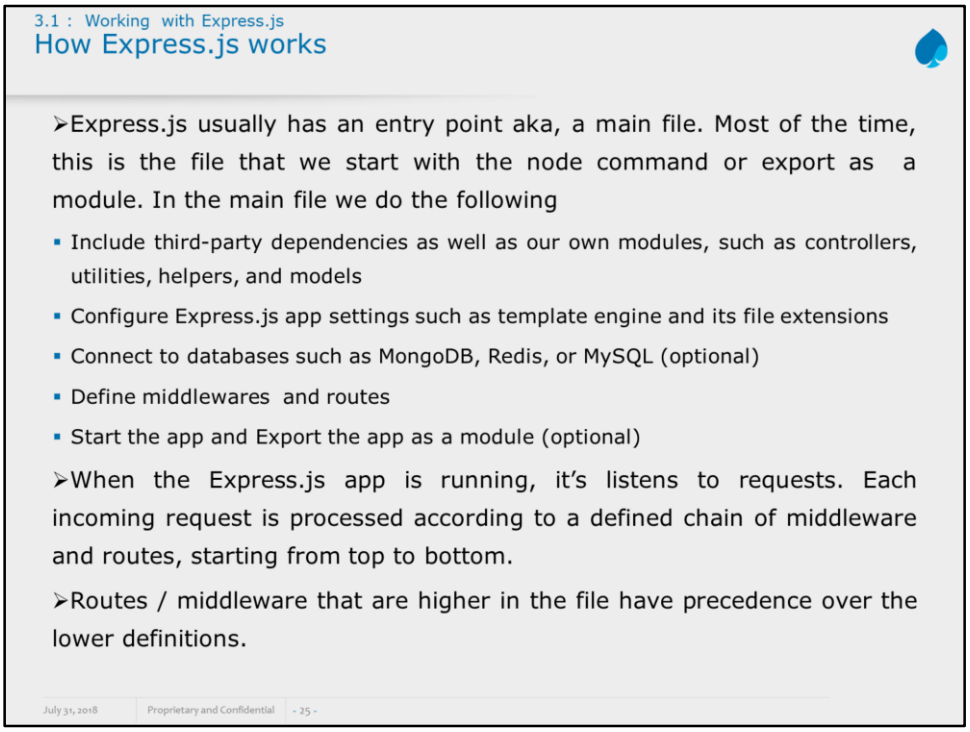
Add instructor notes here.



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

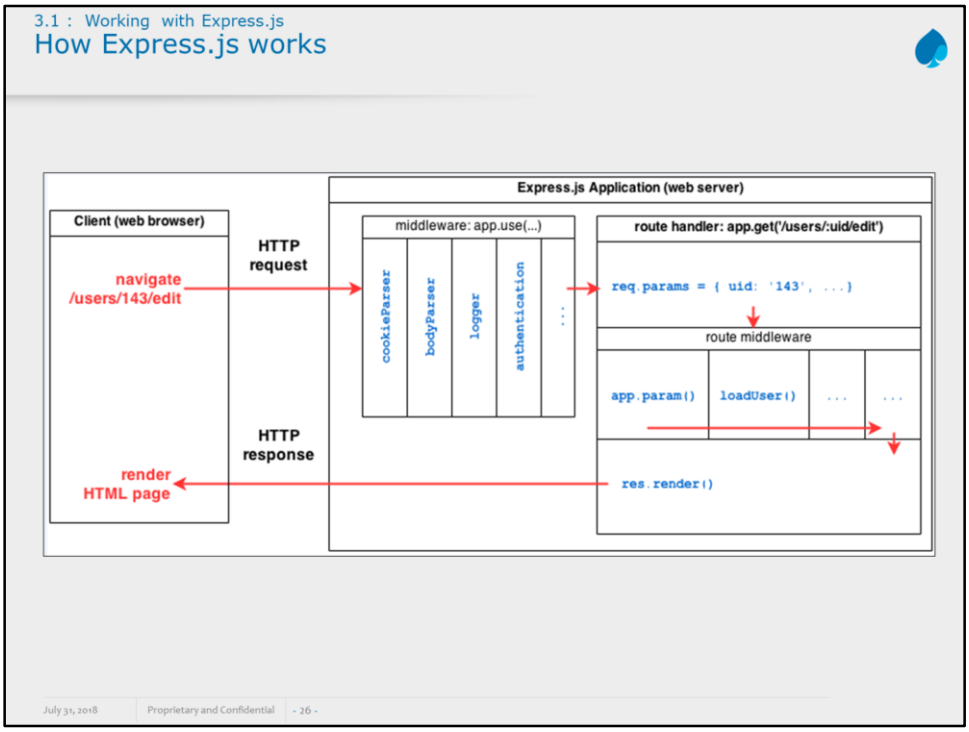
Instructor Notes:

Add instructor notes here.



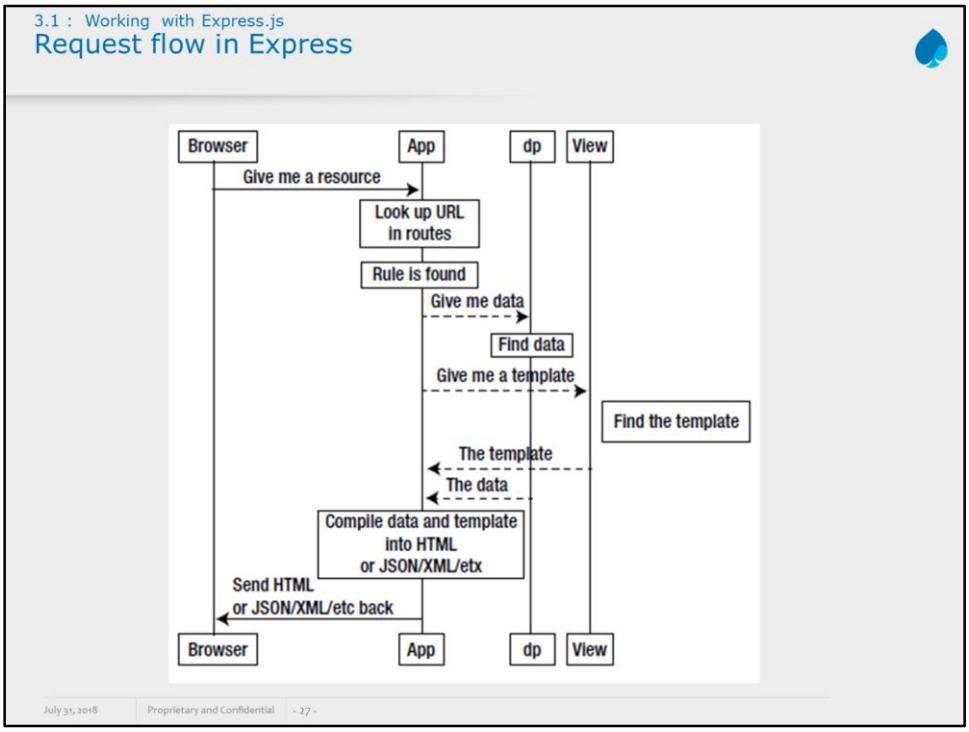
Instructor Notes:

Add instructor notes here.



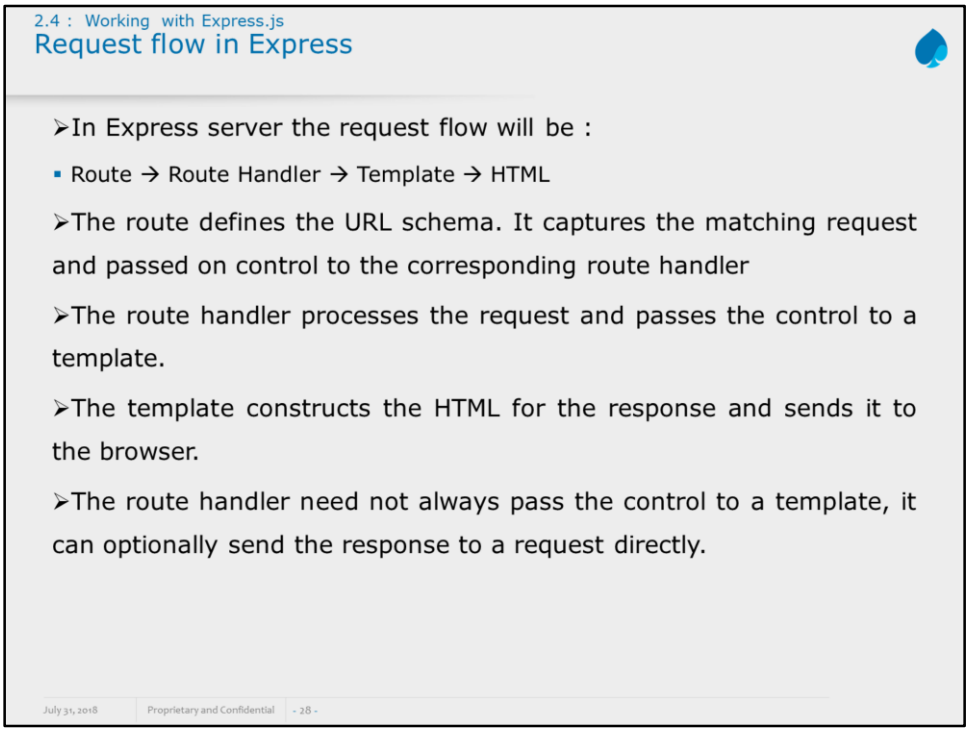
Instructor Notes:

Add instructor notes here.



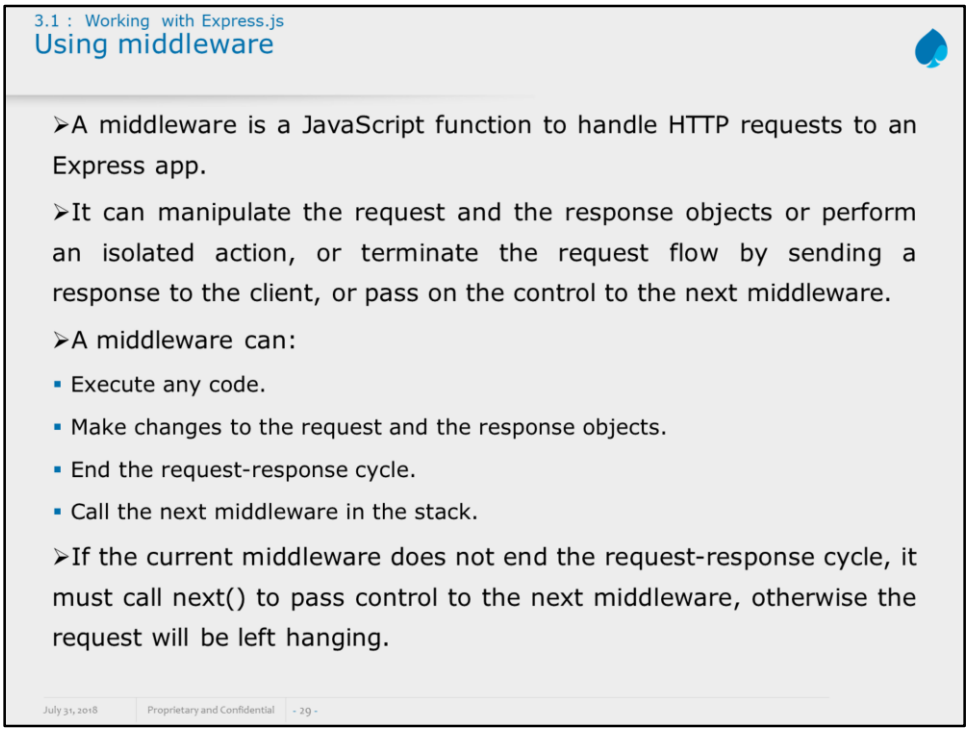
Instructor Notes:

Add instructor notes here.



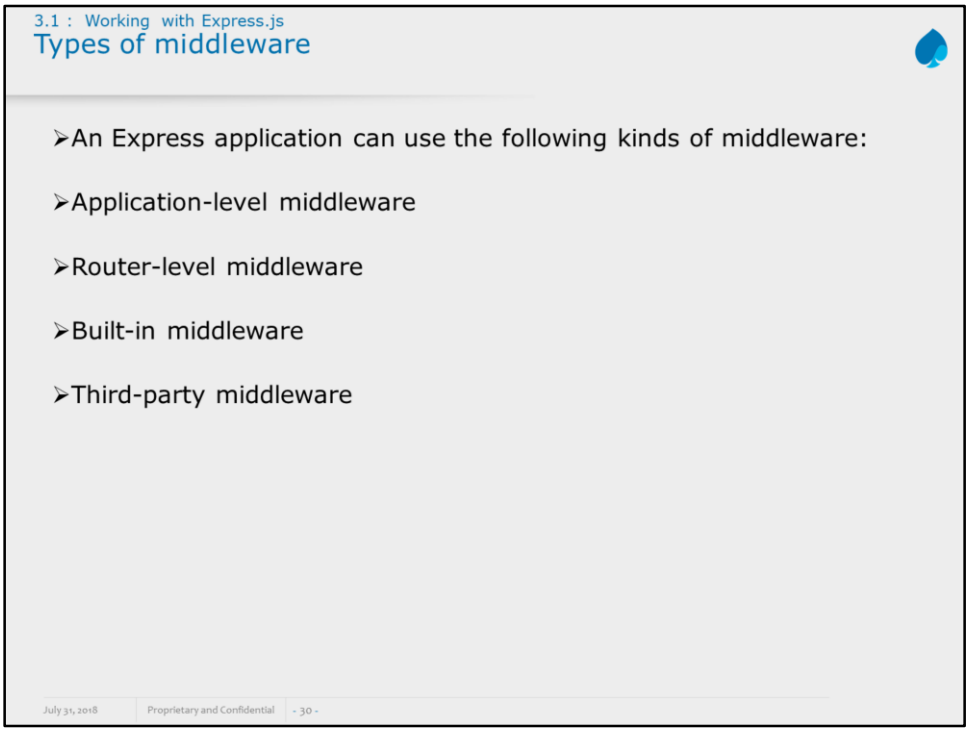
Instructor Notes:

Add instructor notes here.



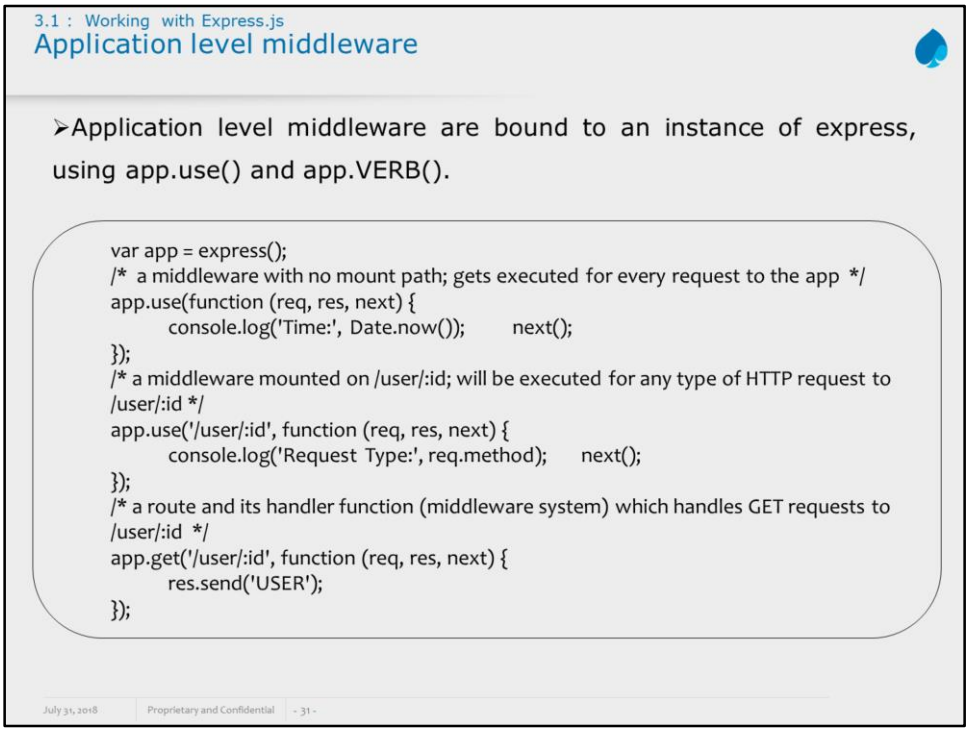
Instructor Notes:

Add instructor notes here.



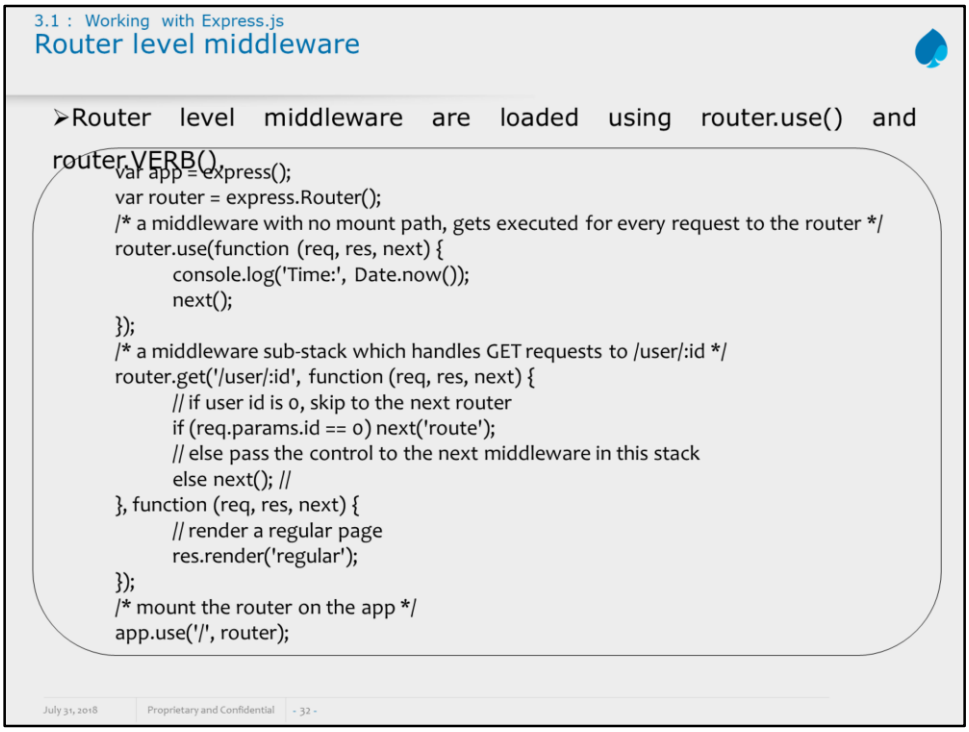
Instructor Notes:

Add instructor notes here.



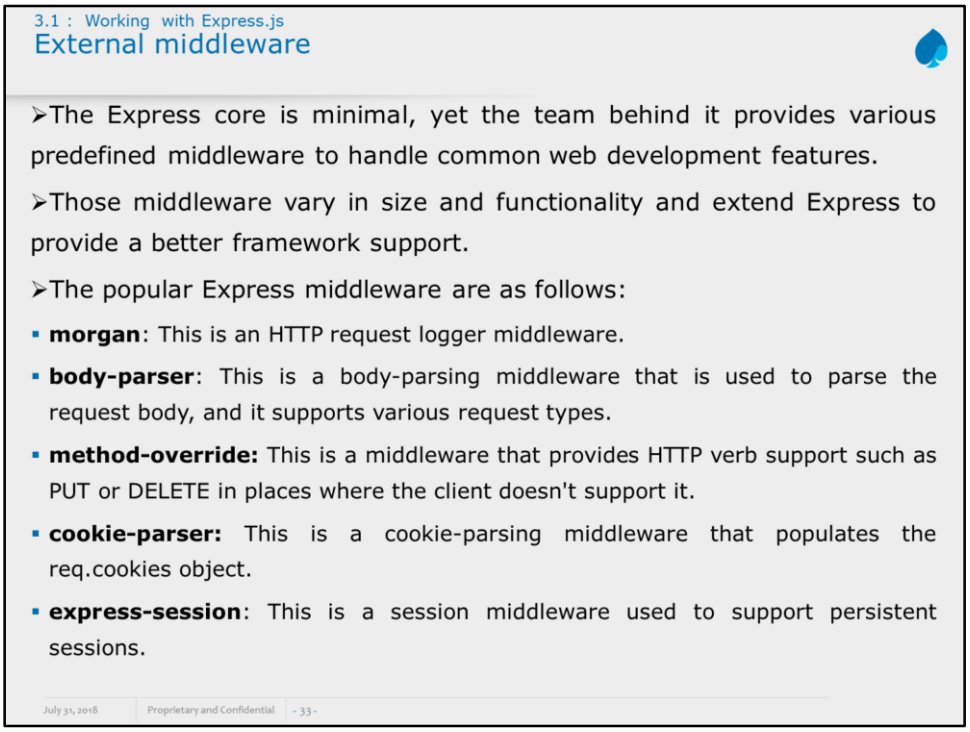
Instructor Notes:

Add instructor notes here.



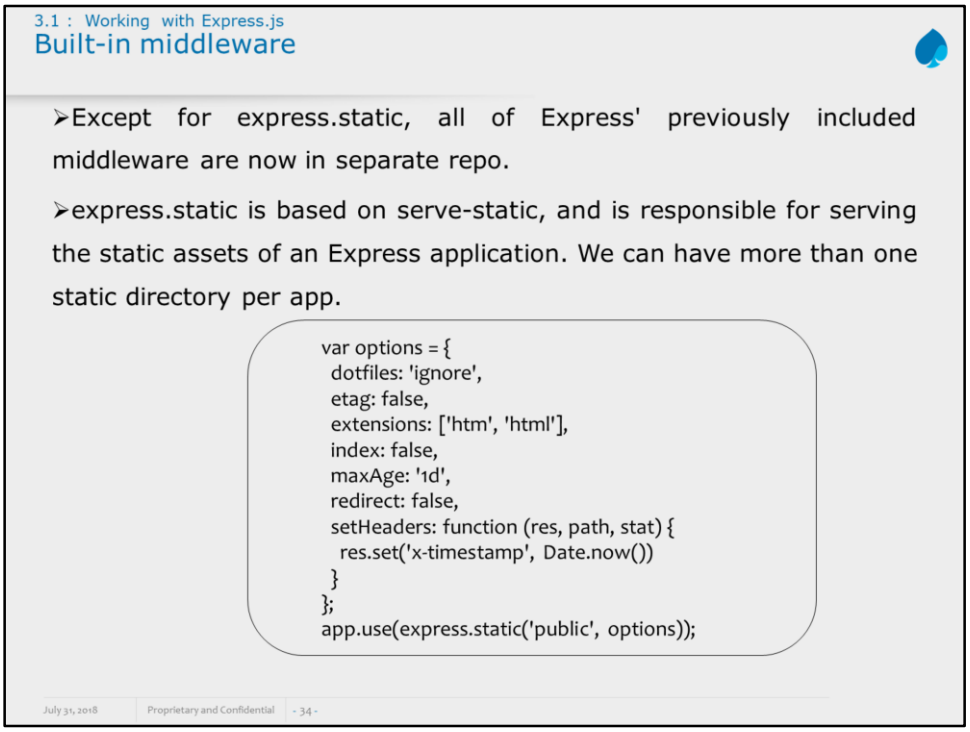
Instructor Notes:

Add instructor notes here.



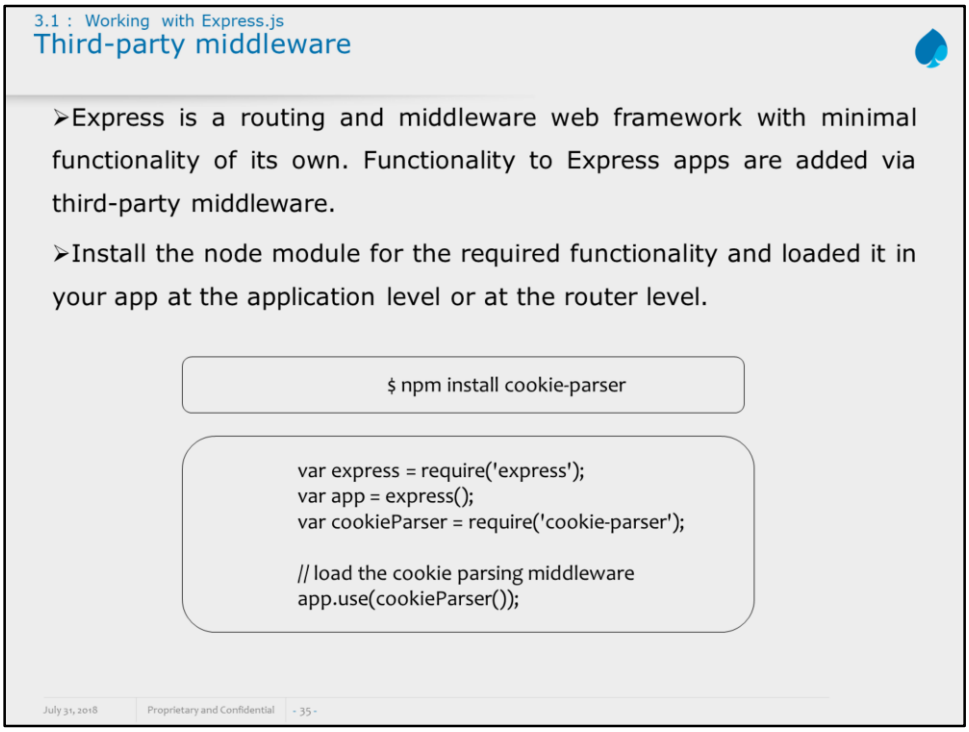
Instructor Notes:

Add instructor notes here.



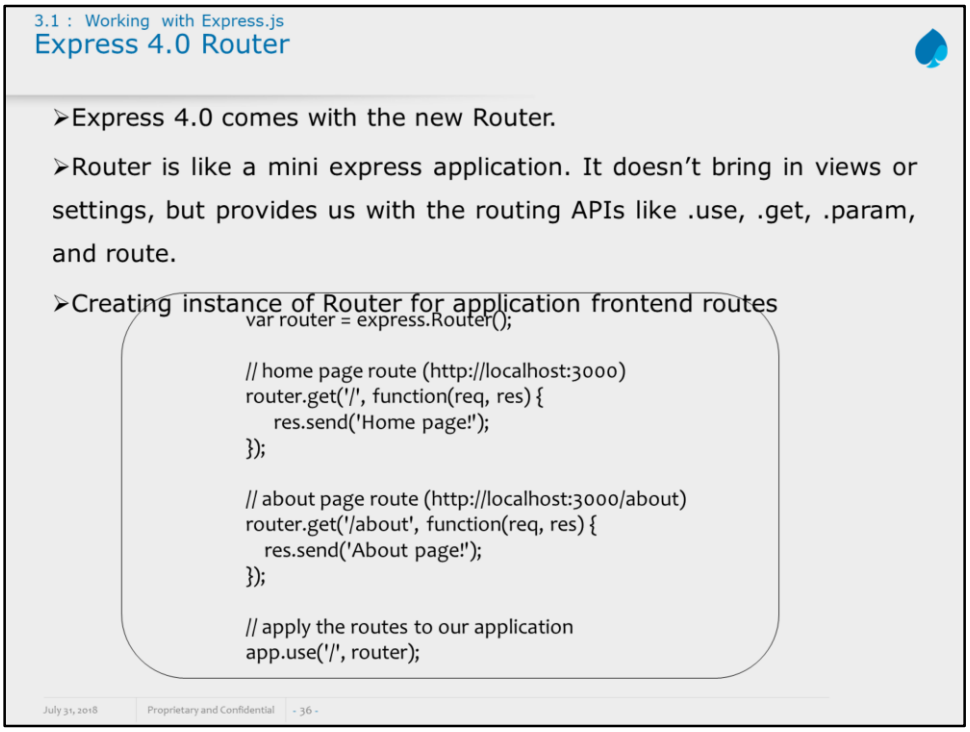
Instructor Notes:

Add instructor notes here.



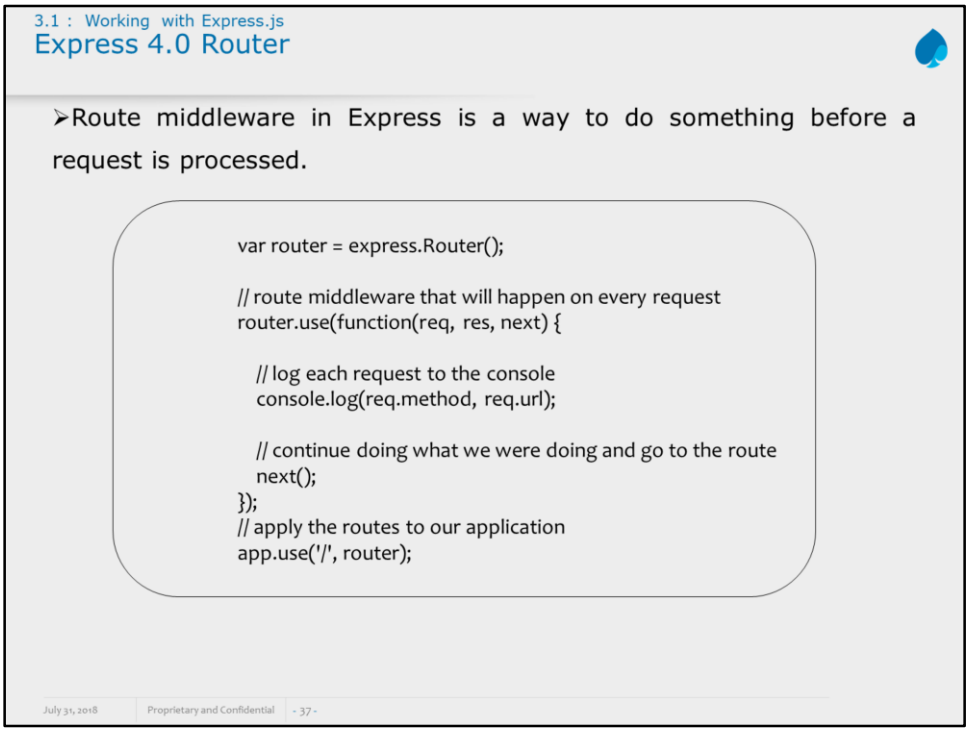
Instructor Notes:

Add instructor notes here.



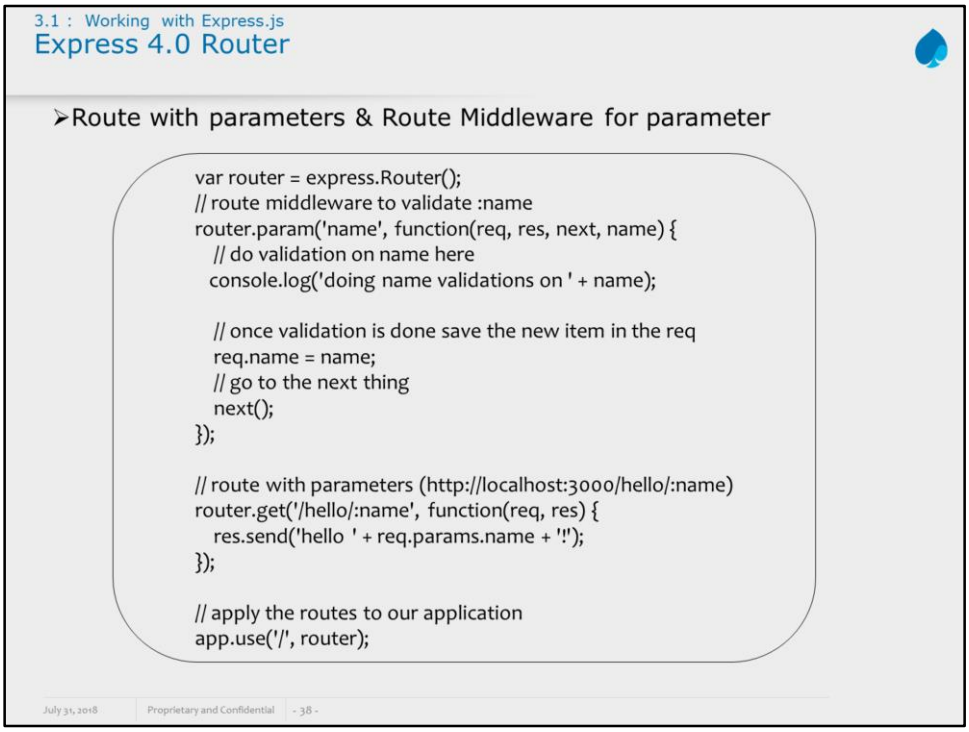
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



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

```
D:\Karthik\NodeJS\Lesson02\SampleApp>express -e -f

   create : .
   create : ./package.json
   create : ./app.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
```

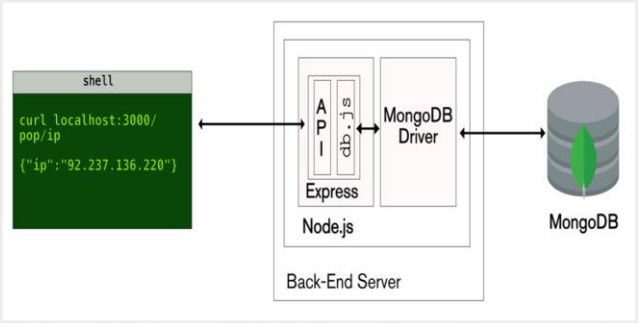
Instructor Notes:

Add instructor notes here.

3.2 : Express with Mongo db

Express With Mongo Db

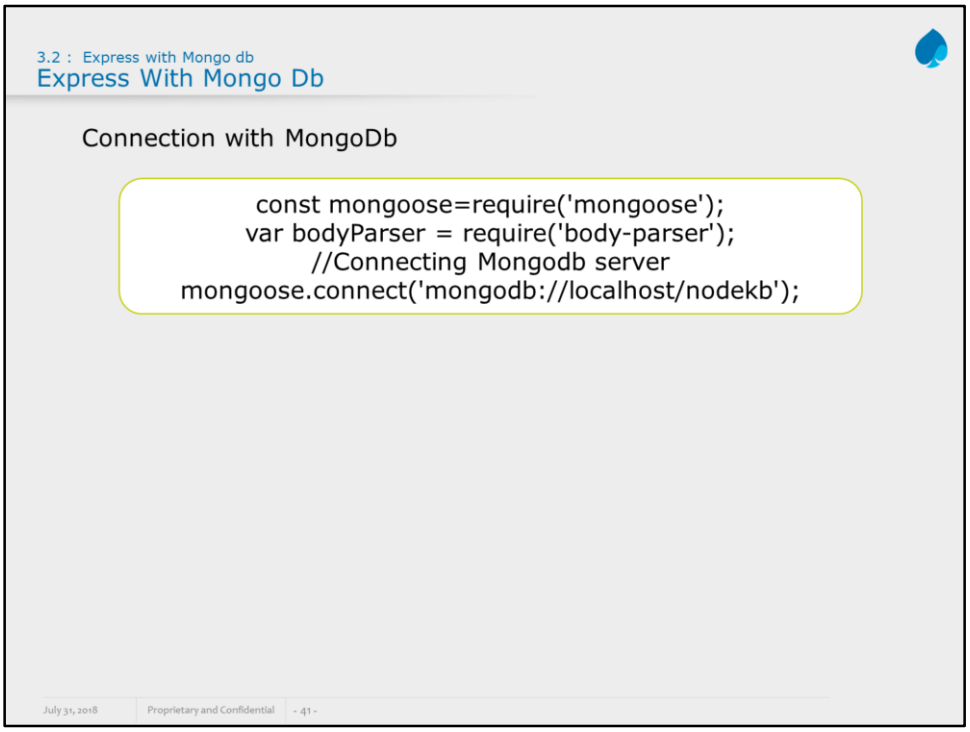
Working with Express with MongoDb



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

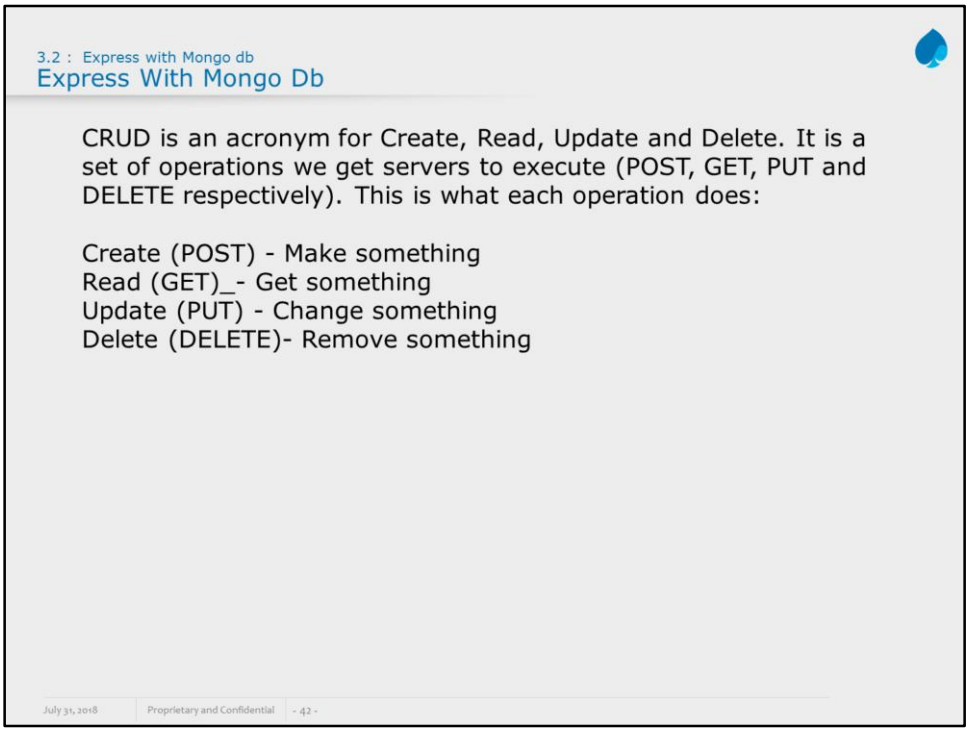
Instructor Notes:

Add instructor notes here.



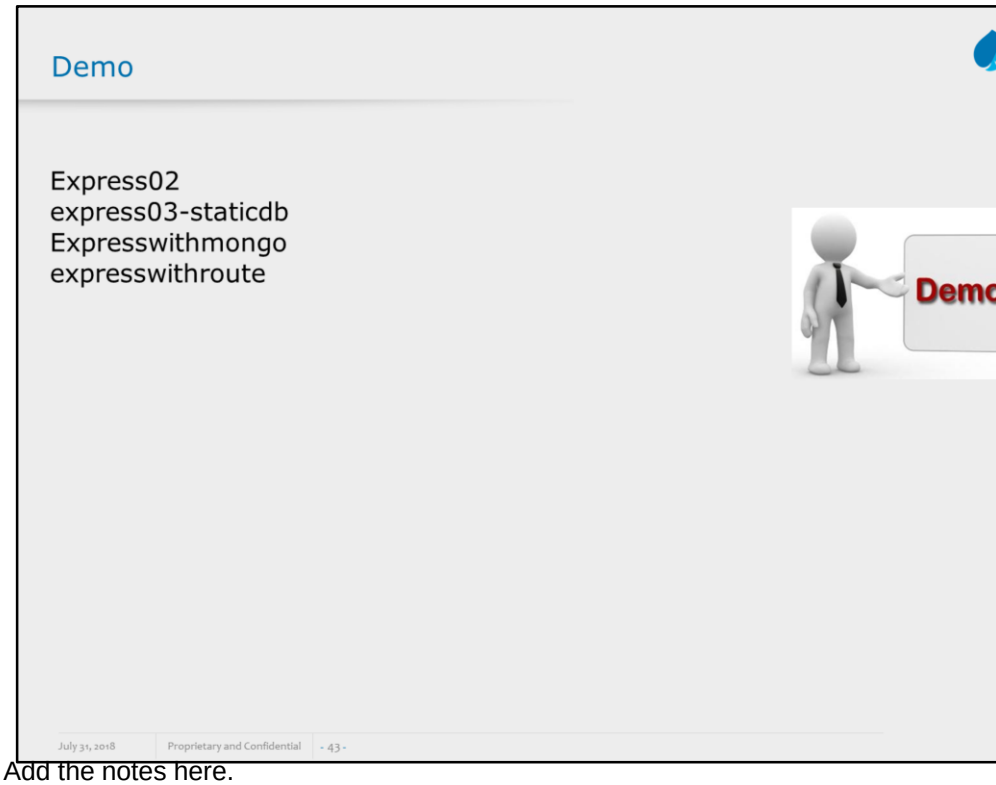
Instructor Notes:

Add instructor notes here.



Instructor Notes:

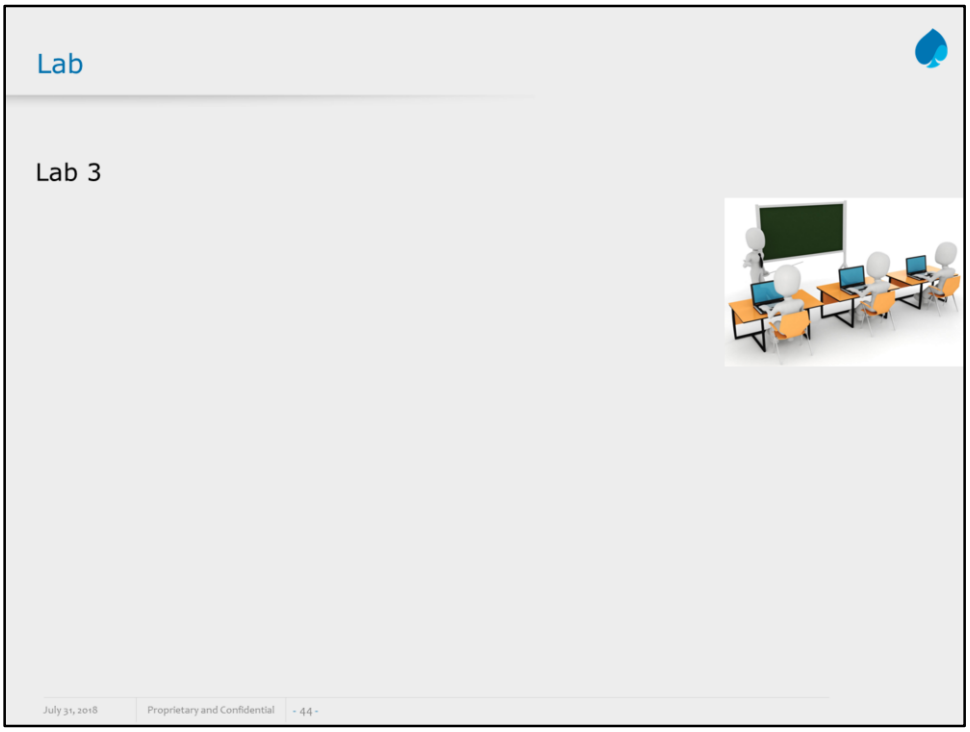
Add instructor notes here.



Add the notes here.

Instructor Notes:

Add instructor notes here.



Add the notes here.