EXPRESS.JS

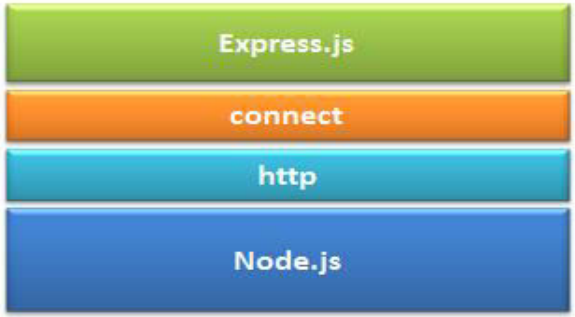
Node.js web application framework

Introduction

- If you write serious apps using only core Node.js modules you most likely find yourself reinventing the wheel by writing the same code continually for similar tasks, such as the following:
  - Parsing of HTTP request bodies and 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

Express 4.17.1

Fast, unopinionated, minimalist web framework for Node.js



- Express is a web application framework for Node
  - Its built on top of Node.js.
  - It provides various features that make web application development fast and easy compared to Node.js.

Express.js Installation

- Create a new folder: eg E:\Express-app
- E:\Express-app>npm init //for creating package.json
- E:\Express-app>npm install express --save

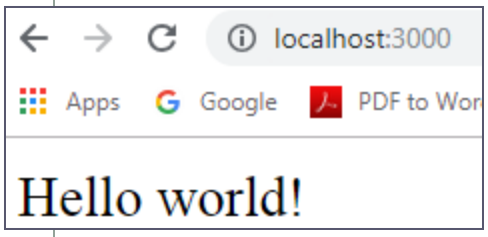
```
{
  "name": "express-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

First app

- An Express app is created by calling the express() function
  - express() : Creates an Express application; is a top-level function exported by the express module
  - The app object conventionally denotes the Express application.
  - This object, which is traditionally named app, has methods for routing HTTP requests, configuring middleware, rendering HTML views, registering a template engine, and modifying application settings that control how the application behaves

```
let express = require("express")  // import the express module

let app = express()  // create an Express application

app.get("/",function(req,resp){
    resp.send("Hello world")
})

app.listen(3000, function(){
    console.log("app running on port 3000")
})
```

The *app.listen()* method returns an http.Server object



Express Routes

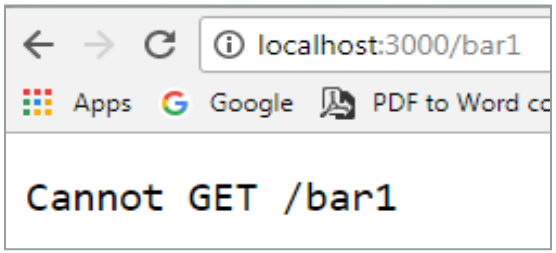
- HTTP verb and URL combinations are referred to as routes, and Express has efficient syntax for handling them.

```
var express = require("express");
var http = require("http");
var app = express();

app.get("/", function(req, res, next) {
      res.send("Hello <strong>home page</strong>");
});

app.get("/foo", function(req, res, next) {
      res.send("Hello <strong>foo</strong>");
});

app.get("/bar", function(req, res, next) {
     res.send("Hello <strong>bar</strong>");
});
//app is passed to the http.createServer() method

http.createServer(app).listen(8000);
```

Open browser and type:
http://localhost:3000
http://localhost:3000/foo
http://localhost:3000/bar
http://localhost:3000/admin - gives error

Express Routes

- If none of your routes match the request, you'll get a ***"Cannot GET <your-request-route>"*** message as response.



- This message can be replaced by a 404 not found page using this simple route

```
app.get('*', function(req, res){
    res.send('Sorry, this is an invalid URL.');
});
```

```
app.get('/course/:id', function(req, res){
    var course = //code to retrieve course
    If(!course){
        res.status(404).send("course not found");
});
```

Routing basics

```
app.get("/", function(req, res, next) {
      res.send("Hello <strong>home page</strong>");
});
```

- The get() method defines routes for handling GET requests.
- Express also defines similar methods for the other HTTP verbs (put(), post(), delete(), and so on).
  - app.method(path, handler) : This METHOD can be applied to any one of the HTTP verbs – get, post, put, delete.
  - All methods take a URL path and a sequence of middleware as arguments.
  - The path is a string or regular expression representing the URL that the route responds to. Note that the query string is not considered part of the route's URL.
- Also notice that we haven't defined a 404 route, as this is the default behavior of Express when a request does not match any defined routes.

Routing basics

```
app.get("/", function(req, res, next) {
        res.send("Hello <strong>home page</strong>");
});
```

- Express also augments the request and response objects with additional methods. Example response.send() .
  - send() is used to send a response status code and/or body back to the client.
  - If the first argument to send() is a number, then it is treated as the status code. If a status code is not provided, Express will send back a 200.
  - The response body can be specified in the first or second argument, and can be a string, Buffer, array, or object.

- send() also sets the Content-Type header unless you do so explicitly.
  - If the body is a string, Express will set the Content-Type header to text/html.
  - If the body is an array or object, then Express will send back JSON.
  - If the response body is a Buffer, the Content-Type header is also set to application/octet-stream

Routing basics

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

app.get("/buff", function(req, res, next) {
    var buff = Buffer.from("Hello World");
    res.send(buff.toString());
});
app.get("/string", function(req, res, next) {
    res.send("Hello <strong>String response</strong>");
});
app.get("/json", function(req, res, next) {
     res.send({name:'Soha',age:23});
});
app.get("/array", function(req, res, next) {
     res.send(['NodeJS','Angular','ExpressJS']);
});
app.get("/file", function(req, res, next) {
    res.sendFile(path.join(__dirname + '/summer.html'));
});
app.listen(3000);
```

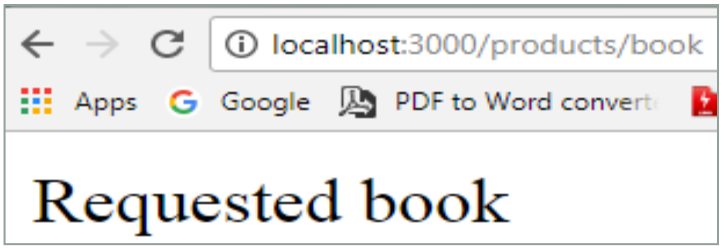
Route Parameters

- Route can be parameterized using a regular expression

```
var express = require("express");
var http = require("http");
var app = express();
app.get(/\/products\/([^\/]+)\/?$/, function(req, res, next) {
    res.send("Requested " + req.params[0]);
});
http.createServer(app).listen(8000);
```

Ignore multiple slash

/products?productId=sweater
/products/sweater



The above regular expression matches anything but /
/products/books
/products/books:aaa
/products/books/
/products/books?aaa
/products/books=aaa

Doesn't match:
/products/books/aaa
/products/books//
/products//

Eg 2 : Using regular expressions to match routes

- Assume you want to match things like /users/123 or /users/456 but not /users/anita. You can code this into a regular expression and also grab the number

```
app.get(/^\/users\/(\d+)$/, function(req, res) {
    var userId = parseInt(req.params[0], 10);
    res.send("Requested " + userId);
});
```

- **req.params -** An object containing parameter values parsed from the URL path.
  - For example, if you have the route /user/:name, then the “name” property is available as req.params.name. This object defaults to {}.
  - GET /user/shrilata
  - req.params.name // => “shrilata“

  - When you use a regular expression for the route definition, each capture group match from the regex is available as req.params[0], req.params[1]
  - GET /file/javascripts/jquery.js
  - req.params[0] // => "javascripts/jquery.js"

Route Parameters

- One of the most powerful features of routing is the ability to use placeholders to extract named values from the requested route, marked by the colon (:) character.
  - When the route is parsed, express puts the matched placeholders on the req.params object for you.

```
app.get('/user/:id', function(req, res) {
      res.send('user ' + req.params.id);
});

app.get('/product/:prodname', function(req, res) {
      res.send('Product :  ' + req.params.prodname);
});
```

Placeholders match any sequence of characters except for forward slashes.

Route Parameters

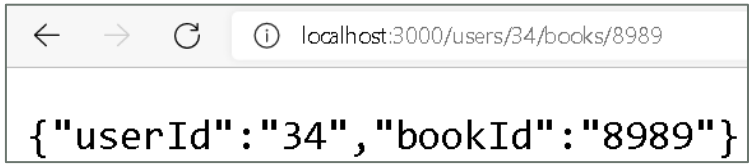
- Below example, creates a named parameter productId.

```
app.get("/product/:productId(\\d+)", function(req, res, next) {
    res.send("Requested " + req.params.productId );
});
```

productId can now only be made up of digits

Invoke as : http://localhost:3000/product/123

```
app.get('/users/:userId/books/:bookId', function (req, res) {
    res.send(req.params)
    //res.send(req.params.userId + ":" + req.params.bookId)
})
  //Route path: /users/:userId/books/:bookId
  //Request URL: http://localhost:3000/users/34/books/8989
  //req.params: { "userId": "34", "bookId": "8989" }
```

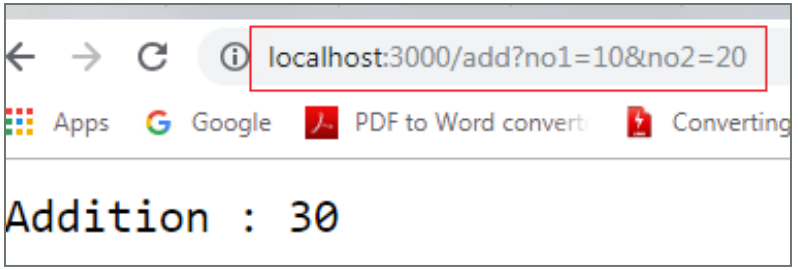


Working with parameters using get

- **req.query**
  - Express parses query string parameters by default, and puts them into the req.query property.
  - If the request is GET /search?username=Shrilata
    req.query.username returns “Shrilata"

  - Lets say the incoming url is : http://localhost:3000/add?no1=10&no2=20
  - Use req.query to query the request parameters

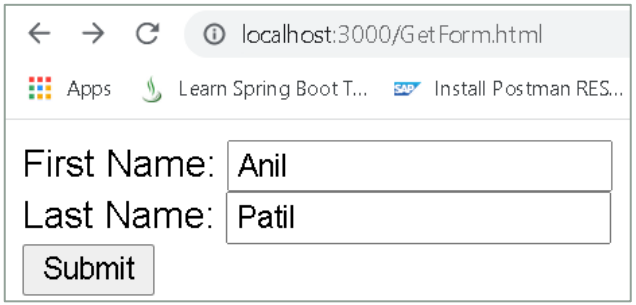
```
app.get("/add", function (req, res) {
    n1 = parseInt(req.query.no1);
    n2 = parseInt(req.query.no2);
    res.end("Addition : " +   (n1 + n2) );
});
```

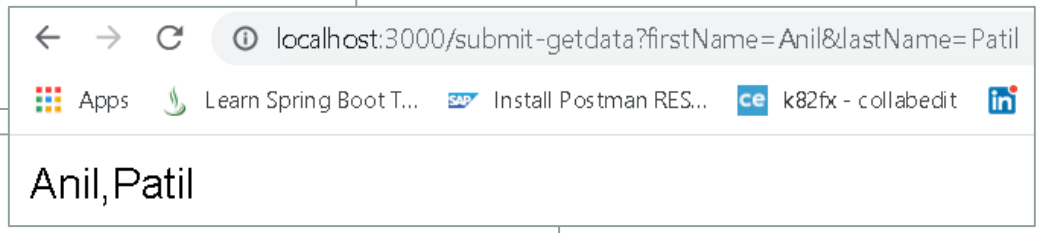


Handle GET Request

GetForm.html > ...

```
<!DOCTYPE html>
<html>
<body>
    <form action="/submit-getdata" method="get">
        First Name: <input name="firstName" type="text" /
        Last Name: <input name="lastName" type="text" />
        <input type="submit" />
    </form>
</body>
</html>
```






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

app.get('/GetForm.html', function (req, res) {
    res.sendFile('public/GetForm.html' , { root : __dirname});
});

app.get('/submit-getdata', function (req, res) {
    console.log(req.query.firstName);
    console.log(req.query.lastName);
    res.send(req.query.firstName+","+req.query.lastName);
});
app.listen(3000);
```

Multiple different methods

- We can also have multiple different methods at the same route.

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

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

app.post('/hello', function(req, res){
   res.send("You just called the post method at '/hello'!\n");
});

app.delete('/hello', function(req, res){
   res.send("You just called the delete method at '/hello'!\n");
});

app.listen(3000);
```

Handle POST Request

- To handle HTTP POST request in Express.js version 4 and above, you need to install a middleware module called body-parser.
  - This is used to parse the body of requests which have payloads attached to them.
  - Install it using : **npm install** --save body-parser
  - Mount it by including the following lines in your js

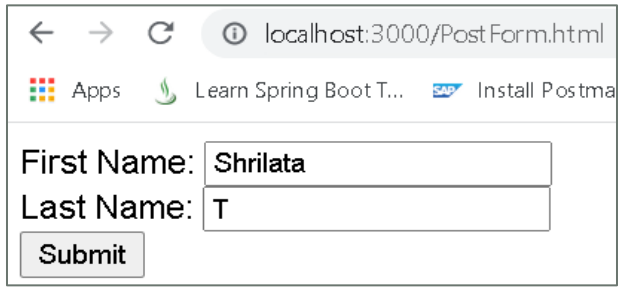
```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
```

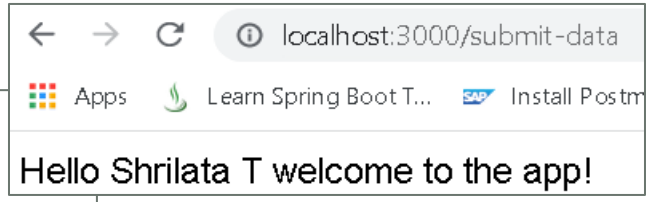
  - This body-parser module parses the JSON, buffer, string and url encoded data submitted using HTTP POST request.
  - Eg :To parse json data: app.use(bodyParser.json())

bodyParser.urlencoded(): Parses the text as URL encoded data (which is how browsers tend to send form data from regular forms set to POST) and exposes the resulting object (containing the keys and values) on **req.body**

Handle POST Request

```html
<!DOCTYPE html>
<html>  <!-- PostForm.html -->
<body>
    <form action="/submit-data" method="post">
        First Name: <input name="firstName" type="text" />
        Last Name: <input name="lastName" type="text" />
        <input type="submit" />
    </form>
</body>
</html>
```






```node
var express = require('express');
var app = express();
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));

app.get('/PostForm.html', function (req, res) {
    //res.sendFile('E:\\Node.js\\Demo\\expressPrj\\PostForm.html');
    res.sendFile('public/PostForm.html' , { root : __dirname});
});

app.post('/submit-data', function (req, res) {
    var name = req.body.firstName + ' ' + req.body.lastName;
    res.send("Hello " + name + ' welcome to the app!');
});
app.listen(3000);
```

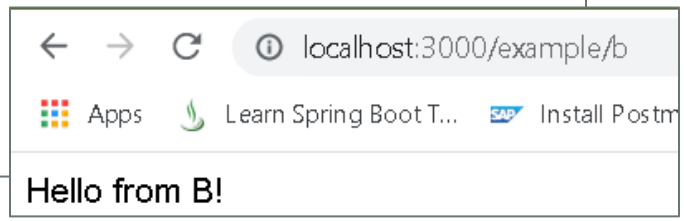
Routing handlers

- app.get(path, callback [, callback ...])
  - Routes HTTP GET requests to the specified path with the specified callback functions. Callback functions can be:
    - A middleware function.
    - A series of middleware functions (separated by commas).
    - An array of middleware functions.
  - The next() function gives you the opportunity to do some additional examinations on the incoming URL and still choose to ignore it

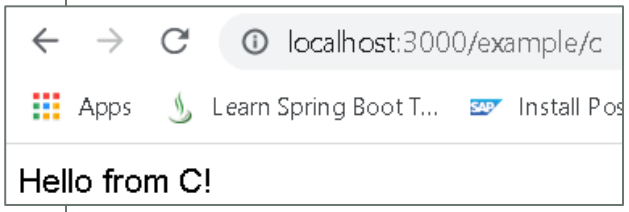
```
app.get('/example/b',  f1 ,  f2 , f3);


……..
function f1(){
     //handle the callback
    next();
}
```

Routing handlers

```
//More than one callback function can handle a route
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...')
  next()
}, function (req, res) {
  res.send('Hello from B!')
})
```




```
//An array of callback functions can handle a route.
var cb0 = function (req, res, next) {
  console.log('CB0')
  next()
}

var cb1 = function (req, res, next) {
  console.log('CB1')
  next()
}

var cb2 = function (req, res) {
  res.send('Hello from C!')
}
app.get('/example/c', [cb0, cb1, cb2])
```



Views, templates, template engines

- A template engine facilitates you to use static template files in your applications.
  - At runtime, it replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
  - This approach makes it easier to design an HTML page.
  - Some popular template engines that work with Express are Pug, Mustache, Haml, Hogan, Swig and EJS.

- Using EJS:
  1. install ejs : npm install ejs –save
  2. Create a folder called “views” in main project folder

```
var express = require("express");
var path = require("path");

var app = express();

app.set("views", path.resolve(__dirname, "views"));
app.set("view engine", "ejs");
```

Tells Express that your views will be in a folder called views

Tells Express that you’re going to use the EJS templating engine

Views, templates, template engines

- Create a file called index.ejs and put it into the “views” directory

index.ejs > ...

```ejs
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello, world!</title>
  </head>
  <body>
    <%= message %>
  </body>
</html>
```

```
//expresslocal -> view_eg.js
var express = require("express");
var path = require("path");
var app = express();

app.set("views", path.resolve(__dirname, "views"));
app.set("view engine", "ejs");

app.get("/", function(request, response) {
    response.render("index", {
        message: "Hey everyone! This is my webpage."
    });
});

app.listen(3000);
```