UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

# Diplomatura en programación web full stack con React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
**www.sceu.frba.utn.edu.ar/e-learning**

# Módulo 3: NodeJS Introducción

# Unidad 2: Express: ruteo, recepción de formularios, envío de mail, archivos estáticos

# Presentación:

Express nos brinda las herramientas para crear y trabajar en NodeJs como servidor. En la unidad anterior vimos cómo instalarlo y dejar corriendo un servidor, ahora vamos a ver como usarlo para recibir peticiones desde un cliente, enviar e-mail y trabajar con archivos estáticos.

# Objetivos:

## Que los participantes:

- Sean capaces de manejar el ruteo en Express.
- Sepan trabajar con archivos estáticos.
- Logren comprender la técnica del envío de mails desde NodeJS.

## Bloques temáticos:

1. Funciones de Express
2. Concepto de ruteo
3. Ruteo en Express
4. Recepción de formularios
5. Envío de mails
6. Archivos estáticos
7. Ejemplos
8. Trabajo Práctico

# Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

## Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

# 1. Funciones de Express

Express es un framework web de los más populares para Node. Proporciona mecanismos para:

- Escritura de manejadores de peticiones HTTP en diferentes rutas
- Integraci con motores de renderización de vistas para generar respuestas mediante la introducción de datos en plantillas
- Establecer ajustes de aplicaciones web tales como indicar puerto de conexión a utilizar, localización de plantillas, etc.
- Añadir procesamiento de peticiones "middleware" adicional en cualquier punto dentro de la secuencia de manejo de peticiones.

Por otro lado, los desarrolladores han creado paquetes de middleware compatibles para abordar casi cualquier problema de desarrollo web.

Hay bibliotecas para trabajar con cookies, sesiones, inicio de sesión de usuario, parámetros URL, datos POST, cabeceras de seguridad, etc.

# 2. Concepto de ruteo

Una petición HTTP es una solicitud que un cliente realiza a un servidor bajo el protocolo HTTP. El protocolo HTTP tiene una cantidad de métodos estándar para realizar estos pedidos. Los 4 métodos principales son:

POST: el que vamos a utilizar para enviar información desde un formulario
GET: con el cual solicitamos información al servidor
PUT: utilizado para solicitar la modificación de datos que ya se encuentran en el servidor
DELETE: donde se solicita que se borre un determinado dato.

En todos los casos desde el cliente va a viajar información hasta el servidor. En el caso del POST la información viaja en el cuerpo del mensaje, del mismo modo suele suceder en el PUT. En el caso del GET y el DELETE, la información suele viajar en el encabezado ya que se trata sólo de un dato de identificación para luego recibir información desde el servidor.

Pensemos en una oficina postal donde hay casillas de correo. La oficina postal sería el puerto de escucha de nuestra aplicación Express. Las casillas postales serían las rutas. Cuando un cliente requiere ser atendido por el servidor, deberá enviar su petición HTML al puerto de escucha habilitado, indicando la ruta (casilla postal) a la cual destina su petición.

Una ruta puede ser por ejemplo “/form” o “/api/cliente”, etc.

Las rutas las define quien esté creando el servidor y deben ser informadas a los cliente ya que en caso contrario, nadie se podrá comunicar con el!

Una petición completa debe indicar la ruta y el método HTTP.

## Método GET

- La info viaja visible directamente en la URL (limitado a 2000 caracteres).

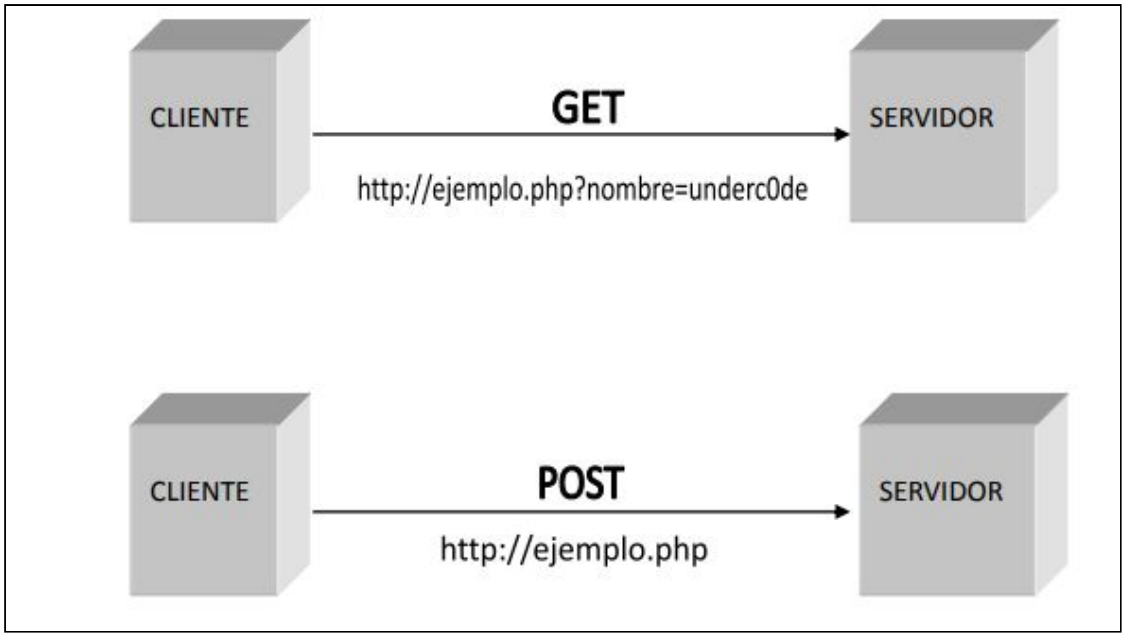
- No se pueden enviar datos binarios (imágenes, archivos, etc.)
- La página web y la información codificada se separan por un interrogante ? y los bloques de datos se separan con &:

```
www.ejemplo.com/index.htm?key1=value1&key2=value2&key3=value3...
```

## Método POST

- La **info** se envía a través del **body del HTTP Request** (no aparece en la URL)
- No tiene límite de cantidad de información a enviar.
- La información proporcionada no es visible, por lo que se puede enviar información sensible.
- Se puede usar para enviar datos binarios (archivos, imágenes...).

## Diferencia entre los 2 métodos



# 3. Ruteo en Express

## Direccionamiento básico

Cómo responde una aplicación a una solicitud del cliente en un determinado punto final?

Tiene 2 componentes básicos:

- una vía de acceso (URI)
- un método de solicitud HTTP específico (GET, POST, etc.)

### Estructura de una ruta

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

**app**: instancia de express
**METHOD**: método de solicitud HTTP (GET, POST, etc.)
**PATH**: vía de acceso al servidor
**HANDLER**: función que se ejecuta cuando se correlaciona la ruta

Ejemplo:

**Responde con Hola! en la página inicial**

```
app.get('/', function (req, res) {
        res.send('Hola!');
});
```

**Responde una solicitud POST en la página de inicio**

```
app.post('/', function (req, res) {
        res.send('Hola pedido POST');
});
```

**direccionamiento all()**

No se deriva con ningún método HTTP.

Se utiliza para responder a cualquier método HTTP.

```
app.all('/productos', function (req, res, next) {
        console.log('Accediendo a la sección de productos ...');
});
```

## Métodos de respuesta

| Método | Descripción |
|---|---|
| res.download() | Solicita un archivo para descargarlo. |
| res.end() | Finaliza el proceso de respuesta. |
| res.json() | Envía una respuesta JSON. |
| res.jsonp() | Envía una respuesta JSON con soporte JSONP. |
| res.redirect() | Redirecciona una solicitud. |
| res.render() | Representa una plantilla de vista. |
| res.send() | Envía una respuesta de varios tipos. |
| res.sendFile | Envía un archivo como una secuencia de octetos. |
| res.sendStatus() | Establece el código de estado de la respuesta y envía su representación de serie como el cuerpo de respuesta. |

**app.route()**

Sirve para incluir en un único lugar los controladores de rutas para una vía

```
app.route('/book')
      .get(function (req, res) {
         res.send('Elegir un libro al azar')
      })
      .post(function (req, res) {
         res.send('Agregar un libro')
      })
```

## Solicitud con parámetros

Se puede hacer de 2 formas:

- http://localhost:3000/usuario/Lorena

- http://localhost:3000/usuario?nombre=Lorena

## Pedido con parámetros

http://localhost:3000/usuario/Lorena

```
app.get('/usuario/:nombre', function (req, res) {
    res.send('Hola '+req.params.nombre);
});
```

## Pedido con parámetros 2

http://localhost:3000/usuario?nombre=Lorena

```
app.get('/usuario', function (req, res) {
        // llamar con ?nombre=algo
        res.send('Hola '+req.query.nombre);
});
```

# 4. Recepción de formularios

Para poder recibir información desde formularios en Express, es necesario incluir el middleware:

```
app.use(express.urlencoded());
```

El método POST es utilizado para recibir la información que el usuario ingresó en un formulario. En el caso del ejemplo que sigue, el formulario es:

```
<!DOCTYPE html>
</html>
  <head>
    <title>Formulario</title>
  </head>
  <body>
    <h1>Formulario</h1>
    <form method="POST" action = "/form">
      <input type = "text" name = "nombre" placeholder = "Nombre"><br/>
      <input type = "text" name = "apellido" placeholder = "Apellido"><br/>
      <input type = "text" name = "mensaje" placeholder = "Mensaje"><br/>
      <input type = "submit" value="Enviar">
    </form>
  </body>
</html>
```

Se recibe la información del formulario por método POST, es decir, que viaja en el cuerpo del mensaje (body) el nombre y contenido de cada campo del formulario.

En **action** se indica a dónde enviar la información del formulario una vez que el usuario oprime el botón. En este caso, se trata de una ruta del mismo servidor. La ruta es “/form”.

Ya en el servidor, la información del formulario se recibe en la variable **req** y como fue mediante método POST, se encuentra en el cuerpo (**body**) de la variable req.

En la variable **res** colocamos el mensaje que el servidor retorna al cliente. En este ejemplo, mediante el método send, se le envía al cliente (el browser), una cadena (string).

```
app.post('/form', function(req, res){
    res.send("Hola " + req.body.nombre + " " + req.body.apellido + " tu mensaje es: " +
req.body.mensaje);
});
```

## Otras formas de devolver información al cliente

### Caso 1

En este caso, enviamos como respuesta también un string pero que contiene código HTML el que será interpretado por el navegador.

Se incluye contenido “dinámico” en el html ya que se agrega el nombre y el mensaje que fueron enviados desde el formulario.

```
app.post('/form', function(req, res){

        var html = "<html><head><title>Hola</title></head><body><h1>Hola " +
req.body.nombre +" </h1><br/><p>Tu mensaje fue: " + req.body.mensaje + "
</p></body><html>";
   res.send(html);
});
```

## Caso 2

Incorporamos otro método: sendFile para enviar un archivo. En este caso, el archivo es un html sencillo que no puede incluir contenido dinámico.

```
app.post('/form', function(req, res){
   res.sendFile('paginas/hola.html' , { root : __dirname});
});
```

## Ejemplo

Tenemos que tener hecho el formulario con method = “POST” y lo ubicamos dentro de alguna carpeta de archivos estáticos

```
<form action="/procesar" method="POST">
    <input type="hidden" name="oculto" val="campo oculto pero no invisible!">
    <div>
            <label for="campofruta">Su fruta preferida: </label>
                    <input type="text" id="campofruta" name="fruta">
    </div>
    <div>
            <button type="submit">Enviar</button>
    </div>
</form>
```

En el servidor

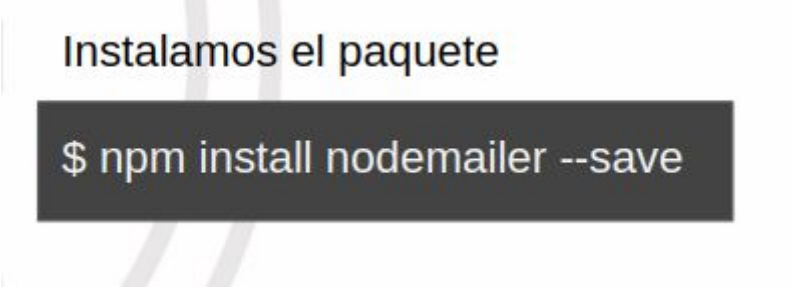
```
var express = require('express');
var app = express();

app.use(express.static('public'));
app.use(express.urlencoded());

app.post('/procesar', function(req, res) {
    var fruta = req.body.fruta;
    var html = 'Tu fruta favorita es: ' + fruta + '.<br>'+
               '<a href="/ejemplo_formulario.html">Probar de nuevo</a>';
    res.send(html);
});

app.listen(3000, function() {
    console.log('Express iniciado en el puerto 3000.');
});
```

# 5. Envío de mails



```
//Creamos el objeto de opciones de envío
var mailOptions = {
  from: 'tucorreo@gmail.com',
  to: 'mi-amigo@yahoo.com',
  subject: 'Asunto Del Correo',
  text: mensaje
};

//Enviamos el mail
transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email enviado: ' + info.response);
  }
});
```

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

//Requerimos el paquete
var nodemailer = require('nodemailer');

var app = express();

//Creamos el objeto de transporte
var transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'tucorreo@gmail.com',
    pass: 'tucontraseña'
  }
});

var mensaje = "Hola desde nodejs...";
```

# 6. Archivos estáticos

Por ejemplo, imágenes, archivos CSS y archivos JavaScript.

Hay que pasar el nombre de la carpeta que contiene los archivos estáticos con la siguiente sentencia:

```
app.use(express.static('public'));
```

Ahora se pueden cargar los archivos que hay en la carpeta 'carpeta', por ejemplo:

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

Express busca los archivos relativos a la carpeta, por lo que el nombre de la carpeta no forma parte de la URL.

Se pueden incluir varias carpetas con archivos estáticos agregando una sentencia por cada carpeta:

```
app.use(express.static('public'));
app.use(express.static('files'));
```

Siendo así, Express busca los archivos en el orden en el que se definen las carpetas.

## Prefijo de vía de acceso virtual

No existe la vía de acceso realmente.

```
app.use('/static', express.static('public'));
```

Entonces ahora puede cargar los archivos que hay en la carpeta public desde el prefijo de vía de acceso /static.

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

### Prefijo de vía de acceso virtual

La vía de acceso que se proporciona sigue siendo relativa a la carpeta desde donde inicia el proceso node.

Por eso, si se ejecuta la aplicación desde cualquier otra carpeta, es más seguro utilizar la vía de acceso absoluta de la carpeta a la que se desea dar servicio:

```
app.use('/static', express.static(__dirname + '/public'));
```

# 7. Ejemplos

En el siguiente ejemplo se puede ver una sencilla aplicación Express que escucha en el puerto 3000 y recibe peticiones HTML en la ruta '/hola'. En dicha ruta recibe 3 tipos de peticiones diferentes: get, post y put.

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

app.get('/hola', function (req, res) {
  res.send('Hola mundo en GET');
});

app.post('/hola', function (req, res) {
  res.send('Hola mundo en POST');
});

app.put('/hola', function (req, res) {
  res.send('Hola mundo en PUT');
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

# 8. Trabajo Práctico

Crear un formulario de registración con los siguientes datos: **nombre, apellido, edad, número de celular, país de nacimiento, país de residencia.**

Recibir los datos en el servidor y armar otra página de respuesta que incluya los datos del usuario y un enlace a la página de registración nuevamente.

## Bibliografía utilizada y sugerida

Express (n. d.) Recuperado de: https://expressjs.com/es/

MDN - JavaScript (n.d.) Recuperado de:

https://developer.mozilla.org/es/docs/Web/JavaScript

NodeJs (n. d.) Recuperado de: https://nodejs.org/es/

NodeJS Documentacion (n. d.) Recuperado de: https://nodejs.org/es/docs/

NPM (n. d.) Recuperado de: https://www.npmjs.com/package/page

Wikipedia - NodeJS (n. d.) Recuperado de: https://es.wikipedia.org/wiki/Node.js

## Lo que vimos:

Express: ruteo, recepción de formularios, archivos estáticos, envío de mails.

## Lo que viene:

Bases de datos: MySQL.