UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Desarrollo con Node 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

Unidad 2:

Express, estructura de directorio y ruteo

Presentación:

En esta unidad vemos qué es una api REST, qué utilidades nos brinda y cómo se desarrollan las aplicaciones actuales. Finalmente, aprendemos a utilizar el framework Express para desarrollar API REST con tecnología javascript y node js.

Objetivos:

Que los participantes:

- Comprendan qué es y para qué utilizar una API REST.
- Aprendan cómo instalar y configurar express con node js.
- Desarrollen un servicio de API REST con express.

Bloques temáticos:

- API REST.
- Express – De qué estamos hablando.
- Express – Primeros pasos.
- Express – Hello World!.
- Express – Generator.
- Express – Estructura de directorio.
- Express – Direccionamiento.

Consignas para el aprendizaje colaborativo

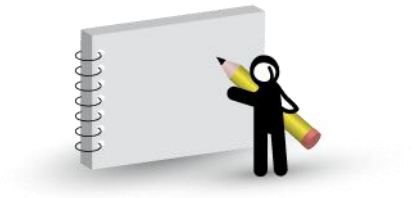
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- 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.



Tomen nota:

La propuesta de actividades

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.

API REST

Definición

REST cambió por completo la ingeniería de software a partir del 2000. Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por Roy Fielding, el padre de la especificación HTTP y uno los referentes internacionales en todo lo relacionado con la Arquitectura de Redes, en su disertación 'Architectural Styles and the Design of Network-based Software Architectures'. En el campo de las APIs, REST (Representational State Transfer- Transferencia de Estado Representacional) es, a día de hoy, el alfa y omega del desarrollo de servicios de aplicaciones.

En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Twitter, YouTube, los sistemas de identificación con Facebook… hay cientos de empresas que generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet.

Buscando una definición sencilla, REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

Características

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo cliente-caché-servidor sin estado: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.

- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).

- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento él identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

- Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

- Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

- Uso de hipermedios: hipermedia es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Ventajas

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

- Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

- La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Ejemplo

A continuación vemos un ejemplo de request y response con la api de Mercado Libre:

Puedes consultar qué modos de envío tiene habilitado el usuario de la siguiente manera:

```
GET https://api.mercadolibre.com/users/:user_id?access_token=
```

Esto devolverá gran cantidad de información sobre el usuario autenticado, incluido el atributo shipping_modes.

Respuesta:

```
"shipping_modes":[
        "custom",
        "not_specified",
        "me1",
        "me2",

]
```

Express – De qué estamos hablando

Express es sin duda el framework más conocido de node.js, es una extensión del poderoso connect y está inspirado en sinatra, además es robusto, rápido, flexible, simple…

Sin duda el éxito de express radica en lo sencillo que es usarlo, y además abarca un sin número de aspectos que muchos desconocen pero son necesarios.

De entre las tantas cosas que tiene este framework podemos destacar:

- Session Handler
- 11 middleware poderosos así como de terceros.
- cookieParser, bodyParser…
- vhost
- router

Express – Generator

Utilizando el cli de generator podremos crear el esqueleto de nuestra aplicación de forma rápida y sencilla.

Para instalar esta herramienta debemos ejecutar:

npm install express-generator -g

```
G:\sites\node\utn>npm install express-generator -g
```

Con el comando **express –h** podemos visualizar las opciones que nos brinda esta herramienta:

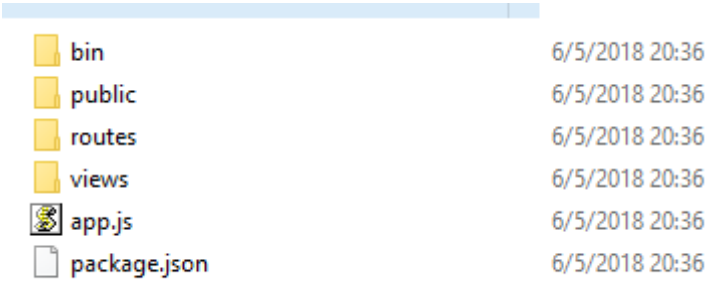
```
G:\sites\node\utn>express -h

  Usage: express [options] [dir]


  Options:

        --version        output the version number
    -e, --ejs            add ejs engine support
        --pug            add pug engine support
        --hbs            add handlebars engine support
    -H, --hogan          add hogan.js engine support
    -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
        --no-view        use static html instead of view engine
    -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
        --git            add .gitignore
    -f, --force          force on non-empty directory
    -h, --help           output usage information
```

Crear nuestra primera aplicación con generator

Abrimos la consola y nos ubicamos en el directorio en el cual queremos crear nuestra aplicación, luego ejecutamos:

```
G:\sites\node>express --view=ejs myapp_utn

   create : myapp_utn\
   create : myapp_utn\public\
   create : myapp_utn\public\javascripts\
   create : myapp_utn\public\images\
   create : myapp_utn\public\stylesheets\
   create : myapp_utn\public\stylesheets\style.css
```

Se creara el directorio de nuestra app utilizando las plantillas de vista **pub**



Ingresamos al directorio en el cual se creó nuestra aplicación (desde líneas de comandos) y ejecutamos: **npm install**

```
G:\sites\node\myapp>npm install
[                ......] - extract:pu
```

Por último ejecutamos:

Desde Windows:

```
G:\sites\node\myapp>set DEBUG=myapp:* & npm start
```

Desde Linux o MAC

```
$ DEBUG=myapp:* npm start
```

Cada vez que queramos "levantar" nuestro servidor en el puerto 3000 debemos ejecutar este último comando

Si en la barra de direcciones de nuestro navegador colocamos:

http://localhost:3000/

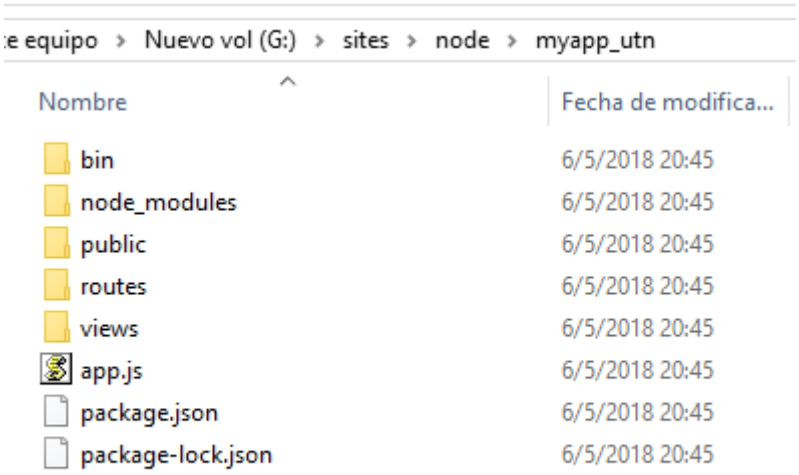
Accederemos al contenido de nuestra aplicación.

Express

Welcome to Express UTN

Express – Estructura de directorio

Al crear nuestra aplicación en node visualizaremos la siguiente estructura de directorio:



- **bin:** Directorio propio de express, en el cual podremos visualizar la creación de un servidor en node js
- **node_modules:** Carpeta propia de node, allí se alojaran todos los modulos instalados con **npm install**
- **public:** Se aloja el contenido publico
    - **Imágenes**
    - **Javascript**
    - **CSS**
- **routes:** Se alojan los archivos que oficiaran de “ruteadores”, es decir que serán visualizados de acuerdo a la URL que accedamos
- **views**: Se alojan las vistas de nuestra aplicación. En este caso son de tipo **ejs**

Express – Direccionamiento

El direccionamiento hace referencia a la determinación de cómo responde una aplicación a una solicitud de cliente en un determinado punto final, que es un URI (o una vía de acceso) y un método de solicitud HTTP específico (GET, POST, etc.). Cada ruta puede tener una o varias funciones de manejador, que se excluyen cuando se correlaciona la ruta. La definición de ruta tiene la siguiente estructura:

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

Donde:

- app es una instancia de express.
- METHOD es un método de solicitud HTTP (GET, POST, PUT, DELETE).
- PATH es una vía de acceso en el servidor. Ejemplo **“/”, “users”, “users/1”**
- HANDLER es la función que se ejecuta cuando se correlaciona la ruta. Es la función de callback que se ejecutara al ingresar por esa ruta.

Por ejemplo en el directorio **route** encontramos el archivo **index.js**, el mismo tendrá la definición de nuestra ruta por default (es decir cuando accedemos a la home de nuestro sitio “/”)

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

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

En este visualizaremos el contenido de la vista **index,** cuando realizamos una solicitud de tipo get.

En el siguiente ejemplo vemos un caso similar pero para una petición de tipo POST:

```
// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Los objetos **req** y **res** contendrán la información del request y reponse.

Como vemos en ambos casos el método **send** corresponde al objeto **response**

Cargar funciones de middleware

Las funciones de middleware son funciones que se ejecutaran en cada request y response que reciba nuestra aplicación, por cada una de estas interacciones se ejecutaran las funciones definidas

Para ello utilizamos el método **all**

```
app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
});
```

En el siguiente ejemplo, el manejador se ejecutará para las solicitudes a “/secret”, tanto si utiliza GET, POST, PUT, DELETE, como cualquier otro método de solicitud HTTP soportado. Con el método **next** se ejecuta el contenido del manejador definido para la petición realizada, es decir primero se ejecutara esta función y luego la definida en el route para dicha petición.

Vías de acceso de rutas con expresiones regulares

```
app.get('/ab?cd', function(req, res) {
  res.send('ab?cd');
});
```

Esta vía de acceso de ruta coincidirá con acd y abcd.

```
app.get('/ab+cd', function(req, res) {
  res.send('ab+cd');
});
```

Esta vía de acceso de ruta coincidirá con abcd, abbcd, abbbcd, etc.

```
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});
```

Esta vía de acceso de ruta coincidirá con abcd, abxcd, abRABDOMcd, ab123cd, etc.

```
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
```

Esta vía de acceso de ruta coincidirá con /abe y /abcde.

N funciones de llamadas

Podremos definir una pila de funciones de llamadas encadenadas, por ejemplo:

Una función de devolución de llamada individual puede manejar una ruta.

```
app.get('/example/a', function (req, res) {
  res.send('Hello from A!');
});
```

Más de una función de devolución de llamada puede manejar una ruta (asegúrese de especificar el objeto next).

```
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!');
});
```

Se ejecuta primero el contenido de la primer funciona (donde está el console.log()) y luego el de la segunda función (en donde está el send())

En este caso es importante utilizar el llamado a la función next().

Una matriz de funciones de devolución de llamada puede manejar una ruta.

```
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]);
```

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.

Router

Utilice la clase express.Router para crear manejadores de rutas montables y modulares. Una instancia Router es un sistema de middleware y direccionamiento completo; por este motivo, a menudo se conoce como una “miniaplicación”.

El siguiente ejemplo crea un direccionador como un módulo, carga una función de middleware en él, define algunas rutas y monta el módulo de direccionador en una vía de acceso en la aplicación principal.

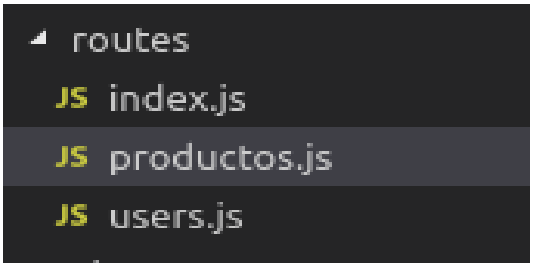
```
var express = require('express');
var router = express.Router();

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

```
var birds = require('./birds');
...
app.use('/birds', birds);
```

Como vemos con el método **use** podemos definir el middleware para esta ruta

Crear una nueva ruta

Debemos crear el archivo correspondiente en la carpeta routes



Debemos incluir el modulo “Express” utilizando require y acceder al método router.

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

/* GET home page. */
router.get('/:id([0-9]+)', function(req, res, next) {
  console.log(req.query.nombre);
  console.log(req.params.id);
  res.render('catalogo', { title: 'Productos' });
});

module.exports = router;
```

Con el método router podemos acceder a métodos según los verbos del protocolo http

- Router.get
- Router.post
- Router.put
- Router.delete

Ruteo

El primer parámetro de los métodos en la URL con la cual se realizara el match.

El match lo realiza en base al use de app.js.

Ejemplo

App.js

```
app.use('/productos', productosRouter);
```

Productos.js (dentro de directorio router)

```
router.get('/',
```

Va a realizar el match con la url **/productos/.**

/productos lo toma del *app.js*

/ lo toma del archivo *productos.js*

Si en el archivo productos.js ahora sumamos

```
router.get('/destacados',
```

En este último caso el match será con **/productos/destacados**

/productos -> *app.js*

/destacados -> *productos.js*

Parámetros por URL

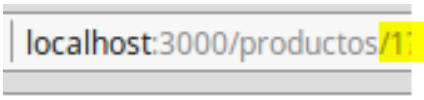
En la declaración de las rutas, podemos indicar la recepción de un parámetro por URL

```
router.get('/:id([0-9]+)', function(req, res, next) {
```

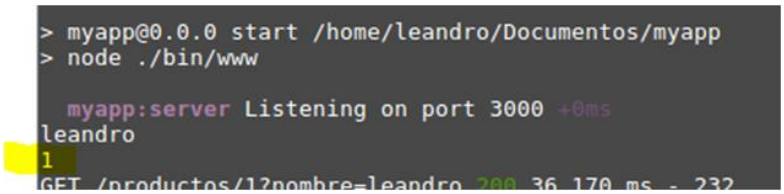
En este caso el dato enviado en la Url se asignara a la variable id.

También se especifica que la misma debe ser solo numérica (expresión regular)

Ejemplo si navegamos la siguiente URL



Vemos el siguiente dato en la consola

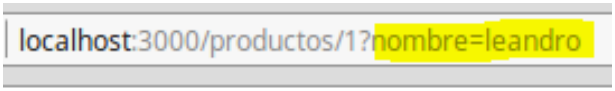


Del lado del router tenemos el siguiente código

```
router.get('/:id([0-9]+)', function(req, res, next) {
  console.log(req.params.id);
  res.render('catalogo', { title: 'Productos', subtitl
});
```

Como vemos el objeto **req.params** tiene la información de las variables mapeadas desde la URL. En este ejemplo **id**

Si recibimos parámetros por query string



Lo recibimos de la siguiente manera

```
router.get('/:id([0-9]+)', function(req, res, next) {
  console.log(req.query.nombre);
  res.render('catalogo', { title: 'Productos', subtitle:'
});
```

El objeto query tiene la información enviada por query string en la URL

En el caso de recibir datos en el body (ejemplo peticiones por POST) los mismos se reciben en **req.body**

```
router.post('/', function(req, res, next) {
    let producto = new productosModel({
        name: req.body.name,
        description:req.body.description,
        sku: req.body.sku,
        price: req.body.price,
        quantity:req.body.quantity,
        category: req.body.category
    })
    let data = await producto.save();
    res.status(201).json({"stauts":"ok","data":data});
```

Ver ejemplo de request en la sección de POSTMAN

Estructurar el código

Para mejorar la estructura del código vamos a crear un nuevo directorio “controllers” al mismo nivel de “routes”

Dentro de ese directorio crearemos un archivo por cada controlador. El objetivo es que nos quede un controlador por cada servicio.

Por ejemplo creamos el controlador “productosController.js”

```
module.exports = {
    getAll: async function(req, res, next) {
        let productos = {"id":1,"nombre":"test"}
        res.status(200).json(productos)
    }
}
```

Luego desde el archivo generado anteriormente “productos.js” dentro del directorio routes nos queda lo siguiente:

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

var productosController = require("../controllers/productosController")

/* GET home page. */
router.get('/', productosController.getAll);

module.exports = router;
```

De esta manera desde “productos.js” realizamos un require del modulo exportado en “productosController”.

Luego en cada ruta dejamos el primer parámetro para el match de la URL, y el segundo parámetro en lugar de tener la función de callback con la resolución del request tiene un llamado al método “getAll” del controlador.

Quedando en los archivos de routes la lógica para el match de las url y en los controladores la lógica de aplicación que en la unidad siguiente conectaremos con los modelos para el uso de base de datos.

Postman

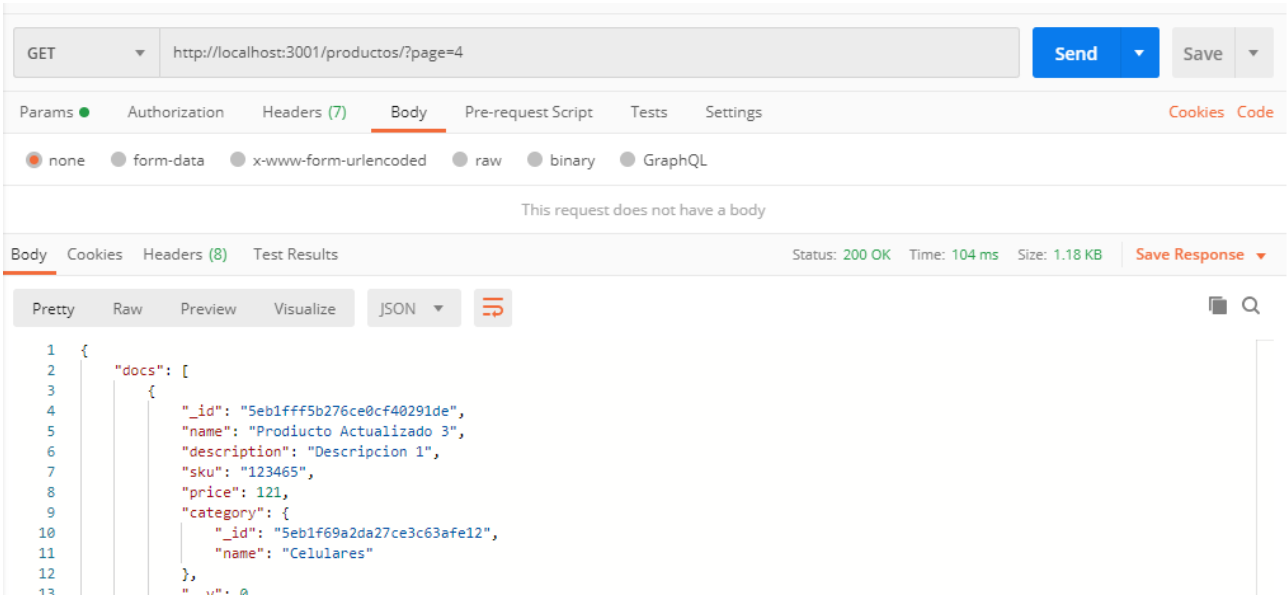
Es un cliente REST, el cual nos va a permitir realizar consultas a la api desarrollada con express.

Debemos instalar postman desde:

https://www.postman.com/downloads/

Una vez instalado vamos a realizar request utilizando URL + Verbo http

GET



En el combo de arriba a la izquierda podemos seleccionar el verbo, en este caso GET

A la derecha colocamos la URL sobre la cual realizaremos el request. En el ejemplo es **http://localhost:3001/productos** (Pero puede variar de acuerdo a la configuración de nuestra api, por defecto express atiende solicitudes en el puerto 3000)

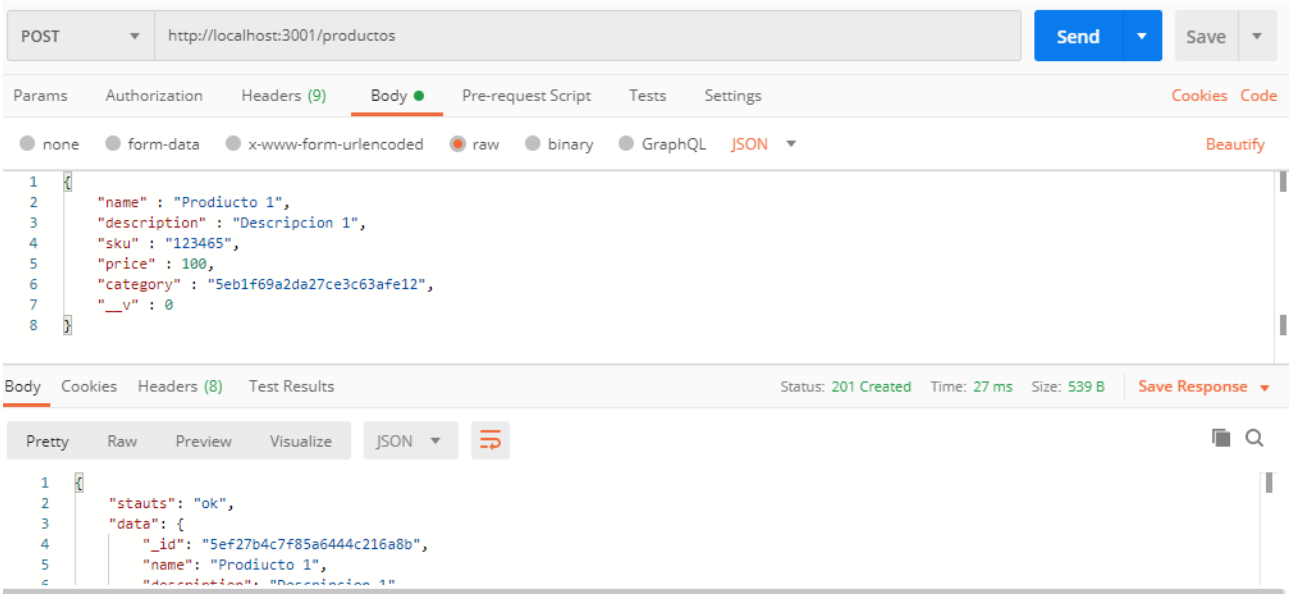
Debajo visualizamos:

- Params
- Autorization
- Headers
- Body

En el caso del GET no enviaremos ningún parámetro.

Debajo de todo vemos el body del response, allí visualizaremos la respuesta del servicio de express

POST



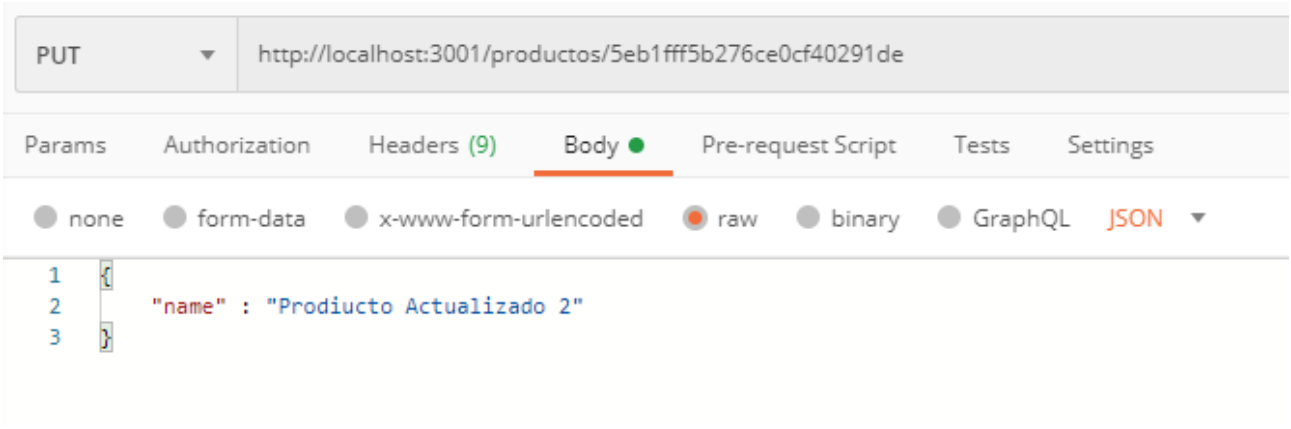
En el caso del POST lo que difiere es el body enviado en el request.

Para el mismo debemos seleccionar la opción “raw” y en el combo de la derecha “JSON”

En el body enviaremos un json, el cual obtendremos en express en **req.body**

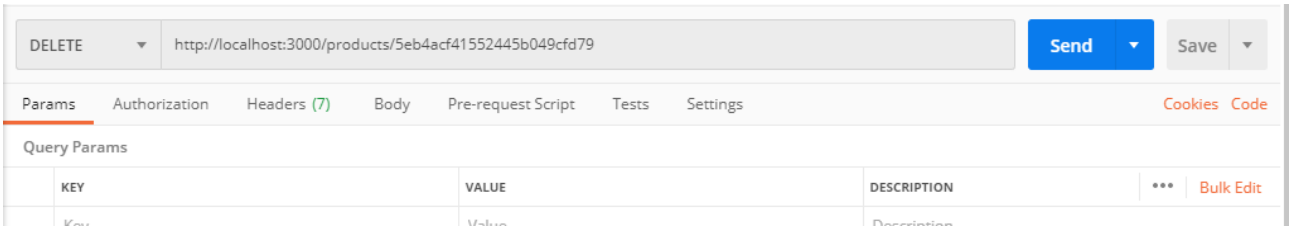
PUT

Al igual que POST se deben enviar un json en el body del request. También podemos enviar el id del producto a modificar por parámetro en la url. Por ejemplo



DELETE

Este caso es más bien similar al GET, es decir solo enviamos parámetro por URL pero no body en el request:



Bibliografía utilizada y sugerida

http://expressjs.com

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/skeleton_website

https://geekytheory.com/introduccion-a-express-js

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

Lo que vimos:

En esta unidad vimos cómo desarrollar utilizando el framework express. Como consumir los servicios utilizando un cliente REST como POSTMAN

Lo que viene:

En la unidad siguiente veremos qué es una base de datos no sql, qué es mongo db, cómo instalar y configurar nuestra base de datos en mongo db y cómo conectar express con mongo db.