UTN FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Desarrollo de Software
Notas de Clase

EXPRESS (*)

Express es un framework de desarrollo web para Node.js que proporciona una arquitectura de aplicaciones web minimalista y flexible. Express se utiliza para crear aplicaciones web y APIs de manera rápida y sencilla, y es una de las opciones más populares para el desarrollo web en Node.js.

Express proporciona una serie de funciones que facilitan la creación de rutas de servidor, la gestión de solicitudes y respuestas HTTP, y la integración de middleware de terceros para implementar características adicionales. Los desarrolladores pueden utilizar Express para crear aplicaciones web que implementen diferentes patrones de arquitectura, como MVC (Model-View-Controller) o Middleware-Based.

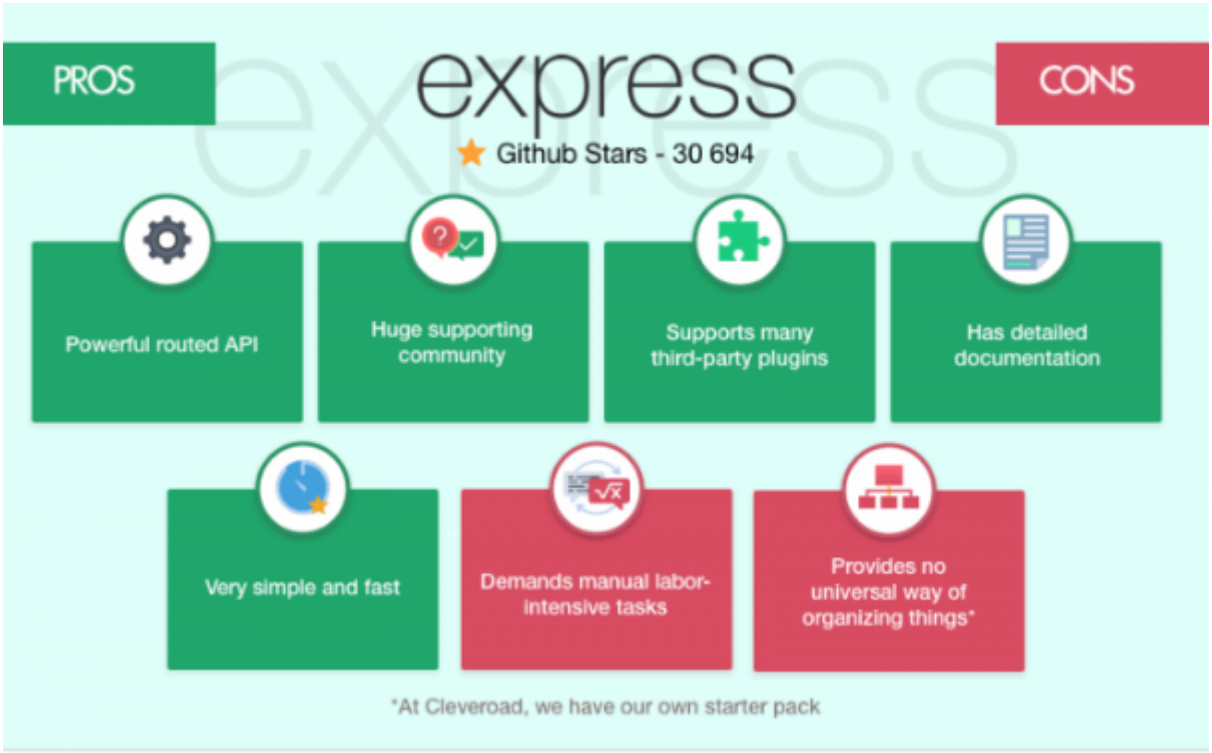
Entre las características principales de Express se incluyen:

- Enrutamiento: Express proporciona una API sencilla y flexible para definir rutas de servidor y responder a solicitudes HTTP.
- Manejo de middleware: Express permite utilizar y crear middleware de manera sencilla, lo que permite integrar funcionalidades adicionales a la aplicación web.
- Renderizado de vistas: Express.js admite el renderizado de vistas en el lado del servidor. Se pueden utilizar motores de plantillas como EJS, Pug y Handlebars para renderizar vistas.
- Acceso a la base de datos: Express puede utilizarse junto con una variedad de bases de datos populares, como MongoDB o MySQL, para crear aplicaciones web con funcionalidades avanzadas.
- Integración con otras librerías de Node.js: Express es compatible con una variedad de librerías y paquetes de Node.js, lo que permite crear aplicaciones web robustas y escalables.
- unopinionated: "no dogmático": lo que significa que contrariamente a otros frameworks nos dará total libertad para plantear nuestro producto mientras que todas las decisiones acerca de la arquitectura, las mejores prácticas y las convenciones deberán ser establecidas por nosotros mismos.

En general, Express es una herramienta muy poderosa y flexible para el desarrollo web en Node.js, que facilita la creación de aplicaciones web y APIs de alta calidad y escalables.

Nota: documentación oficial de express: http://expressjs.com/es/

Pros y contras:



Instalando Express (*)

- En primer lugar, instale el marco Express globalmente usando NPM para que pueda usarse para crear una aplicación web usando la terminal.
- Línea de comando para instalar express:

```
npm install express --save
```

El comando anterior ejecuta una instalación local en el directorio si es en sistema operativo windows:

```
C:\Program Files\nodejs\node_modules
```

Ejemplo hola mundo (*)

La siguiente es una aplicación Express muy básica que inicia un servidor y escucha en el puerto 3000 para las conexiónes. Esta aplicación responde con Hola Mundo! a las solicitudes a la raíz del sitio. Para cualquier otra ruta, responderá con un código: 404 No encontrado.

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
```

```
  res.send('Hola Mundo!')
})

app.listen(port, () => {
  console.log(`Aplicacion Hola Mundo escuchando en el puerto ${port}`)
})
```

Código online: https://stackblitz.com/edit/dds-express-hola-mundo

Para ejecutar localmente siga los siguientes pasos:

- cree un carpeta para la aplicación
- inicialice el proyecto con

```
npm init
```

- cree en dicha carpeta un archivo llamado index.js que contenga el código del ejemplo
- iniciar el servidor, ejecutando desde consola el comando:

```
node index.js
```

Usted deberá ver en consola:

```
Aplicación de ejemplo, escuchando en <http://127.0.01:3000>
```

- finalmente para correr la aplicación:
  - desde el explorador URL:

```
http://localhost:3000/
```

  - desde consola:

```
curl --location --request GET  ‘<http://localhost:3000/>’
```

Observaciones iniciales del código:

- *app*: es el objeto de la aplicación. Se usa este nombre convencionalmente para la aplicación Express. Créelo llamando a la función express() de nivel superior exportada por el módulo Express:
- *app.get*: Enruta las solicitudes HTTP GET a la ruta especificada con las funciones de devolución de llamada especificadas

- *res.send*: Envía la respuesta HTTP

Solicitud (Request) y Respuesta (Response) (*)

La aplicación Express utiliza una función de devolución de llamada (callback) cuyos parámetros son los objetos de solicitud (req) y respuesta (res).

```
app.get('/', función (req, res) {

// --

});
```

Objeto Request: el objeto de solicitud representa la solicitud HTTP y tiene, entre otra, propiedades para la cadena de consulta de la solicitud (query string), los parámetros, el cuerpo (body), los encabezados (Headers) HTTP, etc.

Objeto Response: el objeto de respuesta representa la respuesta HTTP que envía una aplicación Express cuando recibe una solicitud HTTP.

Puede imprimir, mediante el comando **console.log**, los objetos req y res que brindan mucha información relacionada con la solicitud y respuesta HTTP, incluidas cookies, sesiones, URL, etc.

Enrutamiento básico (*)

Hemos visto una aplicación básica que atiende solicitudes HTTP para la página de inicio o la raíz del sitio. El enrutamiento se refiere a determinar cómo responde una aplicación a una solicitud de un cliente a un punto final (EndPoint) en particular, que es un URI (o ruta) y un método o verbo de solicitud HTTP específico (GET, POST, etc.).

Ampliaremos nuestro programa Hola Mundo para manejar más variedades de solicitudes HTTP.


```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.redirect('/Inicio'); // redireccion
});
```

```
app.get('/Inicio', (req, res) => {
  res.send('Pagina de Inicio!');
});
app.get('/AcercaDe', (req, res) => {
  res.send('Pagina de Acerca!');
});
app.use('', (req, res) => {
  res.send('Pagina no econtrada!');
});

app.listen(port, () => {
  console.log(`Aplicacion Hola Mundo escuchando en el puerto ${port}`);
});
```

Código online: https://stackblitz.com/edit/dds-express-rounting

Observaciones: las rutas, por defecto, no son sensitivas a mayúsculas/minúsculas.

En el ejemplo anterior, todas las rutas usadas, utilizan el mismo método (o verbo) GET, a continuación veremos que podemos utilizar diferentes métodos según lo recibido en la solicitud.

La sintaxis general de una ruta en express es la siguiente:

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

En donde,

1. **app** es una instancia del módulo express
2. **METHOD** es un método/verbo de la solicitud HTTP (GET, POST, PUT, DELETE, etc)
3. **PATH** es una ruta en la solicitud.
4. **HANDLER** es la función ejecutada cuando la ruta se corresponde.

Ejemplo:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/articulos', (req, res) => {
  res.send('buscando articulo!');
});
app.post('/articulos', (req, res) => {
  res.send('agregando articulo!');
});
app.put('/articulos', (req, res) => {
  res.send('actualizando articulo!');
});
```

```
app.delete('/articulos', (req, res) => {
  res.send('eliminando articulo!');
});

app.listen(port, () => {
  console.log(`Aplicacion DDS escuchando en el puerto ${port}`);
});
```

Código online: https://stackblitz.com/edit/dds-express-metodos

- **Ejecutar localmente**:
  - bajar el código en una carpeta vacía, ejecutar el comando **npm install** y luego la instrucción **node index.js**
  - test local
    - desde el explorador URLs: http://localhost:3000/articulos

```
        Atención: Desde la URL del navegador, sólo podemos probar las rutas que usan el verbo “get”
```

- **curl**
  - curl ‘http://localhost:3000/articulos’
  - curl -X POST ‘http://localhost:3000/articulos’
  - curl -X PUT ‘http://localhost:3000/articulos’
  - curl -X DELETE ‘http://localhost:3000/articulos’

```
Para una prueba exhaustiva, nos sería de mucha comodidad utilizar la aplicación Postman (https://www.postman.com)
```

Importante si en lugar de usar un un metodo/verbo especifico, usamos la funcion use, ej App.use(ruta, funcionControlador) la misma respondera a todos los metodos/verbos.

Respondiendo las peticiones (*)

Las peticiones recibidas en nuestro servidor pueden devolver datos en diversos formatos. Si nuestro desarrollo está orientado hacia un servidor de web api, la mayoría de las respuestas devolverán datos en formato json y un código de estado adecuado a la misma. Para lo cual haremos uso de algunas funciones que nos ofrece el objeto response:

- **res.send([body]);** Envía la respuesta HTTP.

  Ejemplo Nº 1:

  ```
  res.send('<p>algun html</p>');
  ```

Ejemplo Nº 2:

```
res.send({ usuario: 'tobi' })
```

- **res.json([body]);**

  Envía una respuesta JSON. Este método envía una respuesta (con el tipo de contenido correcto -200 OK-) que es el parámetro convertido a una cadena JSON usando **JSON.stringify()**.

  El parámetro puede ser de cualquier tipo JSON, incluidos objeto, matriz, cadena, booleano, número o nulo, y también puede usarlo para convertir otros valores a JSON.

- **res.status(code)**

  Establece el estado HTTP para la respuesta. Es un alias encadenable de response.statusCode de Node.

```
app.delete('/articulos', (req, res) => {
  //aqui van el código que hace la eliminación
  res.status(200).json('eliminando articulo!') //devuelve estado especifico,
mas json
});
```

- **res.sendStatus(statusCode)**

  Establece el código de estado HTTP de respuesta en statusCode y envía el mensaje de estado registrado como cuerpo de respuesta de texto. Si se especifica un código de estado desconocido, el cuerpo de la respuesta será solo el número de código.

  Ejemplo:

```
app.use('', (req, res) => {
  res.sendStatus(404);  //devuelve solo codigo de estado - 404 Not Found
});
```

Código online: https://stackblitz.com/edit/dds-express-response

- Ejecutar localmente: bajar el código, ejecutar npm install y luego node index.js
  - test local
    - desde el explorador URLs:
      - http://localhost:3000/articulos
    - curl
      - curl 'http://localhost:3000/articulos'
      - curl -X DELETE 'http://localhost:3000/articulos'
      - curl http://localhost:3000/cualquier_URL_inexistente
    - postman

Las solicitudes a nuestro servidor, vendrán con un conjunto de información del navegador del cliente, que deberemos analizar para poder responder correctamente a las misma para lo cual haremos uso de un conjunto de middlewares y propiedades específicas del objeto request:

- middlewares provistos por expresss para poder interpretar los datos, en diferentes formatos, enviados en las peticiones del cliente

```
app.use(express.text());
      // permite leer texto-> req.body

app.use(express.urlencode({extended: false}));
      // permite leer datos formulario get: req.query.CampoX
      // permite leer datos formulario post: req.body.CampoX

app.use(express.json());
      // permite leer json: req.body.CampoX
```

- Propiedades del **request** a través de las cuales recibiremos la información enviada por el cliente:
  - params
  - query
  - body
  - headers

request.params (*)

Esta propiedad es un objeto que contiene propiedades asignadas a los 'parámetros' de la ruta nombrada.

Por ejemplo, si tiene la ruta http://localhost:3000/usuario/:nombre, entonces la propiedad 'nombre' está disponible como **req.params.nombre**. El valor por defecto de este objeto es {}.

Veamos otro ejemplo:

- URL petición con un sólo parámetro: http://localhost:3000/articulos/1

```
      //codigo servidor
      app.get('/articulos/:id', (req, res) => {
        res.send('Se solicito el articulo ' + req.params.id);
      });
```

- URL petición con múltiples parámetros: http://localhost:3000/ventas/articulo/15/cliente/101

```
//codigo servidor
app.get('/ventas/articulo/:IdArticulo/cliente/:IdCliente',(req, res) => {
  const IdArticulo= req.params.IdPedido;
  const IdCliente'= req.params.IdCliente';
  res.send(´Se solicitó las ventas del articulo: ${IdArticulo} para el
```

```
  cliente ${IdCliente}´ );
  });
```

request.query (*)

Esta propiedad es un objeto que contiene a su vez una propiedad para cada parámetro de cadena de consulta en la ruta. Estos parametros siempre estan al final de la URL despues del signo de interrogacion "?". Cuando el middleware ***express.urlencode*** está desactivado, es un objeto vacío **{}**; de lo contrario, contendrá propiedades que representan los parámetros enviados en la URL.

Veamos un ejemplo:

URL petición: http://localhost:3000/articulos?Nombre=aire&Activo=false

```
app.get('/articulos', (req, res) => {
  console.log(req.query.Nombre);
  console.log(req.query.Activo);
  res.send(´Se solicito los artículos con Nombre: ${req.query.Nombre} y con
Activo: ${req.query.Activo}´;
});
```

```
Observe: los parámetros se envían en la URL, luego de la ruta y precedidos por un
signo de pregunta “?”
con el formato de clave=valor separados entre si mediante el operador “&”
```

request.body (*)

Contiene pares clave-valor de datos enviados en el cuerpo de la solicitud. De forma predeterminada, no está definido y se completa cuando usa un middleware de análisis del cuerpo, como **express.json()** o **express.urlencoded()**.

Veamos un ejemplo:

```
router.post('/api/articulosfamiliasmock/', (req, res) => {
  const { IdArticuloFamilia } = req.body;
  console.log(IdArticuloFamilia);
  const { Nombre } = req.body;
  console.log(Nombre);
  // código que permite almacenar
})
```

express.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 se crea un *"router"* como un módulo, define una ruta o varias rutas que tienen alguna relación en el archivo **articulos.js** y posteriormente se monta el módulo de router en la aplicación principal en el archivo **index.js**.

- Archivo: **articulos.js**

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

router.get("/api/articulos", async function (req, res, next) {
  //Código del método
});

router.get("/api/articulos/:id", async function (req, res, next) {
  //Código del método
});
```

- archivo: **index.js**

```
const articulosRouter = require("./routes/articulos");
app.use(articulosRouter);
```