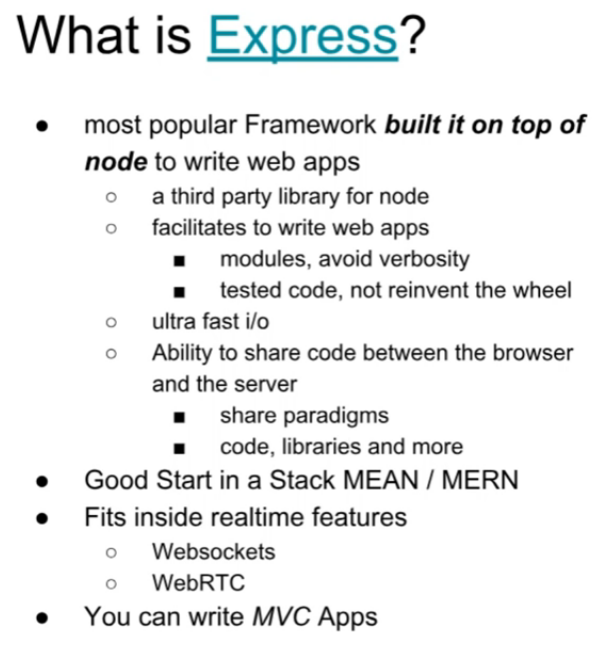
**Curso basico de express**

****

Es un framework de node.js y javascript que nos permite crear aplicaciones web **del lado del servidor**.

**-Cuando trabajamos en una aplicación web necesitamos enviar html, css y javascript, y el servidor es el encargado de enviar esos archivos a los usuarios y recibir datos de ellos.**

**-Todas las funcionalidades que podemos darle a una aplicación, ya existen en algún framework y esto nos facilita el trabajo para no tener que andar escribiendo código sino directamente usarlo.**

**-Existen frameworks especializados para** Procesar imágenes, enviar video, procesar streams, saber como funciona el enrutamiento de una aplicación de servidor, para enviar archivos estáticos, para conectarte a una base de datos, y mas.

**-Express se rige de las reglas de node.js (Single threading, Codigo asíncrono)**

**-Habilidad para compartir código entre el navegador y el servidor**

**MEAN: MONGO DB, EXPRESS** (BACKEND)**, ANGULAR**(FRONT END)**, NODE**

**MERN: MONGO DB, EXPRESS** (BACKEND)**, REACT**(FRONT END)**, NODE**

**Realtime Applications WEBSOCKETS, WEBRTC, MVC APPS (model view controller)**

**Instalacion de express**

**Instalacion: (se instalara solo en el proyecto actual)**

En consola **🡪 npm I express**

En la carpeta **node\_modulos** 🡪 podemos ver los modulos instalados en este proyecto y podemos investigar como esta escrito el código de express

**SERVIDOR BASICO DE NODE (ejemplo)**

**Abrir consola directamente en visual studio code 🡪 ctrl + shift + p (aparece esto>) 🡪 “terminal”🡪 créate new integrated terminal**

**1-** Creamos index.js

**2-**lamamos al modulo http

const http = require ('http'); //llamamos al modulo http

const server = http.createServer((req,res) =>{ //createServer función de http

 //maneja todas las peticiones que vienen del navegador,

    res.status = 200; //codigo de estado es 200

    res.setHeader ('contentType','text/plain');

//para especificar que tipo de contenido le voy a enviar, en este caso texto PLANO

    res.end('Hello World')

});

//guardamos todo lo anterior en la constante server

server.listen(3000, ()=>{

    console.log('server on port 3000');

})

Esta es una aplicaciones sencilla, si queremos recibir imágenes, tener formularios para que el usuario ingrese sus datos, queremos empezar a subir videos, se va complicando mucho mas, y tendríamos que escribir nosotros mismos el código, y para eso ya existen los **frameworks**

**Es bueno empezar creando un archivo package jason escribiendo 🡪 npm init**

**ROUTING**

const express = require('express');

const app =  express();

app.get('/', (req,res) =>{

 //ruta a inicial de la pagina

    res.send('Pagina de inicio');

});

app.get('/about',(req, res)=>{

// ruta que se ejecuta en el navegador con localhost:3000/about

    res.send('Soy Federico Marchetta, tengo 26 años');

})

app.get('/culo', (req, res) =>{

//otra ejemplo

    res.send('culo culo culo culo, es un perro pelotuchi');

})

app.listen(3000, () =>{

    console.log('server on port 3000');

});

**Nodemon 🡪 sirve para no tener que tocar ctrl + c x2 para reiniciar el servidor, es decir que nodemon va a estar vigilando mi servidor y cuando yo presione ctrl + s para guardar el proyecto automáticamente va a refrescar el servidor**

**Instalacion: npm i nodemon –D (para instalarlo en dev dependencies)**

**Ejecucion: npx nodemon “nombre del archivo que ejecutamos”**

**(npx se usa en el caso que nodemon este instalado como dev dependencie, sino con nodemon + nombre del archivo basta)**

**AHORA CUANDO EDITAMOS EL CODIGO Y LO GUARDAMOS AUTOMATICAMENTE LO REFRESCA EN CONSOLA. (SI NO FUNCIONA DE UNA VEZ PRESIONAR ENTER, O REESCRIBIR NPX NODEMON)**

**En json. tenemos dependencies y dev dependencies, la primera son los modulos que se necesitan para que funcione la aplicación, la segunda son solo los modulos que voy a usar yo durante el desarrollo de la app**

**HTTP METHODS (get, head, post, put, delete, connect, options)**

**Sirve para que el navegador pueda pedir determinados recursos al servidor y especificarle que quieres hacer con esos recursos.**

**Get** = **pide** algo **al servidor** para mostrar **(para devolver cosas en pantalla)**

**Post** = le estoy diciendo que el **navegador** le esta **enviando datos al servidor** para que los **almacene** y luego devolver una respuesta **(recibir determinado dato y guardarlo en una base de dato o procesarlo y luego devolverle una respuesta)**

**Put**= le digo desde el **navegador** que le **envio datos** para que **actualice** determinado dato en el **servidor (tomar datos que me da el front end y este me va a enviar datos que yo podre actualizar en una base de datos o aplicar determinada lógica y devolver algo al navegador)**

Toma los datos que me da el front end, cuando el front end haga la petición me va a enviar datos para poder actualizarlos en una base de datos o aplicar determinada lógica y luego devolver algo al navegador

**Delete=** es como **decirle al servidor que quiero que elimine** determinado dato (**y luego puede volver una respuesta)**

Tomo la petición que me da, un dato del servidor, y le doy una respuesta

**Asi funciona la gran mayoría de los servidores**

**Metodo POST :** los navegadores por defecto envían peticiones get entonces no podemos llamar al método post directo del navegador o nos dira cannot GET/ lo que sea

Para esto se utiliza un programa para testear este tipo de rutas llamado **POSTMAN (API)**

**Buscamos la ruta de enlace en la aplicación postman con el nombre del método y nos va a mostrar si funciona.**

**POSTMAN (App) (Simula un navegador o una aplicacion móvil. El lado del usuario)**

**Ejecucion basica:**

1. Seleccionamos la accion en el **dropdown** (GET, POST, ETC)
2. Tipeamos la URL de nuestro servidor (localhost:3000/about)
3. Damos Send
4. Podemos asi ver contenido de un POST, DELETE que normalmente el navegador no muestra (/ cannot GET)

**Pido desde un navegador a una ruta /user (del usuario al servidor) y este me responde con los datos en un json**

**Y lo muestra en la ventana del navegador**

app.get('/user', (req,res) =>{

    res.json({

        username: "Federico",

        lastname: "Marchetta"

    });

});

**Si queremos usar un método POST**

**Podemos enviar datos al servidor a través de postman**

**Arriba de todo**

1. En postman 🡪 lista desplegable en “POST” 🡪 ruta del servidor (ej: localhost:3000/user 🡪 pestaña headers
2. **Ir al campo de texto “keys” y tipear “Content-Type” (**
3. **En el campo de texto “value” tipear “application/json**

**Esto sería como decirle**

**1-**la aplicación “cliente” (el navegador, la aplicación móvil) **🡪** En este caso **Postman**

**2-**quiere **enviarle** a **localhost:3001/user 🡪** a través del método **POST** 🡪 Un archivo **.JSON**

**Para especificar el .JSON (EN POSTMAN)**

1. Ir a la pestaña body
2. **presionar “raw”**
3. Escribir el objeto con sus propiedades
4. Doy send
5. Para **VER** Datos que me envía la aplicaciónn del front end **(POSTMAN EN ESTE SIMULACRO)**
6. **Por consola vamos a ver undefined 🡪** Agregamos línea de código para que express entienda objetos .JSON
7. **IMPORTAMOS ESTE MIDDLEWARE :**

app.use(express.json());

//le dice a express que entienda formato JSON cuando una app/cliente se lo envie

1. **Agregamos este console.log(req.body) a este metodo POST**

app.post('/user',(req,res)=>{

    console.log(req.body); //esta es la respuesta a la consola (req.body)

//aca mostramos el body que enviamos a traves de POSTMAN (Objeto .JSON raw)

    res.send('PETICION POST RECIBIDA')    //esta es la respuesta al navegador

});

1. **De esa manera estaremos viendo los objetos JSON que nos envían por consola**
2. **req.body** va a ser el objeto que veremos por consola y podemos manipularlo

**Puedo extraer, cambiar, borrar, agregar, guardar en una base de datos🡪 los datos y propiedades**

**Recibir datos a través de un parámetro de una url 🡪 rutas dinámicas /loqueyoquieramandar**

app.post('/user/:**usuario**',(req,res)=>{

    console.log(req.body);

//mostramos por consola el body que enviamos a traves de POSTMAN (Objeto .JSON)

    console.log(req.params);

//con esto pedimos la RUTA del **:usuario** (lo recibimos a traves de los parametros de la url)

localhost.3001/user/LOQUESEA

    res.send('TA ANDANDO EL POST')   ;

});

**Req.body 🡪** toma la información que la aplicación o cliente nos manda

**Req.params 🡪** le decimos que vamos a aceptar un parámetro con los dos puntos 🡪 **/:loquesea**

**Estos dos request los podemos aplicar a cualquier ruta, incluso otros métodos (get, put, delete) también podemos guardarlos en variables**

**Delete**

app.delete('/user/:usuario',(req,res)=>{

    res.send(`User ${req.params.usuario} deleted`); //backticks `` 🡪 alt gr + }

//requiero tal (/:usuario) usuario para eliminarlo

    console.log("ese usuario " + req.params.usuario + " ia lo he borrao");

//muestro el nombre del usuario que estoy borrando por consola

});

**Put** (Para actualizar datos a través de un body)

1. Le damos un parámetro 🡪 **:/usuario** y
2. Y un **req.body** al igual que en los posts
3. Y su respuesta 🡪 res.send(“usuario“ + req.params.usuario **+ “ updated”**

Necesitamos saber que usuario queremos actualizar (/:usuario 🡪 req.params) y los datos que necesitamos reemplazar (req.body 🡪 a través del body de postman (o la aplicación que lo envie)

app.put('/user/:usuario',(req,res)=>{ //usuario que nos envía los datos al servidor

    console.log(req.body);

//datos nos envían al servidor desde la app (.JSON)

    res.send(`USUARIO ${req.params.usuario} ACTUALIZADO`);

//mensaje que devolvemos como servidor a la aplicación

});

**app.all**

Cada vez que utilice una ruta /user (o cualquiera que le digamos) va a pasar por este método app.all que podemos escribirle la función que queramos, terminando de una vez con **res.send** o continuando con el método **next()**

app.all('/user',(req,res)=>{ //todas las rutas user primero van a pasar por esta ruta

    console.log('por aqui paso'); //esto se lo digo a mi consola

    res.send('finish'); //y luego respondo al navegador terminando la ruta

});

app.all('/user',(req,res,next)=>{ //agregando next aca

    console.log('por aqui paso'); //cualquier método (post,put,etc) que ejecute la ruta user

    next(); **//y next aca continuo con el siguiente metodo app.get/post/etc…**

});

**Para conocer mucho más sobre las rutas y como funcionan en detalle los enrutadores**

[**www.expressjs.com**](http://www.expressjs.com) **🡪 Basic Routing**

**Guides 🡪 Routing (mas detallado)**

**API Reference 🡪 Applications (a la izquierda) 🡪 (todos los métodos que podemos utilizar APP.ALL – APP.DELETE)**

**MIDDLEWARES 🡪 app.use()**

**si o si pasa por esta ruta antes de continuar con el siguiente método (si es que permite seguir con next();)**

**Debe ir antes en el código que las rutas que afecta para que funcione**

Procesan datos antes que lleguen a las rutas

**app.all() 🡪 funcionaba para una ruta especifica 🡪 ej /user**

**app.use() 🡪 funciona para todas las rutas**

app.use(express.json());

* Este código es para que la app pueda entender formatos json
* Esta función recibe el objeto **req.body** y lo procesa para convertirlo en .json y una vez convertido continua con el resto y llegaba a su ruta final

**La función logger va a funcionar antes que todo, y luego recién pasar a la ruta siguiente**

**Esto es útil si queremos validar un usuario y no permitir pasar a la siguiente ruta sin primero pasar por esta entrada**

**En este caso va a mostrar la URL que el usuario esta pidiendo**

function logger(req, res, next){  //este middleware existe y se llama morgan npm

        console.log()

        console.log(`Route received ${req.protocol}://${req.get('host')}${req.originalUrl}`)

        next();

    }

app.use(logger);

**Req.protocol** 🡪 **http**

**req.get(‘host’)** 🡪 **localhost:3001**

**req.originalUrl** 🡪 ruta del usuario **(/user)**

Devuelve por consola

Route received http://localhost:3001/user

Si en el navegador buscamos cualquier ruta **🡪 ej localhost:3001/culo**

**nos va a avisar si no existiera pero la consola va a registrar de todas formas que se busco esta ruta en el servidor**

**NAVEGADOR 🡪** Cannot GET /asdasdasd

**CONSOLA 🡪 Route received** [**http://localhost:3001/asdasdasd**](http://localhost:3001/asdasdasd)

**Esto quiere decir que paso por el middleware y este lo proceso ya sea una ruta conocida o desconocida**

**Es lo mismo que hacia express.json antes de que llegue a cualquier ruta**

**NPM MORGAN (MIDDLEWARE QUE MUESTRA POR CONSOLA LAS PETICIONES QUE VAN LLEGANDO)**

1. Podemos solicitar el middleware que es igual a **logger** llamado **Morgan**

const morgan = require('morgan');

app.use(morgan('')); //Aparece la lista de formatos Morgan (el mas típico dev)

1. **Nos devuelve datos**

**(METODO – RUTA – COD DE RESPUESTA – TIEMPO DE RESPUESTA - PESO EN BYTES )**

GET / 404 22.668 ms - 139

GET /user 304 4.741 ms - -

POST /user/machiruli 200 7.785 ms – 18

**Existen middlewares de autenticacion, para cambiar datos, para subir imagines**

**(dependiendo de la app que quiera desarrollar)**

**Static Files** (**Se encarga de mandar archivos HTML, CCS y JS al FRONT )**

(Se llaman archivos estáticos porque no cambian, una vez que lo hemos creado lo único que hace el servidor es devolverlo)

El siguiente código lo agregamos al final de todo, antes del **app.listen(3001, () => {} )**

Para que se ejecute recién al terminar el resto, si ve que ninguna ruta (ej: /user ) fue utilizada llega a esta ultima ruta.

1. //este middleware ya viene con express, igual que express.json

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

//public = el nombre de la carpeta que creamos dentro del workspace

1. Creamos una carpeta llamada **public (la carpeta que va a ser accedida desde el navegador)**
2. Creamos un archivo nuevo dentro de **public 🡪 index.html 🡪**
3. Escribimos algún texto para mostrar en el **<body>** 🡪 **se mostrara en la pagina principal de este servidor** como cuando trabajamos sin un servidor
4. **Puedo hacer lo mismo con archivos CSS Y JS (LINKEANDOLOS A HTML)**
5. Si ejecutaramos localhost:3001/app.js 🡪 se mostraría el código de JS (el código de front end siempre es visible)

**CONFIGURACIONES EXPRESS (Settings)**

**Arriba de los middleware**

**Simplemente para mostrar cuando se ejecuta el servidor**

//express settings

app.set('appName', 'Curso express 2' )

// variable appName = ‘curso express 2' --> lo llamamos desde app.listen

app.listen(3001, ()=>{

    console.log(app.get('appName')) 🡨 AQUI

    console.log("Servidor en puerto 3001")

})

**Existen variables reservadas como ‘port’**

**Configurar el puertos**

**mas fácil cambiar el puerto desde aca**

app.set('port', 5000);

**que bajar a editar esto**

app.listen(app.get('port'), ()=>{

    console.log(app.get('appName'))

    console.log("Servidor en puerto", app.get('port') )

    //con esto evito tener que venir a cambiar mi puerto si es una app grande

})

**MOTOR DE PLANTILLAS (EJS)**

Nos ayuda a extender el HTML 🡪 **Si queremos consultar una base de datos y mostrar esos datos por pantalla**

1. **Instalación** 🡪 npm i ejs
2. **Lo configuramos de esta manera**

app.set('view engine', 'ejs');

1. **Creamos una carpeta llamada views (** de la misma forma que public )
2. **Dentro creamos 🡪 index.ejs 🡪 creamos estructura html5**
3. **Para mostrar esto por pantalla 🡪 En el índice del servidor 🡪**

app.get('/',(req,res)=>{ //cuando visiten mi ruta inicial

    res.render('index.ejs'); //que renderice la vista index.ejs

})

**Diferencia entre EJS y HTML 🡪**

**EJS 🡪 SON DATOS DINAMICOS, CAMBIANDINAMICAMENTE Y SE ACTUALIZAN EN PANTALLA**

**En el código del servidor**

**(EN DATA PONER BIEN CORCHETES PORQUE CON PARENTESIS TIRA ERROR)**

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

    const data = [{name: "Jose"},{name: "Fede"},{name: "Rubio"}]

//datos que recibo de una BASE DE DATOS

    res.render('index.ejs', {people: data});

 //people sera una variable que tendra esos datos, que llamaremos desde index.ejs

})

**En index.ejs**

**Este código se ve raro pero es simple, podemos saber mas googleando en npm ejs**

    <ul>

        <% people.forEach(function (people) { %>

            <li>

                <%= people.name %>

            </li>

        <% }) %>

    </ul>

**Se muestra en el inicio de mi servidor la lista de people.name**

**EXISTEN OTROS MOTORES DE PLANTILLA / TEMPLATE ENGINE Y CADA UNO TIENE SU SINTAXIS**

**-PUG**

**-HANDLEBARS (REDES SOCIALES)**

**LA FORMA MAS FACIL DE LOGRAR INFORMACION DE UN MODULO ES YENDO A SU PAGINA DE NPM**

**INTEGRACION CON BASES DE DATOS (EXPRESS ES AGNOSTICO A BASE DE DATOS)**

**Express database integration (no te dice que base de datos utilizar, uno decide)**

**Por la consola del proyecto puedo ingersar npm install mongodb y empezar a guardar mis datos alli**

**Puedo llamarlas con modulos npm integrándolas en express**

**ORACLE SQL**

ORM (EJECUTAR CODIGO PARA UTILIZAR LA BASE DE DATOS? )

EVITA ESCRIBIR SINTAXIS NATIVA DE LA BASE DE DATOS YA QUE TIENE SU PROPIO LENGUAJE

EL ORM HACE LAS CONSULTAS POR MI, YO SOLO EJECUTO LA SINTAXIS

MONGOOSE

EXISTE ALGO COMO EXPRESS LLAMADO

**ADONIS 🡪 QUE TE DA UNA CONFIGURACION Y TE GENERA UN PROYECTO ESTABLECIDO, UNA FORMA DE TRABAJAR, TE DICE COMO TENES QUE DESARROLLAR TU SERVIDOR 🡪 PARA INVESTIGAR MAS ADELANTE**

**SALIS.JS** 🡪 **TAMBIEN TE DA DE MANERA PRE ESTABLECIDA PARA QUE GENERES TUS RUTAS Y TUS APIS, ES BASTANTE SIMILAR A LARAVEL O DJANGO**

**HAPI.JS 🡪 DICEN QUE ES TAN BUENO O MEJOR QUE EXPRESS**

**KRAKEN.JS 🡪 ES UN FRAMEWORK QUE ESTA UTILIZADO EN PRODUCCION CREADO POR PAYPAL (NO TAN POPULAR PERO BUENO**

VAMOS A APLICAR UN POCO EXPRESS Y NODE.JS EN EL PROYECTO DE LAS REMERAS

O VAMOS A VER EL CURSO DEL CRUD DE NODE.JS

RELEER UN POCO TODO LO VISTO PREVIAMENTE, SOBRE TODO LO ULTIMO DE APP.ALL Y LA VUELTA QUE HACE POST CON EL :ID EL REQ.PARAMS Y LOS DETALLES QUE MUESTRA EN EL NAVEGADOR Y EL PUT COMO ACTUALIZA releer middlewares, settings, static files, dinamic files (ejs/motor de pantalla) , integración con base de datos (MONGOOSE)