Programación Web 2
Framework Express

U-TAD

Introducción de Express

Express es el framework web más popular de Node.js

https://www.npmjs.com/package/express

Casi 30M de descargas semanales!

Instalación: npm install express

Uso, en index.js:

```
const express = require('express')

const app = express()

app.get('/', function(req, res) {

  res.send('Hello World')

})

app.listen(3000)
```

Después lo instalamos e implementamos…

Introducción de Express

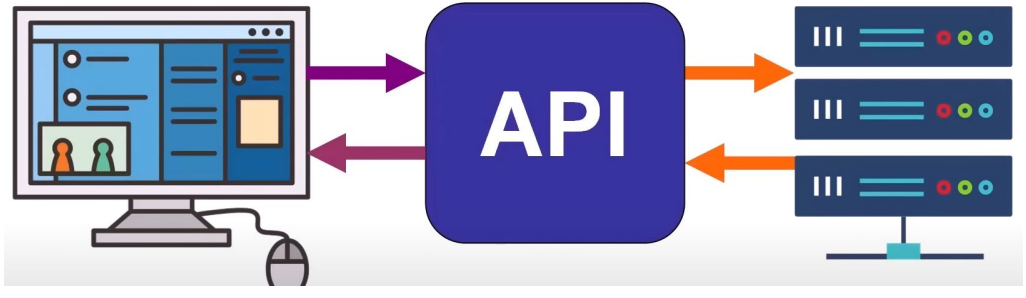
Nos ayuda a desarrollar aplicaciones de Node.js

Características:

- Routing
- Enfocado en alto rendimiento
- Nos permite desarrollar aplicaciones de Node.js más rápidamente con un código más claro.

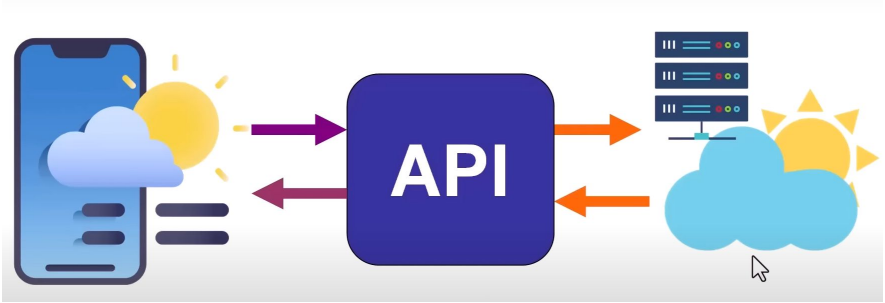
Conceptos básicos de repaso:

- CRUD: Operaciones básicas que se pueden realizar con los datos almacenados.
  - Create, Read, Update, Delete -> POST, GET, PUT, DELETE respectivamente.
- API (Application Programming Interface):
  - Tipo de interfaz de software que permite que dos o más programas se comuniquen entre sí.
  - Permite que un software ofrezca un servicio a otro software.
  - No es usada directamente por el usuario sino por el programador que usa la API para implementar su programa.



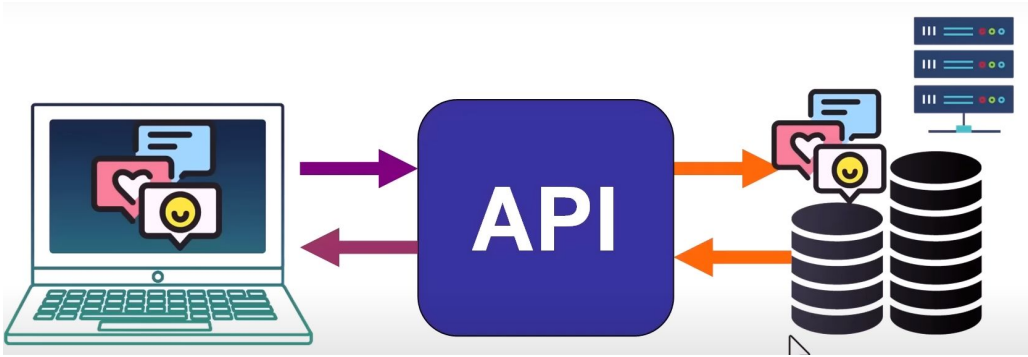
Introducción a Express

Ejemplos de casos de uso de un API:



Información del tiempo (o también podría ser información de mapas).

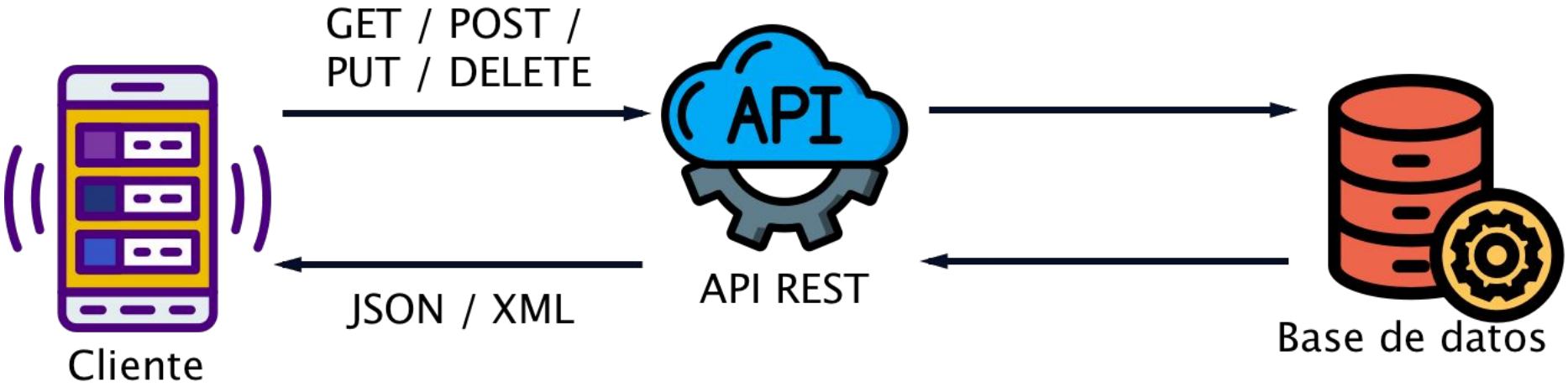
Redes sociales que proveen datos de los usuarios (posts, mensajes) vía API



REST

REST (Representational State Transfer): Estilo de arquitectura de software para sistemas hipermedia distribuidos como la World Wide Web.

RESTful API: Es una API basada en REST.



Crear un proyecto con Express

mkdir express && cd express

npm init –yes

npm install –save-dev nodemon

(añadir *“start”: “nodemon index.js”* en scripts de package.json)

npm install dotenv

(define tus variables de entorno en el fichero .env con PORT=3000, y añade .env a .gitignore cuando lo subas a Github)

npm install express

(crea el fichero index.js)

index.js con Express

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

//Simulamos una base de datos con el archivo de cursos.js anterior
const { infoCursos } = require('./cursos.js');

// Loading process.env
require('dotenv').config();

// Routing
app.get('/', (req, res) => {
  res.send('Hello World')
})

// Listening
const port = process.env.PORT || 3000;
app.listen(port, () => {
  console.log('Servidor iniciado en el puerto' , port);
});
```

Routing con Express

Devolver cursos, cursos.programacion y cursos.matematicas en texto (String)

Ejemplo:

```
app.get('/api/cursos/programacion', (req, res) => {
    //res.send(infoCursos.programacion); //devuelve un objeto de JS
    res.send(JSON.stringify(infoCursos.programacion); // devuelve un String
}
```

Nota: Si intentamos acceder a un recurso que no esté definido, automáticamente Express nos devuelve un 404 NOT FOUND.

Parámetros de url

```
app.get('/api/cursos/programacion/:lenguaje', (req, res) => {
    const lenguaje = req.params.lenguaje;
    const data = infoCursos.programacion.filter(curso => curso.lenguaje === lenguaje);
    if(data.length === 0) {
        return res.status(404).send("No se encontró" + lenguaje);
    }
    res.send(JSON.stringify(data));
});
```

Ejercicios: 1) Lo mismo para matematicas.tema y 2) con programacion.lenguaje y nivel

Parámetros query

En index.js, dentro de cualquier función *app.get()*

```
if (req.query.ordenar === 'vistas') {
    //Orden DESC, si lo queremos ASC, sería (a.vistas, b.vistas)
    res.send(JSON.stringify(data.sort((a, b) => b.vistas - a.vistas )));
} else {
    res.send(JSON.stringify(data));
  }
```

En index.http, la llamada sería

```
GET http://localhost:3000/api/cursos/programacion/python?ordenar=vistas HTTP/1.1
```

Routers en Express

```
const routerProgramacion = express.Router();
app.use('/api/cursos/programacion', routerProgramacion);

//app.get('/api/cursos/programacion', (req, res) => {...})
routerProgramacion.get('/', (req, res) => {...})
```

y

```
//app.get('/api/cursos/programacion/:lenguaje', (req, res) => {...})
routerProgramacion.get('/:lenguaje', (req, res) => {...})
```

Haz lo mismo con matemáticas.

Routers en distintos archivos

Dentro de nuestro proyecto express, creamos un nuevo directorio “routers”

Con dos ficheros:

- programacion.js
- matematicas.js

Nos llevamos todas las rutas relacionadas con programación y matemáticas a sus respectivos .js

Creamos el directorio “datos” y movemos *cursos.js* ahí

POST, PUT, PATH y DELETE

programacion.js

```
//Middleware
routerProgramacion.use(express.json());

routerProgramacion.post('/', (req, res) => {
    const cursoNuevo = req.body;
    //Aquí irían algunas comprobaciones de formato
    programacion.push(cursoNuevo);
    res.send(JSON.stringify(programacion));
});
```

index.http

```
POST http://localhost:3000/api/cursos/programacion  HTTP/1.1
Content-Type:  application/json

{
    "id": 4,
    "titulo": "Aprende Node.js",
    "lenguaje": "javascript",
    "vistas": 45676,
    "nivel": "basico"
}
```

Las funciones **middleware** se ejecutan:

- Después de recibir una solicitud.
- Antes de enviar una respuesta.

Tienen acceso al objeto de la solicitud, al objeto de la respuesta y a next(), una función que se llama para ejecutar el próximo middleware.

POST, PUT, PATH y DELETE

programacion.js

```
routerProgramacion.put('/:id', (req, res) => {
    const cursoActualizado = req.body;
    const id = req.params.id;

    const indice = programacion.findIndex(curso => curso.id == id);
    // Si no lo encuentra, devuelve -1
    if (indice >= 0) {
        programacion[indice] = cursoActualizado;
    }
    res.send(JSON.stringify(programacion));
});
```

index.http

```
PUT http://localhost:3000/api/cursos/programacion/2  HTTP/1.1
Content-Type: application/json

{
    "id": 2,
    "titulo": "Python intermedio con proyectos",
    "lenguaje": "python",
    "vistas": 123996,
    "nivel": "intermedio"
}
```

POST, PUT, PATH y DELETE

programacion.js

```
routerProgramacion.patch('/:id', (req, res) => {
    const infoActualizada = req.body;
    const id = req.params.id;
    const indice = programacion.findIndex(curso => curso.id == id);
    if (indice >= 0) {
        const cursoAModificar = programacion[indice];
        //Modifica solo algunas propiedades del objeto
        Object.assign(cursoAModificar, infoActualizada);
    }
    res.send(JSON.stringify(programacion));
});
```

index.http

```
PATCH http://localhost:3000/api/cursos/programacion/2  HTTP/1.1
Content-Type: application/json

{
    "titulo": "Python intermedio con proyectos",
    "vistas": 223996
}
```

POST, PUT, PATH y DELETE

programacion.js

```
routerProgramacion.delete('/:id', (req, res) => {
    const id = req.params.id;
    const indice = programacion.findIndex(curso => curso.id == id);
    if (indice >= 0) {
        //Elementos a eliminar desde el índice
        programacion.splice(indice, 1);
    }
    res.send(JSON.stringify(programacion));
});
```

index.http

```
DELETE http://localhost:3000/api/cursos/programacion/1  HTTP/1.1
```

EJERCICIO

- Modifica el código de routerMatematicas para que acepte POST, PUT, PATCH y DELETE
- Modifica el código de routerProgramacion y routerMatematicas para que devuelve el código de error 404 en caso de no encontrar la solicitud. Y en ese caso enviar .end() sin parámetros en vez de .send(“texto”)
- Cambiar el método .send(), que puede enviar cualquier cosa, por el método .json() pasando el objeto simplemente, ya que él se encarga de pasarlo siempre a JSON.
- Investiga sobre el código 204 Non Content y, donde aplique, cámbialo por el 404 Not Found.