

Programación IV

Profesora Cynthia Estrada

Fecha: 09/09

"Express"

Integrantes:

- "Díaz Rossini Juan José",
- "Gallo Genaro"
- "Navarro Victor Leandro"

-
1. ¿Por qué utilizar Express?
 2. Diferencias entre Node y Express, realice un cuadro de los mismos.
 3. ¿Express es un framework backend o frontend?
 4. Explique la estructura de una aplicación Express
 5. ¿Existen algunas alternativas populares a Express?
 6. ¿Qué herramientas se pueden integrar con Express?
 7. ¿Para qué se utiliza el archivo .env?
-

1. ¿Por qué utilizar Express?

Express se ha convertido en el framework más popular de Node.js porque soluciona los problemas que aparecen al trabajar solo con Node puro.

Algunas razones clave para usarlo son:

- **Rapidez en el desarrollo:** Express permite levantar un servidor con pocas líneas de código, mientras que con Node.js puro habría que escribir funciones más largas y configuraciones manuales.
- **Middlewares:** ofrece un sistema modular donde se pueden integrar funciones intermedias (ej. autenticación, validación de datos, logs, seguridad, etc.) sin alterar directamente la lógica principal.
- **Gestión de rutas:** Express facilita crear rutas dinámicas y organizarlas en módulos, evitando mezclar toda la lógica en un mismo archivo.
- **Escalabilidad:** su estructura modular permite crecer de proyectos pequeños a grandes sistemas de producción.
- **Compatibilidad:** se integra fácilmente con bases de datos (MongoDB, MySQL, PostgreSQL), motores de plantillas (EJS, Handlebars, Pug), librerías de seguridad y más.
- **Comunidad y soporte:** al ser muy popular, tiene abundante documentación, foros, ejemplos y paquetes compatibles.

Express es elegido porque es simple, flexible y extensible, lo que reduce el tiempo de desarrollo y evita reinventar la rueda.

2. Diferencias entre Node y Express (cuadro comparativo)

Aspecto	Node.js	Express.js
Definición	Entorno de ejecución que permite correr JavaScript en el servidor.	Framework minimalista construido sobre Node.js para crear aplicaciones web y APIs.

Nivel de abstracción	Bajo, cercano al sistema: hay que trabajar con objetos HTTP nativos.	Medio: abstrae lo complejo y lo simplifica con funciones listas para usar.
Manejo de rutas	Debe hacerse manualmente, escuchando <code>req.url</code> y evaluando condiciones.	Ofrece un sistema robusto de enrutamiento (<code>app.get()</code> , <code>app.post()</code> , etc.).
Uso principal	Servidores, herramientas CLI, backend en general.	Aplicaciones web, APIs REST/GraphQL y microservicios.
Complejidad	Verboso, más difícil para proyectos grandes.	Más sencillito, rápido de implementar y mantener.
Flexibilidad	Permite cualquier tipo de aplicación desde cero.	Especializado en aplicaciones web, aunque sigue siendo flexible.
Productividad	Menor, ya que requiere más código repetitivo.	Alta, gracias a sus utilidades y middlewares.

3. ¿Express es un framework backend o frontend?

Express es un framework backend. Se ejecuta en el servidor y su función principal es manejar la lógica de negocio y comunicación cliente-servidor. Entre sus responsabilidades están:

- Recibir peticiones HTTP del cliente.
- Procesar la información y ejecutar la lógica correspondiente (consultar una base de datos, aplicar validaciones, ejecutar reglas de negocio).
- Enviar respuestas HTTP al cliente, que pueden ser datos en JSON, archivos, páginas renderizadas, etc.
- Coordinar la seguridad (ej. JWT, sesiones, cookies).

No se usa en el frontend, pero puede proveer datos o plantillas a frameworks como **React**, **Angular** o **Vue**.

4. Estructura de una aplicación Express

La estructura depende del tamaño del proyecto, pero una forma organizada y escalable suele ser así:

```

/mi-proyecto
├── node_modules/      -> Dependencias instaladas
├── src/
│   ├── routes/        -> Definición de rutas (ej: userRoutes.js, productRoutes.js)
│   ├── controllers/   -> Lógica de cada ruta (ej: funciones para usuarios, productos)
│   ├── models/        -> Modelos de datos y conexión a BD
│   ├── middlewares/   -> Funciones intermedias (auth, validaciones, manejo de errores)
│   ├── config/        -> Configuración (ej: conexión a DB, variables globales)
│   └── app.js         -> Configuración principal de Express (middlewares y rutas globales)
├── .env               -> Variables de entorno (puertos, DB_URL, claves secretas)
├── package.json       -> Dependencias y scripts del proyecto
└── server.js          -> Punto de inicio del servidor (importa app.js y lo ejecuta)

```

Esta separación permite que la aplicación sea más mantenible, legible y fácil de escalar.

5. ¿Existen algunas alternativas populares a Express?

Sí, aunque Express sigue dominando, existen frameworks alternativos que cubren distintas necesidades:

- **Koa.js** → Creado por los mismos autores de Express. Es más moderno, modular y ligero, ideal para aplicaciones pequeñas.
- **Fastify** → Optimizado para alto rendimiento. Muy usado en APIs rápidas con baja latencia.
- **NestJS** → Framework basado en TypeScript y arquitectura modular. Inspirado en Angular, ideal para proyectos grandes y bien estructurados.
- **Hapi.js** → Pensado para seguridad y validación, con configuración más estricta que Express.
- **Sails.js** → Inspirado en Ruby on Rails, basado en el patrón MVC, muy usado en aplicaciones grandes y en tiempo real.

6. ¿Qué herramientas se pueden integrar con Express?

Express, al ser muy flexible, se integra con múltiples herramientas y librerías, entre ellas:

Bases de datos:

- MongoDB (con Mongoose).
- MySQL / PostgreSQL (con Sequelize, Prisma, Knex.js).

Seguridad:

- Helmet (protección de cabeceras).
- CORS (control de orígenes cruzados).
- JWT (autenticación con tokens).

Gestión de sesiones:

- express-session
- cookie-parser
- Passport.js.

Motores de plantillas:

- EJS
- Pug
- Handlebars.

Validación:

- Joi
- express-validator
- Yup.

Testing:

- Mocha
- Chai
- Jest
- Supertest.

Documentación de APIs:

- Swagger
- Redoc
- Postman.

Otros middlewares útiles:

- Morgan (logs de peticiones).
- Multer (subida de archivos).
- Socket.IO (comunicación en tiempo real).

En conjunto, estas integraciones hacen que Express sea muy versátil para cualquier tipo de aplicación backend.

7. ¿Para qué se utiliza el archivo .env?

El archivo .env sirve para manejar variables de entorno que configuran la aplicación de forma externa, sin tener que modificar el código.

Se usa para almacenar información sensible o dependiente del entorno, como:

- Puertos: `PORT=3000`
- Credenciales de base de datos: `DB_USER=root / DB_PASS=1234`
- Claves secretas: `JWT_SECRET=miClaveSecreta`
- URLs de servicios externos: `API_KEY_GOOGLE=XXXX`

Ventajas:

1. **Seguridad:** las claves no quedan expuestas en el código.
2. **Portabilidad:** se pueden usar diferentes configuraciones en desarrollo, testing y producción.
3. **Estandarización:** es compatible con múltiples librerías (dotenv) que cargan estas variables automáticamente.

Ejemplo típico de .env:

```
ini

PORT=4000
DB_URL=mongodb://localhost:27017/miapp
JWT_SECRET=claveSuperSecreta
```

Y en tu código, con **dotenv**:

```
js

require('dotenv').config();
const PORT = process.env.PORT || 3000;
```