COLOMBIA POTENCIA DE LA VIDA

TIC

TALENTO TECH

Bootcamp

Desarrollo Web Full Stack

Nivel Básico

David Ricardo Rivera Arbeláez

Agosto 21/2024 - Septiembre 3/2024

UT TALENTOTECH

cymetria | tecnalia colombia

TIC

Módulo 5
Backend con NodeJS

- Introducción a línea de comandos.
- Ejecución de Scripts con NodeJS
- Instalación de librerías y paquetes con NPM.
- Creación de API con Expressjs
- Conexión a Bases de Datos con Express
- Lectura y guardado de información con Express y Bases de Datos.

Introducción a la línea de comandos

¿Qué es la terminal?

Es una interfaz gráfica que simula una línea de comandos y cuando hablamos de una línea de comandos nos referimos a una shell.

Introducción a la línea de comandos

¿Por qué aprender a usarla?

- Flexibilidad.
- Velocidad.
- No siempre tendremos una interfaz gráfica

Introducción a la línea de comandos

¿Qué es un comando?

- Programa ejecutable.
- Una instrucción de utilidad de la Shell.
- Una función de Shell.
- Un alias

Introducción a la línea de comandos

Tipos de líneas de comando

- Bourne Shell
- **Bash Shell**
- **Z Shell**
- C Shell
- Korn Shell
- Fish Shell
- **PowerShell**

Introducción a la línea de comandos

Primeros comandos

Comando	Acción
ls	Lista los archivos y carpetas del directorio.
ls –l	Lista los archivos y carpetas con toda la información de cada uno.
ls –lh	Lista los archivos y carpetas con la información legible para humanos.
cd	Mueve la terminal al directorio home del usuario.
cd {folder}	Mueve la terminal al directorio indicado.
clear	Limpia la pantalla de la terminal (shortcut: cmd + L).
pwd	Imprime la ruta actual en la que nos encontramos en la terminal.
file {name_file}	Describe el tipo de archivo que le pasamos como parámetro.

Introducción a la línea de comandos

Primeros comandos

Comando	Acción
ls –la	Lista todos los elementos del directorio, incluidos los ocultos.
ls –lS	Lista todos los elementos iniciando por los más pesados.
ls –lr	Lista todos los elementos de forma inversa.
tree	Despliega todos nuestros directorios como un árbol.
tree –L {#}	Despliega los elementos que se encuentren en el nivel indicado.

Introducción a la línea de comandos

Manipulación de elementos

Comando	Acción
mkdir {folder}	Crea un nuevo directorio con el nombre indicado.
touch {file}	Crea un nuevo archivo con el nombre indicado.
cp {original} {copia}	Copia un archivo.
mv {file} {path}	Mueve el archivo a la ubicación deseada.
mv {name} {new_name}	Renombra el archivo o directorio.
rm {file}	Elimina el archivo indicado.
rm -i {file}	Pide confirmación antes de eliminar el archivo.
rm -r {folder}	Elimina el directorio indicado.

Introducción a la línea de comandos

Explorando contenido de los archivos

Comando	Acción
head {file}	Muestra las primeras 10 líneas de un archivo de texto.
head {file} -n {#}	Muestra las primeras líneas de código indicadas.
tail {file}	Muestra las últimas 10 líneas de un archivo de texto.
tail {file} -n {#}	Muestra las últimas de líneas de código indicadas.
less {file}	Muestra todo el archivo de texto seleccionado.
open	Abre un archivo desde la terminal (MacOs).
xdg-open	Abre un archivo desde la terminal (Linux).
nautilus	Abrir el sistema de archivos (Linux)

Introducción a la línea de comandos

Comandos útiles

Comando	Acción
help {comando}	Muestra información sobre cómo usar un comando (BASH).
man {comando}	Muestra el manual del comando indicado.
info {comando}	Muestra el manual del comando indicado.
whatis {comando}	Muestra una descripción corta del comando indicado.

Introducción a la línea de comandos

Rutas relativas

- . → Ubica en el directorio actual.
- .. → Regresa a un directorio.

NodeJS

Es un entorno de ejecución de JavaScript que permite ejecutar código JavaScript fuera de un navegador web.

Ejecución de Scripts con NodeJS

Rutas relativas

- . → Ubica en el directorio actual.
- .. → Regresa a un directorio.

NodeJS

Características

- Centrado en módulos.
- Orientado a eventos.
- Monohilos con entradas y salidas asíncronas.
- Motor V8

NodeJS

¿Por qué usarlo?

- Crear servidores web.
- Automatizar tareas.
- Desarrollar aplicaciones de línea de comandos.
- Y muchas más...

Instalación de NodeJS

1. Visitar y descargar Node.js adecuado al sistema operativo.
2. Seguir los pasos del instalador.
3. Abrir una terminal y ejecutar.

```
node -v
npm -v
```

Ejecución de scripts con NodeJS

1. **Crear un archivo.**
2. **Escribir código.**

```
JS   console.log("Hola Node.js");
```

3. Ejecutar el script

```
Bash  node mi_primer_script.js
```

Proceso de ejecución de scripts con NodeJS

- Motor V8.
- Ciclo de eventos.
- Módulos.

Módulos y NPM

- **Módulos:** Bloques de código reutilizable que se pueden importar en otros archivos.
- **NPM:** Es el gestor de paquetes de Node.js. Se utiliza para instalar y gestionar módulos de terceros.

Instalación de módulos con NPM

1. Listar todos los paquetes instalados localmente:
•npm list:

2. Listar paquetes globales instalados:
•npm list -g:

3. Listar paquetes instalados con más detalle:
•npm list --depth=0:

NPM

- npm list: Lista todos los paquetes locales.
- npm list -g: Lista todos los paquetes globales.
- npm list --depth=0: Lista solo los paquetes de nivel superior.
- npm list --long: Lista todos los paquetes con información detallada.
- npm outdated: Muestra los paquetes desactualizados.
- npm uninstall <paquete>: Desinstala un paquete.

Instalación de módulos con NPM

npm install NOMBRE DEL MODULO

Instalación de módulos con NPM

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('¡Hola desde mi servidor Node.js!');
});

server.listen(3000, () => {
    console.log('Servidor escuchando en el puerto 3000');
});
```

Creación de API con Expressjs

API RESTful: es un conjunto de reglas y convenciones para diseñar interfaces de programación de aplicaciones (API) que permiten que diferentes sistemas de software se comuniquen entre sí.

¿Por qué usar Expressjs para crear API?

- Sencillo y eficiente.
- Rendimiento.
- Flexibilidad.
- Middleware potente.
- Amplia comunidad.

Ejemplo práctico Expressjs: API de tareas

- Estructura del proyecto
  - Task-API
    - package.json
    - index.js
    - models
      - Task.js
    - Routes
      - tasks.js
    - config

    - .env

Ejemplo práctico Expressjs: API de tareas

- Instalación de dependencias
  - npm init –y
  - npm install express mongoose dotenv cors body-parser

- Archivo .env (Para variables de entorno):
  - MONGODB_URI=tu_cadena_de_conexion_a_mongodb

Ejemplo práctico Expressjs: API de tareas

- Archivo config/database.js:

```
const mongoose = require('mongoose');
require('dotenv').config();

mongoose.connect(process.env.MONGODB_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
.then(() => console.log('Conectado a MongoDB'))
.catch(err => console.error(err));
```

Ejemplo práctico Expressjs: API de tareas

- Archivo models/Task.js:

```
const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    description: String,
    completed: {
        type: Boolean,
        default: false
    }
});

module.exports = mongoose.model('Task', taskSchema);
```

Ejemplo práctico Expressjs: API de tareas

- Archivo routes/tasks.js:

```
const express = require('express');
const router = express.Router();
const Task = require('../models/Task');

// Obtener todas las tareas
router.get('/', async (req, res) => {
    const tasks = await Task.find();
    res.json(tasks);
});

// Crear una nueva tarea
router.post('/', async (req, res) => {
    const { title, description } = req.body;
    const newTask = new Task({ title, description });
    await newTask.save();
    res.json(newTask);
});
```

Ejemplo práctico Expressjs: API de tareas

- Archivo routes/tasks.js:

```
// Obtener una tarea por ID
router.get('/:id', async (req, res) => {
    const task = await Task.findById(req.params.id);
    if (!task) {
        return res.status(404).json({ message: 'Tarea no encontrada' });
    }
    res.json(task);
});

// Actualizar una tarea
router.put('/:id', async (req, res) => {
    const { title, description, completed } = req.body;
    const task = await Task.findByIdAndUpdate(req.params.id, { title, description, completed }, { new: true });
    res.json(task);
});
```

Ejemplo práctico Expressjs: API de tareas

- Archivo routes/tasks.js:

```
// Eliminar una tarea
router.delete('/:id', async (req, res) => {
    await Task.findByIdAndDelete(req.params.id);
    res.json({ message: 'Tarea eliminada' });
});

module.exports = router;
```

Ejemplo práctico Expressjs: API de tareas

- Archivo index.js:

```
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const tasksRouter = require('./routes/tasks');

const app = express();
const port = process.env.PORT || 3000;

app.use(cors());
app.use(bodyParser.json());

app.use('/api/tasks', tasksRouter);

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

Conexión a Bases de Datos con Express

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('¡Hola desde mi servidor Node.js!');
});

server.listen(3000, () => {
    console.log('Servidor escuchando en el puerto 3000');
});
```

Lectura y guardado de información con Express y Bases de Datos

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('¡Hola desde mi servidor Node.js!');
});
```

Buenas prácticas

- Estructura de proyectos.
  - Organización clara.
  - Convenciones de nombrado.
  - Linter
- Manejo de asincronía.
  - Callbacks, Promises o Async/Await.
  - Async/Await.
  - Control de errores.

Buenas prácticas

- Gestión de dependencias.
  - npm o yarn.
  - package.json.
  - Semver
- Pruebas.
  - Pruebas unitarias.
  - Pruebas de integración.
  - Cobertura de código.

Buenas prácticas

eguridad.
- Sanitización de datos.
- Cifrado.
- Actualizaciones de seguridad.

- Rendimiento.
  - Profiling.
  - Caching.
  - Clusterización.

Recursos de apoyo

[Documentación oficial de Node.js]

Tutoriales en línea
[Curso de NodeJS desde cero – MiduDev]

[Curso NodeJS desde cero – Sergie Code]

COLOMBIA POTENCIA DE LA VIDA

TALENTO TECH

TIC

¡Gracias!

UT TALENTOTECH

cymetria | tecnalia colombia