Clase 12: Material Complementario

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:24
Libro:	Clase 12: Material Complementario		

Tabla de contenidos

1. Introducción a Desarrollo Back-End

Bloques temáticos

1. Conceptos de desarrollo back-end y arquitectura cliente-servidor
2. Introducción a Node.js y su papel en el desarrollo back-end
3. Creación de un servidor web con Node.js y Express
4. Manejo de rutas, parámetros y datos de solicitud
5. Conexión a una base de datos (por ejemplo, MongoDB) utilizando Node.js
6. Implementación de autenticación y autorización en aplicaciones web
7. Seguridad y prevención de vulnerabilidades en el desarrollo back-end

1.1. Introducción a Desarrollo Back-End

1. Conceptos de desarrollo back-end y arquitectura cliente-servidor

El desarrollo back-end se enfoca en la parte de una aplicación web que los usuarios no ven. Es responsable de procesar solicitudes, administrar bases de datos y garantizar que todo funcione sin problemas. Algunas de las tareas comunes en el desarrollo back-end incluyen:

- Procesamiento de solicitudes HTTP.
- Autenticación y autorización de usuarios.
- Almacenamiento y recuperación de datos en bases de datos.
- Mantenimiento de la seguridad de la aplicación.
- Gestión de servidores y escalabilidad.

Arquitectura Cliente-Servidor

La mayoría de las aplicaciones web siguen una arquitectura cliente-servidor. Aquí, el cliente (generalmente un navegador web) y el servidor interactúan para proporcionar una experiencia de usuario. El cliente envía solicitudes al servidor, y el servidor procesa esas solicitudes y envía respuestas de vuelta al cliente.

Los conceptos clave de esta arquitectura incluyen:

- **Cliente:** El cliente es la interfaz de usuario con la que interactúa el usuario. Esto puede ser un navegador web, una aplicación móvil o cualquier otro dispositivo que accede a la aplicación.
- **Servidor:** El servidor es donde reside la lógica del back-end. Procesa solicitudes, accede a bases de datos y realiza cálculos antes de enviar una respuesta al cliente.
- **Comunicación:** La comunicación entre el cliente y el servidor se realiza a través de solicitudes HTTP (Hypertext Transfer Protocol) y respuestas. Las solicitudes pueden incluir la obtención de páginas web, el envío de datos del usuario y más.

Responsabilidades del Back-End

El desarrollo back-end asume una serie de responsabilidades cruciales para el funcionamiento de una aplicación web. Estas responsabilidades incluyen:

- **Gestión de Datos:** Almacenar, recuperar y gestionar datos en una base de datos. Esto puede incluir la creación, lectura, actualización y eliminación de datos.

· **Lógica de Negocio:** Implementar la lógica de negocio de la aplicación, lo que puede incluir cálculos, validaciones y decisiones empresariales.

· **Seguridad:** Garantizar la seguridad de la aplicación, protegiendo los datos y evitando amenazas como la inyección de SQL y los ataques de seguridad.

· **Escalabilidad:** Asegurarse de que la aplicación pueda manejar un alto volumen de tráfico y crecer a medida que más usuarios la utilizan.

2. Introducción a Node.js y su papel en el desarrollo back-end

En el mundo del desarrollo back-end, Node.js ha revolucionado la forma en que se construyen aplicaciones web. Es una plataforma de tiempo de ejecución de JavaScript basada en el motor V8 de Google Chrome que permite ejecutar código JavaScript en el servidor.

Node.js se enfoca en la lógica del servidor y la gestión de datos. Algunos conceptos clave incluyen:

· **Tiempo de ejecución:** Node.js proporciona un entorno de tiempo de ejecución que permite la ejecución de código JavaScript en el servidor.

· **Event Loop:** utiliza un bucle de eventos no bloqueante que permite manejar muchas solicitudes simultáneas sin bloquear el hilo principal de ejecución.

Ecosistema de Node.js y Sus Módulos

Node.js incluye el Administrador de Paquetes de Node (npm), que es una herramienta que permite a los desarrolladores instalar y gestionar paquetes de código reutilizable. El ecosistema de Node.js es rico y diverso, con miles de módulos y bibliotecas disponibles en el repositorio de npm. Estos módulos abarcan desde la creación de servidores web hasta el acceso a bases de datos y la autenticación de usuarios.

Ventajas de Utilizar Node.js en el Desarrollo Back-End

Node.js ha ganado popularidad en el desarrollo back-end por varias razones:

· **Lenguaje Unificado:** Utilizar JavaScript tanto en el cliente como en el servidor reduce la necesidad de aprender múltiples lenguajes de programación.

· **Rendimiento:** El modelo de E/S no bloqueante y el motor V8 de alto rendimiento de Node.js lo hacen rápido y eficiente.

· **Ecosistema y Módulos:** Node.js cuenta con un amplio ecosistema de módulos que facilita la incorporación de funcionalidades adicionales en una aplicación.

· **Comunidad Activa:** Una comunidad activa de desarrolladores contribuye constantemente a la mejora y la expansión de Node.js.

3. Creación de un servidor web con Node.js y Express

Express.js es un marco de aplicaciones web para Node.js que simplifica la creación de servidores y la gestión de rutas, solicitudes y respuestas. Algunas ventajas clave de Express incluyen:

· **Enrutamiento:** Express permite definir rutas y manejar solicitudes HTTP de manera clara y organizada.

· **Middleware:** Permite usar middleware para ejecutar funciones intermedias antes de procesar una solicitud.

· **Escalabilidad:** Express es altamente escalable, lo que lo hace adecuado para aplicaciones de cualquier tamaño.

Configurando el Entorno de Desarrollo Node.js

Antes de comenzar a construir el servidor web con Node.js y Express, es importante configurar un entorno de desarrollo. Esto incluye la instalación de Node.js y npm, el Administrador de Paquetes de Node, así como la creación de un directorio de proyecto. Con un entorno de desarrollo adecuado, estarás listo para comenzar a trabajar con Express.

Creando un Servidor Web Básico con Express

Para crear un servidor web básico con Express, necesitas instalar Express en tu proyecto mediante npm. Luego, puedes configurar una aplicación Express, definir rutas y manejar solicitudes HTTP.

A continuación, se muestra un ejemplo sencillo de cómo crear un servidor web básico:

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

app.get('/', (req, res) => {
  res.send('¡Hola, mundo!');
});
```

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

4. Manejo de rutas, parámetros y datos de solicitud

Implementando Rutas y Manejando Solicitudes

Express te permite definir rutas y manejar solicitudes de manera flexible. Puedes responder a diferentes verbos HTTP y crear rutas dinámicas que contengan parámetros. Además, puedes utilizar middleware para realizar acciones adicionales antes de manejar solicitudes específicas.

Definiendo Rutas en Express

Las rutas son el corazón de la gestión de solicitudes. Puedes definir rutas utilizando diferentes verbos HTTP (GET, POST, PUT, DELETE) y proporcionar una función controladora que se ejecuta cuando se accede a esa ruta. Por ejemplo:

```
app.get('/ruta', (req, res) => {
  // Manejar la solicitud GET a /ruta
});
```

Rutas Dinámicas y Parámetros

Express permite crear rutas dinámicas que contienen parámetros, lo que facilita la gestión de recursos variables en tu aplicación. Puedes definir rutas con dos puntos seguidos de un nombre de parámetro, como :id. Luego, puedes acceder a estos parámetros en tu función controladora. Por ejemplo:

```
app.get('/usuario/:id', (req, res) => {
  const userId = req.params.id;
  // Usar el valor de userId en la respuesta
```

```
});
```

Manejo de Datos de Solicitud

Cuando un cliente realiza una solicitud HTTP, envía datos al servidor en la forma de encabezados, parámetros de consulta y cuerpos de solicitud. Express facilita la obtención y análisis de estos datos utilizando objetos como req.query, req.headers y req.body. Por ejemplo:

```
app.post('/enviar-datos', (req, res) => {
  const datos = req.body;
  // Procesar los datos en la solicitud POST
});
```

Respondiendo a Solicitudes de Manera Efectiva

Además de manejar datos de solicitud, Express te permite responder a las solicitudes del cliente de manera efectiva. Puedes enviar diferentes tipos de respuestas, como HTML, JSON o archivos estáticos. Por ejemplo:

```
app.get('/json', (req, res) => {
  const data = { mensaje: 'Hola, mundo' };
  res.json(data);
});
```

El manejo de rutas, parámetros y datos de solicitud es una parte fundamental en el desarrollo de aplicaciones web con Express. Al comprender cómo definir rutas, utilizar rutas dinámicas, acceder a parámetros y analizar datos de solicitud, los desarrolladores pueden crear aplicaciones web altamente interactivas y dinámicas.

5. Conexión a una base de datos (por ejemplo, MongoDB) utilizando Node.js

Integración de MongoDB en una Aplicación Node.js

La integración de una base de datos es esencial para muchas aplicaciones back-end. MongoDB, una base de datos NoSQL ampliamente utilizada, se integra de manera eficiente con Node.js. Exploraremos cómo establecer una conexión a una base de datos MongoDB en una aplicación Node.js.

Importancia de la Integración de Bases de Datos

En muchas aplicaciones web, es fundamental almacenar y recuperar datos. Una base de datos actúa como el repositorio central para almacenar información crítica. Al integrar una base de datos en una aplicación Node.js, puedes realizar operaciones de lectura y escritura en la base de datos, lo que es esencial para la persistencia de datos y la interacción con el servidor.

MongoDB: Una Base de Datos NoSQL

MongoDB es una base de datos NoSQL que se caracteriza por su flexibilidad y escalabilidad. Algunas ventajas clave de MongoDB incluyen:

· **Modelo de Datos Flexible:** permite almacenar datos en documentos BSON (formato binario JSON), lo que facilita la representación de datos de manera flexible.

· **Escalabilidad Horizontal:** se puede escalar horizontalmente, lo que permite manejar grandes volúmenes de datos y tráfico.

· **Consultas Ricas en Funciones:** admite consultas ricas en funciones que permiten filtrar, ordenar y realizar operaciones avanzadas en los datos.

Configurando la Conexión a MongoDB en Node.js

Para comenzar a trabajar con MongoDB, es necesario instalar el controlador oficial de MongoDB para Node.js, llamado "mongodb". Luego, puedes configurar la conexión a la base de datos proporcionando la URL de conexión y opciones de configuración.

```
const { MongoClient } = require('mongodb');

const url = 'mongodb://localhost:27017'; // URL de conexión
const dbName = 'miBaseDeDatos'; // Nombre de la base de datos

const client = new MongoClient(url, { useUnifiedTopology: true });

client.connect()
  .then(() => {
    const db = client.db(dbName);
    // Realizar operaciones en la base de datos
  })
```

```
  .catch(err => console.error('Error de conexión a la base de datos:', err))
  .finally(() => client.close());
```

Operaciones Básicas en la Base de Datos

Una vez que la conexión se ha establecido, puedes realizar operaciones básicas en la base de datos, como inserción, actualización, lectura y eliminación de documentos. MongoDB utiliza colecciones para almacenar documentos.

```
const collection = db.collection('miColeccion');

// Insertar un documento
collection.insertOne({ campo: 'valor' });

// Buscar documentos
const resultados = collection.find({ campo: 'valor' });

// Actualizar documentos
collection.updateOne({ campo: 'valor' }, { $set: { campoActualizado: 'nuevoValor' } });

// Eliminar documentos
collection.deleteOne({ campo: 'valor' });
```

6. Implementación de autenticación y autorización en aplicaciones web

La autenticación y autorización son dos pilares fundamentales en el desarrollo de aplicaciones web. La autenticación se encarga de verificar la identidad de los usuarios, mientras que la autorización regula el acceso de los usuarios a recursos y funcionalidades específicas.

Importancia de la Autenticación y Autorización

La autenticación y autorización son fundamentales para garantizar la seguridad, la privacidad y el control de acceso en aplicaciones web. La autenticación se encarga de verificar que los usuarios son quienes dicen ser, mientras que la autorización determina qué recursos y funcionalidades tienen permitido utilizar.

Métodos de Autenticación

Existen varios métodos comunes de autenticación, entre ellos:

· **Autenticación de Usuario y Contraseña:** Los usuarios proporcionan un nombre de usuario y una contraseña para verificar su identidad. Es el método tradicional y ampliamente utilizado.

· **Autenticación Basada en Tokens:** Se emite un token de acceso después de la autenticación exitosa, que se utiliza para identificar al usuario en solicitudes posteriores. Esta forma es muy utilizada en aplicaciones de una sola página (SPA) y servicios web.

· **Autenticación de Terceros:** Permite a los usuarios autenticarse a través de servicios de terceros, como Google, Facebook o Twitter. Este método simplifica el proceso de registro e inicio de sesión para los usuarios.

Implementación de Autenticación y Autorización en Node.js

La implementación de autenticación y autorización en una aplicación Node.js implica los siguientes pasos:

1. **Definir Estrategias de Autenticación:** Debes configurar cómo los usuarios se autenticarán, ya sea mediante contraseñas, tokens u otros métodos. En Node.js, la biblioteca Passport es una herramienta común para implementar estrategias de autenticación.

2. **Crear Rutas de Autenticación:** Debes definir rutas para registrar, iniciar sesión y cerrar sesión de usuarios. Por ejemplo, puedes utilizar las rutas "/registro", "/inicio-sesion" y "/cerrar-sesion" para estas funcionalidades.

3. **Proteger Rutas:** Utiliza middleware de autenticación para proteger las rutas que requieran autenticación. Esto garantiza que solo los usuarios autenticados tengan acceso a esas rutas. En Node.js, Passport y middleware como `passport.authenticate` se utilizan para este propósito.

Gestión de Permisos y Roles

Los permisos definen qué acciones pueden realizar los usuarios, como "leer", "escribir" o "eliminar". Los roles agrupan permisos y se asignan a los usuarios. Por ejemplo, puedes tener roles como "usuario estándar" y "administrador". Los usuarios pueden tener uno o varios roles. Algunas estrategias comunes para la autorización incluyen:

· **Listas de Control de Acceso (ACL):** Asignan permisos directamente a usuarios o recursos.

· **Sistemas Basados en Roles:** Asignan roles a usuarios y agrupan permisos en roles.

7. Seguridad y prevención de vulnerabilidades en el desarrollo back-end

La seguridad es un aspecto crítico en el desarrollo back-end. La prevención de vulnerabilidades y la protección de datos son responsabilidades fundamentales para los desarrolladores. Es fundamental por

varias razones:

· **Protección de Datos Sensibles:** Las aplicaciones web a menudo manejan datos confidenciales, como información de usuarios, contraseñas, datos financieros y más. La filtración de estos datos puede tener consecuencias graves.

· **Mantenimiento de la Reputación:** Las brechas de seguridad pueden dañar la reputación de una empresa o servicio, lo que a su vez puede llevar a la pérdida de usuarios o clientes.

· **Cumplimiento Legal:** En muchas jurisdicciones, existen regulaciones estrictas sobre la protección de datos y la privacidad. No cumplir con estas regulaciones puede resultar en sanciones legales.

Vulnerabilidades Comunes en el Desarrollo Back-End

Existen numerosas vulnerabilidades que pueden explotarse en aplicaciones web. Algunas de las vulnerabilidades más comunes incluyen:

Inyección SQL

Permite a los atacantes ejecutar comandos SQL maliciosos en una base de datos. La prevención de este tipo de vulnerabilidad implica utilizar consultas parametrizadas o funciones proporcionadas por el ORM para evitar la inyección de SQL.

Cross-Site Scripting (XSS)

Permite a los atacantes inyectar scripts maliciosos en páginas web vistas por otros usuarios. La prevención de XSS implica validar y escapar adecuadamente los datos de entrada y evitar la inserción directa de contenido no confiable en el DOM del navegador.

Cross-Site Request Forgery (CSRF)

Un ataque que fuerza a los usuarios a realizar acciones no deseadas en una aplicación web en la que ya están autenticados. La prevención de CSRF implica el uso de tokens CSRF y la comprobación del origen de las solicitudes para garantizar que solo las solicitudes legítimas sean procesadas.

Autenticación Débil

Contraseñas débiles o autenticación inadecuada pueden exponer cuentas de usuario. La prevención implica el uso de políticas de contraseñas sólidas, almacenamiento seguro de contraseñas (hashing y salting) y la implementación de autenticación de dos factores.

Exposición de Datos Sensibles

Revelar información confidencial a través de errores en la aplicación. Para prevenir la exposición de datos sensibles, se deben evitar mensajes de error detallados que muestren información confidencial y limitar la

cantidad de información proporcionada en respuestas de error.

Estrategias de Prevención de Vulnerabilidades

· **Validación de Datos de Entrada:** Validar y filtrar todos los datos de entrada para evitar inyecciones y ataques XSS.

· **Uso de Parámetros Preparados:** Utilizar parámetros preparados en consultas de bases de datos para evitar inyecciones SQL.

· **Seguridad en la Autenticación:** Implementar autenticación sólida con contraseñas seguras, autenticación de dos factores y gestión de sesiones segura.

· **Protección CSRF:** Utilizar tokens CSRF y comprobar el origen de las solicitudes para prevenir ataques CSRF.

· **Control de Acceso:** Implementar un control de acceso estricto y asignar permisos y roles de usuario adecuados.

La protección de datos es fundamental. Utiliza cifrado para proteger datos sensibles en tránsito y en reposo. Realiza auditorías de seguridad periódicas y asegúrate de mantener todas las bibliotecas y dependencias actualizadas para corregir posibles vulnerabilidades.

Al comprender las vulnerabilidades comunes y aplicar las mejores prácticas de seguridad, los desarrolladores pueden crear aplicaciones web más seguras y proteger los datos de los usuarios. La seguridad debe ser una preocupación constante a lo largo del ciclo de vida de desarrollo y pruebas de una aplicación.

Bibliografía utilizada y sugerida

· Documentation | Node.js. (s. f.). Node.js. https://nodejs.org/en/docs

· Infante, D. C. H., & Infante, D. C. H. (2023, 13 febrero). Seguridad en aplicaciones web: qué es, cómo funciona y los mejores servicios. Tutoriales Hostinger. https://www.hostinger.com.ar/tutoriales/seguridad-en-aplicaciones-web

· MongoDB Node Driver — Node.js. (s. f.). https://www.mongodb.com/docs/drivers/node/current/

· Node.js MongoDB Create database. (s. f.). https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp

· Node.Js tutorial. (s. f.). https://www.w3schools.com/nodejs/default.asp