Rutas en Node.js

CÁTEDRA: ESTRATEGIAS DE PERSISTENCIA

¿Qué son las Rutas (Routes)?

Las rutas (routes) son endpoints que se definen para gestionar diferentes solicitudes HTTP (GET, POST, PUT, DELETE, etc.) en una aplicación web.

Cuando una solicitud HTTP llega al servidor, la ruta se encarga de determinar qué función se debe ejecutar para manejar dicha solicitud. Esto es esencial para crear aplicaciones web donde diferentes partes de la aplicación responden a diferentes solicitudes del cliente.

Concepto Básico de Rutas en Express.js

Una ruta consiste típicamente en tres componentes:

- **Método HTTP:** El tipo de solicitud HTTP, como GET, POST, PUT, DELETE, etc.
- **Ruta de acceso (path):** La parte de la URL después del dominio que se está apuntando, por ejemplo, **/users** o **/products/:id**.
- **Controlador de función:** Una función que se ejecuta cuando la ruta coincide con una solicitud entrante.

Métodos HTTP más comunes

Método	Descripción	Ejemplo
GET	Recuperar datos de un servidor. Es utilizado para solicitar un recurso específico o para obtener información.	Cuando visitas una página web, el navegador generalmente realiza una solicitud GET para obtener el contenido de la página.
POST	Enviar datos al servidor para crear un nuevo recurso o procesar datos. Es utilizado comúnmente para enviar datos de formularios o para cargar archivos.	Cuando llenas un formulario de registro en un sitio web y haces clic en "Enviar", el navegador envía una solicitud POST con los datos del formulario.
PUT	Actualizar un recurso existente en el servidor o crear uno nuevo si no existe. Es utilizado para enviar datos que deben reemplazar por completo el recurso en el servidor.	Actualizar la información de un perfil de usuario.

Métodos HTTP más comunes

Método	Descripción	Ejemplo
PATCH	Aplicar modificaciones parciales a un recurso existente. A diferencia de PUT, que reemplaza completamente un recurso, PATCH solo aplica los cambios necesarios.	Cambiar solo la dirección de correo electrónico de un usuario.
DELETE	Eliminar un recurso en el servidor.	Cuando deseas eliminar una entrada en un sistema de administración de contenidos.

Ejemplo de Rutas en Express.js

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

// Ruta GET para el path '/'
app.get('/', (req, res) => {
  res.send('¡Hola Mundo!');
});

// Ruta GET para el path '/usuarios'
app.get('/usuarios', (req, res) => {
  res.send('Lista de usuarios');
});
```

Ejemplo de Rutas en Express.js

```
// Ruta POST para el path '/usuarios'
app.post('/usuarios', (req, res) => {
  // Lógica para manejar la creación de un nuevo usuario
  res.send('Usuario creado');
});

// Ruta GET para un usuario específico con ID dinámico
app.get('/usuarios/:id', (req, res) => {
  const userId = req.params.id; // Captura el valor de ID dinámico
  res.send(`Detalles del usuario con ID ${userId}`);
});

app.listen(3000, () => {
  console.log('Servidor ejecutándose en http://localhost:3000');
});
```

Explicación del Ejemplo

- **app.get('/'):** Define una ruta que responde a solicitudes GET a la raíz del sitio (/). Cuando un cliente realiza una solicitud GET a la raíz, el servidor responde con "¡Hola Mundo!".
- **app.get('/usuarios'):** Define una ruta que responde a solicitudes GET a **/usuarios**. Esto podría usarse para listar todos los usuarios.
- **app.post('/usuarios'):** Define una ruta que responde a solicitudes POST a **/usuarios**. Aquí podrías manejar la lógica para crear un nuevo usuario.

Explicación del Ejemplo

- **app.get('/usuarios/:id'):** Define una ruta con un parámetro dinámico (**:id**). Esta ruta responderá a solicitudes GET a URLs como **/usuarios/1** o **/usuarios/42**, donde **id** es un parámetro de ruta que se puede acceder a través de **req.params.id**.

Importancia de las Rutas en Node.js

- **Organización del Código:** Facilitan la separación del código por funcionalidad, permitiendo que cada endpoint tenga su propia lógica.

- **Escalabilidad:** Ayudan a escalar la aplicación agregando nuevas rutas y lógica asociada sin afectar las rutas existentes.

- **Mantenibilidad:** Al tener rutas bien definidas y organizadas, es más fácil mantener y actualizar el código en el futuro.

Controladores en Node.js

¿Qué es un Controlador (Controller)?

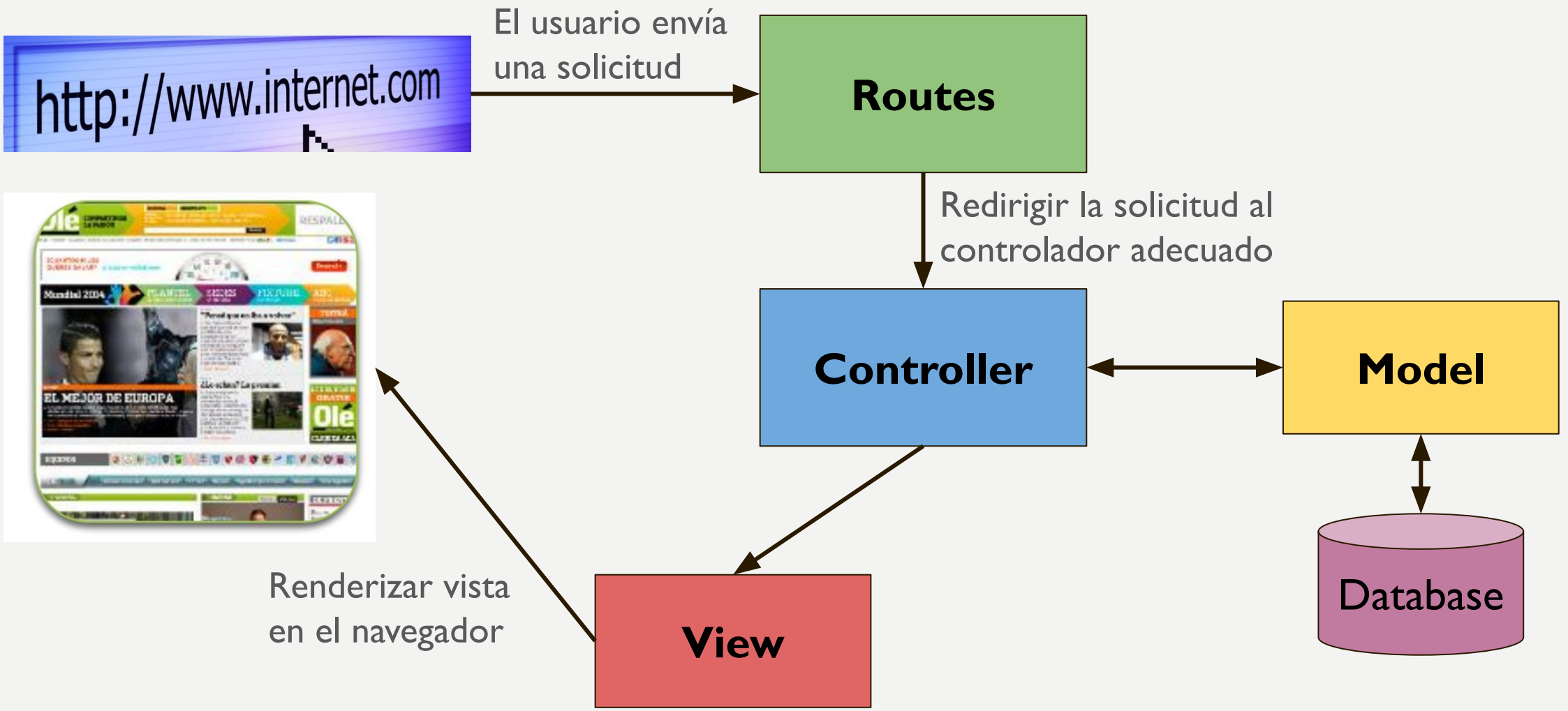
Los controladores (o handlers) son funciones que gestionan la lógica de la aplicación para responder a las solicitudes HTTP en puntos finales (endpoints) específicos de una aplicación.

¿Qué es un Controlador (Controller)?

Un controlador es una función que se ejecuta cuando se recibe una solicitud HTTP en una ruta específica de tu aplicación.

Su propósito principal es manejar la solicitud, procesar cualquier dato necesario, interactuar con la base de datos u otros servicios, y enviar una respuesta al cliente.

Diagrama de ruta en el desarrollo web



Características de un Controlador

- **Manejo de Solicitudes:** Los controladores reciben objetos de solicitud (**req**) y de respuesta (**res**) que contienen toda la información necesaria para manejar la solicitud entrante y enviar la respuesta.
- **Procesamiento de Datos:** Pueden realizar diversas tareas como validar datos de entrada, realizar cálculos, o formatear respuestas.

Características de un Controlador

- **Interacción con Modelos o Bases de Datos:** A menudo, los controladores interactúan con modelos que representan la lógica de negocio o las bases de datos para leer, insertar, actualizar o eliminar datos.
- **Envío de Respuestas:** Después de procesar la solicitud, el controlador envía una respuesta al cliente utilizando el objeto de respuesta (**res**).

Ejemplo de Rutas en Express.js

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

// Controlador para manejar la solicitud GET en la ruta '/'
const homeController = (req, res) => {
  res.send('¡Bienvenido a la página principal!');
};

// Definir rutas y asociarlas con controladores
app.get('/', homeController);

app.listen(3000, () => {
  console.log('Servidor ejecutándose en http://localhost:3000');
});
```

Explicación del Ejemplo

homeController: Es una función que actúa como un controlador para la ruta '/'. Simplemente envía un mensaje de bienvenida cuando se accede a la ruta raíz del sitio web.

Otro ejemplo de Rutas en Express.js

```
// Controlador para manejar la solicitud GET en la ruta '/usuarios/:id'
const getUserController = (req, res) => {
  const userId = req.params.id;
  // Simulando la obtención de datos de usuario
  const user = { id: userId, name: 'Juan Pérez' };
  res.json(user);
};

// Definir rutas y asociarlas con controladores
app.get('/usuarios/:id', getUserController);
```

Explicación del Ejemplo

getUserController: Es un controlador para la ruta '**/usuarios/:id**'. Captura el parámetro de ruta **id** desde la URL, simula la obtención de un usuario, y envía el objeto de usuario en formato JSON como respuesta.

Ventajas de Usar Controladores

- **Modularidad:** Los controladores permiten separar la lógica de manejo de solicitudes de la definición de rutas, lo que hace que el código sea más modular y fácil de mantener.
- **Reutilización:** Puedes reutilizar los controladores en diferentes partes de la aplicación, lo que evita la duplicación de código.
- **Mantenibilidad:** Separar la lógica del controlador de la lógica de la ruta ayuda a mantener el código limpio y legible, facilitando su mantenimiento y pruebas.

Organización de Controladores

En aplicaciones más grandes, es común organizar los controladores en archivos separados dentro de una carpeta llamada **controllers** o similar. Esto ayuda a mantener una estructura de código organizada y facilita la colaboración en proyectos con múltiples desarrolladores.

Ejemplo de Organización de Controladores

Imagina una aplicación que maneja usuarios y productos. Podrías tener un archivo **userController.js** para todas las funciones relacionadas con los usuarios y otro archivo **productController.js** para los productos.

Ejemplo de Organización de Controladores

```
src/
│
├── controllers/
│   ├── userController.js
│   └── productController.js
│
├── routes/
│   ├── userRoutes.js
│   └── productRoutes.js
│
└── app.js
```

Postman

Postman

Postman

Existen muchas herramientas para facilitar las pruebas de backends.

Uno de ellos es Postman.

Postman permite a los desarrolladores enviar solicitudes HTTP (GET, POST, PUT, DELETE, etc.) a servidores web y obtener respuestas.

Descarga de Postman

Ir al sitio oficial de Postman:

https://www.postman.com/downloads/

HTTP – Códigos de estados

El código de estado es un entero de 3 dígitos

Rango	Descripción
1XX	Informativos
2XX	Éxito
3XX	Redirección
4XX	Error de cliente
5XX	Error de servidor

HTTP – Códigos de estados

Los más comunes

Código	Descripción
200	OK Solicitud exitosa. La respuesta se envía en el cuerpo
404	Not Found El recurso no existe.
303	See Other El recurso se ha movido a otra URL (Dada en el header Location)
500	Server Error Error no esperado en el servidor.