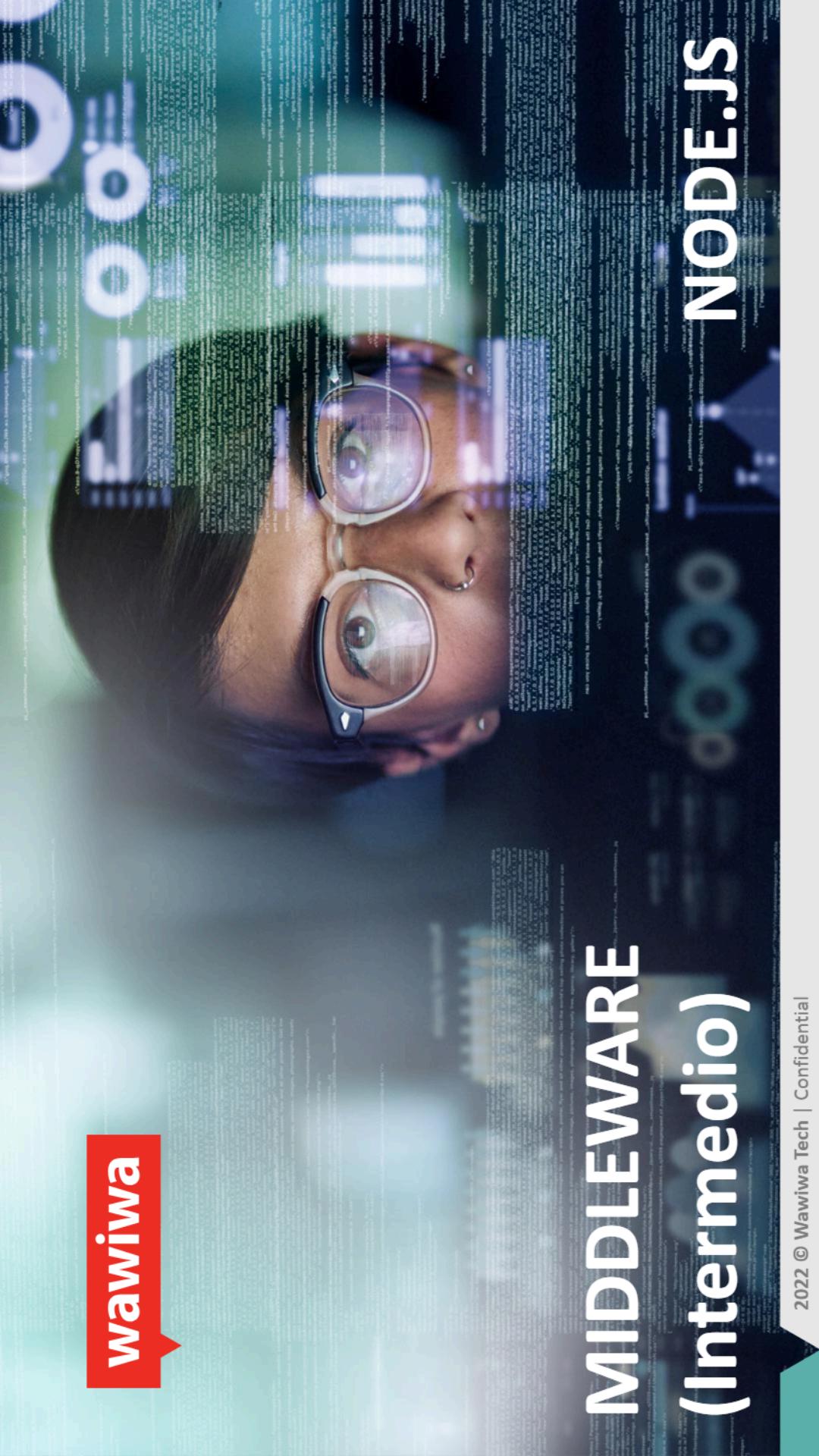




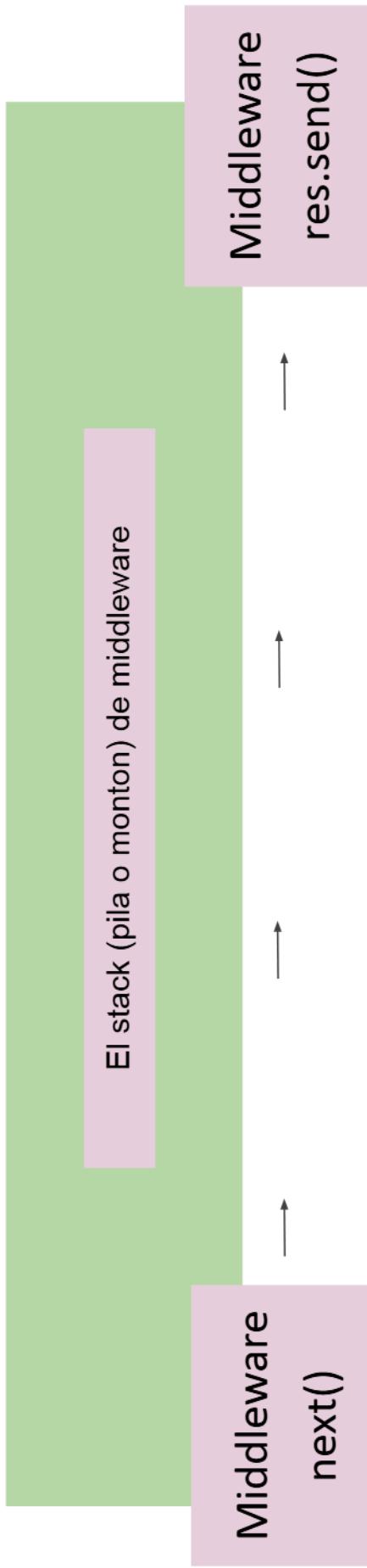
wawiwa

MIDDLEWARE (Intermedio)

NODE.JS



Ciclo de vida de una solicitud



Usar funciones de middleware

Las funciones middleware son funciones que tienen acceso al objeto de la solicitud (`req`), al objeto de la respuesta (`res`) y a la siguiente función en el ciclo de solicitud-respuesta de la aplicación. La siguiente función middleware es comúnmente detonada por una variable llamada “`next`”.

Express es un marco web de enrutamiento y middleware que tiene una funcionalidad propia mínima: una aplicación Express es esencialmente una serie de llamadas a funciones de middleware.

Las funciones middleware puede ejecutar las siguientes tareas:

- Ejecutar cualquier código.
- Hacer cambios a los objetos de solicitud y respuesta.
- Terminar los ciclos solicitud-respuesta.
- Llamar a la siguiente función middleware en la pila.

Si la función middleware actual no termina el ciclo solicitud-respuesta, debe llamar `next()` para pasar el control a la siguiente función middleware. De otra manera la solicitud va a ser dejada colgando.

Aplicación a nivel middleware

Vincula el middleware de nivel de aplicación a una instancia del objeto de aplicación mediante las funciones app.use() y app.METHOD(), donde METHOD es el método HTTP de la solicitud que maneja la función de middleware (como GET, PUT o POST) en minúsculas.

Este ejemplo muestra una función middleware sin una ruta de montaje.

La función es ejecutada cada vez que la app recibe una solicitud.

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

app.use((req, res, next) => {
  console.log(`Time: ${Date.now()}`);
  next();
});

app.listen(3000);
```

← → C ⓘ localhost:3000

cannot GET /

↓
Time: 1639299082293
□

Aplicación a nivel middleware

Este ejemplo muestra una función middleware montada en la ruta `/user/:id`. La función es ejecutada por cualquier tipo de solicitud HTTP en la ruta `/user/:id`.

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

app.use((req, res, next) => {
  console.log(`Time: ${Date.now()}`);
  next();
});

app.use('/user/:id', (req, res, next) => {
  console.log(`Request Type: ${req.method}`);
  next();
});

app.listen(3000);
```

En la pila de este servidor no hay respuesta

Aplicación a nivel middleware

server2.js

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

app.use('/user/:id', (req, res, next) => {
  console.log('Request Type:', req.method);
  next();
});

app.get('/user/:id', (req, res, next) => {
  res.send('USER');
});

app.listen(3000);
```

Aplicación a nivel middleware

Middleware también puede ser declarado en un arreglo para su reutilización.

Este ejemplo muestra un arreglo con un sub-montón de middleware que maneja solicitudes GET a la ruta /user/:id

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

function logOriginalUrl(req, res, next) {
  console.log(`Request URL: ${req.originalUrl}`);
  next();
}

function logMethod(req, res, next) {
  console.log(`Request Type: ${req.method}`);
  next();
}

const logStuff = [logOriginalUrl, logMethod];

app.get('/user/:id', logStuff, (req, res, next) => {
  res.send('User Info');
});

app.listen(3000);
```

```
graph TD
    S[solicitud] --> LO[logOriginalUrl]
    LO --> LM[logMethod]
    LM --> N1[next()]
    N1 --> R[res.send("User Info")]
    R -- respuesta --> RI[res.send("User Info")]
```

Middleware tercero

Npm install cookie-parser

Usa un middleware tercero para agregar funcionalidad a tus apps de Express.

Instala el módulo Node.js para la funcionalidad requerida, después descárgalo en tu app al nivel de la aplicación o del enrutamiento.

El siguiente ejemplo ilustra la instalación y carga de la función de middleware de análisis de cookies cookie-parser.

express.Router()

part5/server.js

```
1 var express = require("express");
2 var fs = require("fs");
3 var app = express();
4 var toyRouter = express.Router();
5 app.use('/api/v1/toys', toyRouter);
6 app.use(express.json());
7 var port = 3000;
8 app.use('/assets', express.static(__dirname + '/public'));
9 let toys = JSON.parse(fs.readFileSync("./data/data.json", 'utf-8'));
10 > let getAllToys=(req, res)=>{ ...
11 }
12 > let createToy=(req, res)=>{ ...
13 }
14 > let getToyById=(req, res)=>{ ...
15 }
16 > let updateToyById=(req, res)=>{ ...
17 }
18 > let deleteToyById=(req, res)=>{ ...
19 }
20 > let toyRouter.get('/', getAllToys);
21 toyRouter.post('/toys', createToy);
22 toyRouter.get('/:id', getToyById);
23 toyRouter.patch('/id', updateToyById);
24 toyRouter.delete('/id', deleteToyById);
25 }
26 toyRouter.listen(port);
```

Declaro toyRouter

usa toyRouter cuando el usuario pide por /api/v1/toys

Puedes borrar /api/v1/toys
Porque lo mencionamos en toyRouter

Hazlo tú mismo 1- ejercicio de refactorización 3 del capítulo 11 y agrega un enrutamiento en acorde a la última diapositivas

Crea un servidor express que responda a las siguientes solicitudes:

Solicitud	Respuesta	Método	Comentario
/api/v1/players	Todos los "players"	Get	Obtener todos los "players" de players.json
/api/v1/players	El "player creado"	Post	Crea un nuevo Player
/api/v1/players/:id	"Player" por su id	Get	Obtener el "player" por su id
/api/v1/players/:id	Actualizar "player"	Patch	Actualizar "player" por su id
/api/v1/players/:id	Borrar "player"	Delete	Borrar "player" por su id

wawiwa

¿Preguntas?