

Express.js

Antes de empezar

```
$ npm install lodash
```

Fundamentos

Fundamentos

- Express.js es un *framework*
- Escribir servidores web
- Minimalista
- Complejidad orgánica a partir de http

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

app.use((req, res) => {
  res.end('Hello, World!')
})

http.createServer(app).listen(3000)
```

Fundamentos

- ***app*** es una función
- el *handler* que le pasamos a ***http.createServer***
- ***express*** = conjunto de herramientas para **facilitar** la escritura del *handler*
- Que no se nos olvide: ¡***app*** no es más que el handler!

Fundamentos

- la operación fundamental es **app.use(fn)**
 - ejecuta **fn** cada vez que se reciba una petición
 - Manejar la request
 - Generar una respuesta


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

```
app.use((req, res) => {  
  res.end('Hello, World!')  
})
```

```
http.createServer(app).listen(3000)
```

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

```
app.use((req, res) => {
  res.end('Hello, World!')
})
```

```
http.createServer(app).listen(3000)
```

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

app.use((req, res) => {
  res.end('Hello, World!')
})
```

```
http.createServer(app).listen(3000)
```

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

app.use((req, res) => {
  res.end('Hello, World!')
})

app.listen(3000)
```

Fundamentos

- Los parámetros **req** y **res**...
 - Subclases de `http.IncomingMessage` y `http.ServerResponse`
 - Métodos añadidos
 - Para facilitarnos la vida

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

app.use((req, res) => {
  res.status(200).json({ hello: 'world' })
})

app.listen(3000)
```

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

```
const app = express()
```

```
app.use((req, res) => {
```

```
  res.status(200).json({ hello: 'world' })
```

```
})
```

```
app.listen(3000)
```

Fundamentos

- req
 - *req.cookies*
 - *req.body*
 - *req.query*
 - *req.method*
 - ...

Fundamentos

- **res**

- `res.cookie(...)`
- `res.status(...)`
- `res.send(...)`
- `res.json(...)`
- `res.redirect(...)`
- ...

Ejercicio

- Escribe una **app express** que sirva **su propio código** por **HTTP**
 - <http://localhost:3000/>

Middleware

Middleware

- Podemos invocar **app.use(fn)** múltiples veces
 - poner **múltiples handlers** a una misma request
 - **encadenados** en serie
 - para dividir **las responsabilidades** en fragmentos
 - **independientes**
 - **reutilizables**

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

```
app.use((req, res, next) => {  
  console.log('this runs first')  
  next()  
})
```

```
app.use((req, res) => {  
  res.send('Hello, world!')  
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {
  console.log('this runs first')
  next()
})
```

```
app.use((req, res) => {
  res.send('Hello, world!')
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {
  console.log('this runs first')
  next()
})
```

```
app.use((req, res) => {
  res.send('Hello, world!')
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {  
  console.log('this runs first')  
  next()  
})
```

```
app.use((req, res) => {  
  res.send('Hello, world!')  
})
```

```
app.listen(3000)
```


Middleware

- Cada uno de los handlers es un **middleware**
 - Reciben **req**, **res** y **next**
 - Podemos **encadenar** tantos como queramos
 - *Generalmente*, el último middleware genera la respuesta

Ejercicio

- Escribe una **app express** con **dos middlewares**:
 - El 50% de las veces, el primero contesta y termina la conexión
 - envía “CARA”
 - El otro 50%, es el segundo el que contesta
 - envía “CRUZ”

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

app.use((req, res, next) => {
  Math.random() > .5 ? next() : res.send('Cara!')
})

app.use((req, res) => {
  res.send('Cruz!')
})

app.listen(3000)
```

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

```
app.use((req, res, next) => {  
  Math.random() > .5 ? next() : res.send('Cara!')  
})
```

```
app.use((req, res) => {  
  res.send('Cruz!')  
})
```

```
app.listen(3000)
```

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

app.use((req, res, next) => {
  console.log('Never gonna give you up...')
})

app.listen(3000)
```

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

app.use((req, res, next) => {
  next()
})

app.listen(3000)
```

Middleware

- Cada **middleware**...
 - Tiene que **llamar a next** o **generar una respuesta!**
 - Si no, la petición se queda **abierta** hasta que de timeout
 - Si ningún middleware genera respuesta, **express** contesta con **404**

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

```
app.use((req, res, next) => {  
  res.send('Hola, Don Pepito')  
})
```

```
app.use((req, res, next) => {  
  res.send('Hola, Don José')  
})
```

```
app.listen(3000)
```



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

```
app.use((req, res, next) => {  
  res.send('Hola, Don Pepito')  
})
```

```
app.use((req, res, next) => {  
  res.send('Hola, Don José')  
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {  
  res.send('Hola, Don Pepito')  
  next()  
})
```

```
app.use((req, res, next) => {  
  res.send('Hola, Don José')  
})
```

```
app.listen(3000)
```

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

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

```
app.use((req, res, next) => {  
  res.locals.start = Date.now()  
  next()  
})
```

```
app.use((req, res) => {  
  const { start } = res.locals  
  setTimeout(() => {  
    res.send(`Esta petición tardó ${Date.now() - start}ms`)  
  }, Math.random() * 2000, )  
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {
  res.locals.start = Date.now()
  next()
})
```

```
app.use((req, res) => {
  const { start } = res.locals
  setTimeout(() => {
    res.send(`Esta petición tardó ${Date.now() - start}ms`)
  }, Math.random() * 2000, )
})
```

```
app.listen(3000)
```

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

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

```
app.use((req, res, next) => {  
  res.locals.start = Date.now()  
  next()  
})
```

```
app.use((req, res) => {  
  const { start } = res.locals  
  setTimeout(() => {  
    res.send(`Esta petición tardó ${Date.now() - start}ms`)  
    }, Math.random() * 2000, )  
})
```

```
app.listen(3000)
```

Ejercicio

- Escribe una **app express** que devuelva **diez** tiradas de un dado de 6 caras
 - escribe un middleware que “añada” una tirada a la request
 - utiliza ese middleware para generar las 10 tiradas

Middleware

- Los **middlewares**...
 - Cumplen una función específica
 - decorar la petición
 - decorar la respuesta
 - Se pueden comunicar con otros middlewares mediante **res.locals**

Ejercicio

- Escribe un **middleware** para **autenticar** **peticiones**
 - si el string “sup3rs3cr3t” aparece en la url...
 - **req.url**
 - la request está autenticada!
 - ponemos **req.authenticated** a **true**
 - en **otro middleware**, comprobamos **req.authenticated**
 - si es **true**, contestamos correctamente
 - si es **false**, contestamos con un **401**


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

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

```
app.use((req, res, next) => {
```

```
  req.authenticated = req.url.match('sup3rs3cr3t')
```

```
  next()
```

```
})
```

```
app.use((req, res) => {
```

```
  res.send(req.authenticated ? 'Bienvenido' : 'Sal de aquí!')
```

```
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {  
  req.authenticated = req.url.match('sup3rs3cr3t')  
  next()  
})
```

```
app.use((req, res) => {  
  res.send(req.authenticated ? 'Bienvenido' : 'Sal de aquí!')  
})
```

```
app.listen(3000)
```

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

app.use((req, res, next) => {
  res.red = (msg) => res.send(`
    <html><body>
      <h1 style='color: red'>${msg}</h1>
    <body/></html>
  `)
  next()
})

app.use((req, res, next) => {
  res.red('Hello, World')
})

app.listen(3000)
```

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

app.use((req, res, next) => {
  res.red = (msg) => res.send(`
    <html><body>
      <h1 style='color: red'>${msg}</h1>
    </body></html>
  `)
  next()
})

app.use((req, res, next) => {
  res.red('Hello, World')
})

app.listen(3000)
```

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

app.use((req, res, next) => {
  const now = new Date()
  console.log(`[${now}] request to ${req.url}...`)
  res.locals.start = now.getTime()
  next()
})

app.use((req, res, next) => {
  res.send('Hola, Mundo!')
  next()
})

app.use((req, res) => {
  const delta = Date.now() - res.locals.start
  console.log(`${req.url} handled in ${delta} ms\n`)
})

app.listen(3000)
```

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

app.use((req, res, next) => {
  const now = new Date()
  console.log(`[${now}] request to ${req.url}...`)
  res.locals.start = now.getTime()
  next()
})

app.use((req, res, next) => {
  res.send('Hola, Mundo!')
  next()
})

app.use((req, res) => {
  const delta = Date.now() - res.locals.start
  console.log(`${req.url} handled in ${delta} ms\n`)
})

app.listen(3000)
```

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

app.use((req, res, next) => {
  const now = new Date()
  console.log(`[${now}] request to ${req.url}...`)
  res.locals.start = now.getTime()
  next()
})

app.use((req, res, next) => {
  res.send('Hola, Mundo!')
  next()
})

app.use((req, res) => {
  const delta = Date.now() - res.locals.start
  console.log(`${req.url} handled in ${delta} ms\n`)
})

app.listen(3000)
```

Middleware

- En resumen, **middleware** puede...
 - Ejecutar código arbitrario
 - Modificar los objetos **req** y **res**
 - Terminar la petición enviando una respuesta
 - Llamar al siguiente middleware de la cadena


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

```
app.use((req, res, next) => {
  throw new Error('Oh, oh...')
  next()
})
```

```
app.use((req, res, next) => {
  res.send('Hola, Mundo!')
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {
  throw new Error('Oh, oh...')
  next()
})
```

```
app.use((req, res, next) => {
  res.send('Hola, Mundo!')
})
```

```
app.listen(3000)
```

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

```
app.use((req, res, next) => {
  next(new Error('Oh, oh...'))
})
```

```
app.use((req, res, next) => {
  res.send('Hola, Mundo!')
})
```

```
app.listen(3000)
```

Middleware

- Podemos **capturar errores**
 - con un **middleware especial**
 - que recibe **cuatro parámetros**
 - recibe los errores emitidos por los middlewares *que le preceden en la cadena*
 - **pero NO** los que se añaden después!

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

app.use((req, res, next) => {
  next(new Error('Un bug!'))
})

app.use((req, res, next) => {
  res.send('Hola, Mundo!')
})

app.use((err, req, res, next) => {
  res.send(`Paso algo malo: ${err.message}`)
})

app.listen(3000)
```

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

```
app.use((req, res, next) => {
  next(new Error('Un bug!'))
})
```

```
app.use((req, res, next) => {
  res.send('Hola, Mundo!')
})
```

```
app.use((err, req, res, next) => {
  res.send(`Paso algo malo: ${err.message}`)
})
```

```
app.listen(3000)
```

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

app.use((req, res, next) => {
  next()
})

app.use((err, req, res, next) => {
  res.send(`Paso algo malo: ${err.message}`)
})

app.use((req, res, next) => {
  res.send('Hola, Mundo!')
})

app.listen(3000)
```

Ejercicio

- Escribe un **middleware**...
 - que **inspeccione** `req.path`
 - si existe un fichero con esa ruta, lo sirva
 - partiendo de la carpeta `__dirname/public`
 - si no existe, continúa con la cadena

Middleware

- **app.use(...)** acepta **múltiples middlewares** de una vez
 - Un **array de funciones**
 - Múltiples funciones como **múltiples parámetros**
 - Cualquier **combinación** de los anteriores

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

const cara = (req, res, next) => {
  Math.random() > .5 ? next() : res.send('Cara!')
}

const cruz = (req, res) => res.send('Cruz!')

app.use([cara, cruz])

app.listen(3000)
```

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

const cara = (req, res, next) => {
  Math.random() > .5 ? next() : res.send('Cara!')
}

const cruz = (req, res) => res.send('Cruz!')

app.use([cara, cruz])

app.listen(3000)
```

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

const cara = (req, res, next) => {
  Math.random() > .5 ? next() : res.send('Cara!')
}

const cruz = (req, res) => res.send('Cruz!')

app.use(cara, cruz)

app.listen(3000)
```

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

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

```
const cara = (req, res, next) => {  
  Math.random() > .5 ? next() : res.send('Cara!')  
}
```

```
const cruz = (req, res) => res.send('Cruz!')
```

```
app.use(cara, cruz)
```

```
app.listen(3000)
```

Middleware

- Express.js es poco más que una “carcasa”
 - casi **toda** la funcionalidad son middlewares
 - son un mecanismo **muy flexible**

```
const bottles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
app.use((req, res, next) => {  
  res.locals.lines = ['Voy a hacer una torre de botellas...']  
  next()  
})
```

```
app.use(bottles.map(n => (req, res, next) => {  
  const { lines } = res.locals  
  if (Math.random() < .2) {  
    lines.push('Oh no, todas se cayeron!')  
    res.send(lines.join('<br/>'))  
  } else {  
    lines.push(`ya llevo ${n} botellas...`)  
    next()  
  }  
}))
```

```
app.use((req, res) => {  
  const { lines } = res.locals  
  lines.push('Ya están todas!')  
  res.send(lines.join('<br/>'))  
})
```

```
const bottles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
app.use((req, res, next) => {  
  res.locals.lines = ['Voy a hacer una torre de botellas...']  
  next()  
})
```

```
app.use(bottles.map(n => (req, res, next) => {  
  const { lines } = res.locals  
  if (Math.random() < .2) {  
    lines.push('Oh no, todas se cayeron!')  
    res.send(lines.join('<br/>'))  
  } else {  
    lines.push(`ya llevo ${n} botellas...`)  
    next()  
  }  
}))
```

```
app.use((req, res) => {  
  const { lines } = res.locals  
  lines.push('Ya están todas!')  
  res.send(lines.join('<br/>'))  
})
```



```
const bottles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
app.use((req, res, next) => {  
  res.locals.lines = ['Voy a hacer una torre de botellas...']  
  next()  
})
```

```
app.use(bottles.map(n => (req, res, next) => {  
  const { lines } = res.locals  
  if (Math.random() < .2) {  
    lines.push('Oh no, todas se cayeron!')  
    res.send(lines.join('<br/>'))  
  } else {  
    lines.push(`ya llevo ${n} botellas...`)  
    next()  
  }  
}))
```

```
app.use((req, res) => {  
  const { lines } = res.locals  
  lines.push('Ya estan todas!')  
  res.send(lines.join('<br/>'))  
})
```

```
const bottles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
app.use((req, res, next) => {  
  res.locals.lines = ['Voy a hacer una torre de botellas...']  
  next()  
})
```

```
app.use(bottles.map(n => (req, res, next) => {  
  const { lines } = res.locals  
  if (Math.random() < .2) {  
    lines.push('Oh no, todas se cayeron!')  
    res.send(lines.join('<br/>'))  
  } else {  
    lines.push(`ya llevo ${n} botellas...`)  
    next()  
  }  
}))
```

```
app.use((req, res) => {  
  const { lines } = res.locals  
  lines.push('Ya están todas!')  
  res.send(lines.join('<br/>'))  
})
```

Ejercicio

- Escribe un **middleware** que...
 - conteste con *“Hola, Mundo!”* cuando se acceda a la url **/hello**
 - conteste con 404 para *cualquier otra url*

Ejercicio

- Escribe una función **route(app, path, fn)** que automatice el proceso
 - instala un middleware en **app**
 - cuando la url de la petición coincide con **path** ejecuta **fn**
 - si no, continúa con la cadena

Ejercicio

```
const app = express()
```

```
function route(app, path, fn) {  
  // ...  
}
```

```
route(app, '/hello', (req, res) => res.send('Hello!'))  
route(app, '/bye', (req, res) => res.send('Goodbye!'))
```

Router

Router

- Un **middleware** que...
 - Permite asociar rutas a manejadores
 - Seleccionando el *verbo* http
 - Suele ser el **último** eslabón de la cadena

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

app.get('/hello', (req, res) => {
  res.send(`Welcome to ${req.url}`)
})

app.get('/bye', (req, res) => {
  res.send('Come back soon!')
})

app.listen(3000)
```



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

```
app.get('/hello' (req, res) => {
  res.send(`Welcome to ${req.url}`)
})
```

```
app.get('/bye' (req, res) => {
  res.send('Come back soon!')
})
```

```
app.listen(3000)
```

Router

- Un **middleware** que...
 - Permite asociar rutas a manejadores
 - Seleccionando el *verbo* http
 - Suele ser el **último** eslabón de la cadena

Router

- Tenemos un método para cada *verbo* **HTTP**:
 - **get**
 - **post**
 - **update**
 - **delete**
 - *(hay otros, pero se usan menos)*

```
app.use((req, res, next) => {  
  console.log('before...')  
  next()  
})
```

```
app.get('/hello', (req, res) => {  
  res.send(`Welcome to ${req.url}`)  
})
```

```
app.get('/bye', (req, res) => {  
  res.send('Come back soon!')  
})
```

```
app.use((req, res) => {  
  res.send('Sorry, I do not understand...')  
})
```

```
app.use((req, res, next) => {  
  console.log('before...')  
  next()  
})
```

```
app.get('/hello', (req, res) => {  
  res.send(`Welcome to ${req.url}`)  
})
```

```
app.get('/bye', (req, res) => {  
  res.send('Come back soon!')  
})
```

```
app.use((req, res) => {  
  res.send('Sorry, I do not understand...')  
})
```

```
app.use((req, res, next) => {  
  console.log('before...')  
  next()  
})
```

```
app.get('/hello', (req, res) => {  
  res.send(`Welcome to ${req.url}`)  
})
```

```
app.get('/bye', (req, res) => {  
  res.send('Come back soon!')  
})
```

```
app.use((req, res) => {  
  res.send('Sorry, I do not understand...')  
})
```

```
app.use((req, res, next) => {  
  console.log('before...')  
  next()  
})
```

```
app.get('/hello', (req, res) => {  
  res.send(`Welcome to ${req.url}`)  
})
```

```
app.get('/bye', (req, res) => {  
  res.send('Come back soon!')  
})
```

```
app.use((req, res) => {  
  res.send('Sorry, I do not understand...')  
})
```

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

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

```
app.get('/hello', (req, res) => {  
  res.send(`Welcome to ${req.url}`)  
})
```

```
app.get('/hello', (req, res) => {  
  res.send('Come back soon!')  
})
```

```
app.listen(3000)
```



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

app.get('/hello', (req, res, next) => {
  res.send(`Welcome to ${req.url}`)
  next()
})

app.get('/hello', (req, res) => {
  console.log('does this run?')
})

app.listen(3000)
```

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

```
app.get('/hello', (req, res, next) => {  
  res.send(`Welcome to ${req.url}`)  
  next()  
})
```

```
app.get('/hello', (req, res) => {  
  console.log('does this run?')  
})
```

```
app.listen(3000)
```

Ejercicio

- Escribe una **app** tal que...
 - tenga una ruta pública: **/public**
 - tenga una ruta privada: **/protected**
 - sólo se puede acceder si la url contiene “**s3cr3t**”
 - en caso contrario: **401**

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

function protected(req, res, next) {
  if (req.url.match(/s3cr3t/)) next()
  else res.status(401).send('Not authorized')
}

app.get('/hello', protected, (req, res, next) => {
  res.send(`Welcome to ${req.url}`)
  next()
})

app.listen(3000)
```

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

```
function protected(req, res, next) {
  if (req.url.match(/s3cr3t/)) next()
  else res.status(401).send('Not authorized')
}
```

```
app.get('/hello', protected, (req, res, next) => {
  res.send(`Welcome to ${req.url}`)
  next()
})
```

```
app.listen(3000)
```

```
const form = `
  <form action="/form" method="POST">
    <input name="field"/>
  </form>
</body>`
```

```
function readStream(stream, cb) {
  const chunks = []
  stream.on('data', chunk => chunks.push(chunk))
  stream.on('end', () => cb(Buffer.concat(chunks)))
}
```

```
app.get('/form', (req, res) => res.send(form))
```

```
app.post('/form', (req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' })
  readStream(req, data => res.end(data.toString()))
})
```

Router

- El método **.all(...)** maneja cualquier verbo

```
app.all('/', (req, res, next) => {  
  res.send('Welcome!')  
})
```

Router

- Las rutas pueden ser patrones
 - Descritos con $*$, $+$ y $?$
- También pueden ser expresiones regulares
- Se aplican en cascada
 - poner los más específicos arriba y los más generales abajo!


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

app.get('/users/*', (req, res) => res.send('Inside /users/'))

app.listen(3000)
```

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

app.get('/users/homer', (req, res) => res.send('Hi, Homer!'))
app.get('/users/*', (req, res) => res.send('Inside /users/'))
app.all('*', (req, res) => res.send('This place doesn\'t exist'))

app.listen(3000)
```

Router

- Podemos extraer segmentos de la ruta como **parámetros**
 - describiendo qué lugar ocupa el segmento que queremos extraer
 - dándole un nombre

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

app.get('/users/:name', (req, res) => {
  res.send(`Welcome, ${req.params.name}`)
})

app.listen(3000)
```

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

app.get('/users/:name', (req, res) => {
  res.send(`Welcome, ${req.params.name}`)
})

app.listen(3000)
```

Para no escribir una API

Para no escribir una API

- **express.js** trae integrado...
 - un servidor de ficheros estáticos
 - un sistema de rendero de templates
 - middleware para parsear body
 - urlencoded
 - json

```
const express = require('express')  
const app = express();  
  
app.use(express.static('path/to/public/folder'))  
  
app.listen(3000)
```



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

app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'pug')

app.get('/users/:name', (req, res) => {
  res.render('welcome', { name: req.params.name })
})

app.listen(3000)
```

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

```
app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'pug')
```

```
app.get('/users/:name', (req, res) => {
  res.render('welcome', { name: req.params.name })
})
```

```
app.listen(3000)
```

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

app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'pug')

app.get('/users/:name', (req, res) => {
  res.render('welcome', { name: req.params.name })
})

app.listen(3000)
```

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

app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'pug')

app.get('/users/:name', (req, res) => {
  res.render('welcome', { name: req.params.name })
})

app.listen(3000)
```

views/welcome.pug

```
html
  body
    h1 Welcome, #{name}
```

Para no escribir una API

- Por defecto, el cuerpo de la petición POST o PUT **no se parsea**
 - los datos vienen “en crudo”, tal cual los envía el navegador
 - urlencoded
 - multipart
 -

Para no escribir una API

- express.js trae dos middlewares para parsear `req.body`:
 - `express.urlencoded`
 - `express.json`

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

```
app.use(express.urlencoded())
```

```
app.post('/form', (req, res) => {  
  console.log(req.body)  
  res.send('Ok!')  
})
```

```
app.listen(3000)
```



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

```
app.use(express.urlencoded())
```

```
app.post('/form', (req, res) => {
  console.log(req.body)
  res.send('Ok!')
})
```

```
app.listen(3000)
```

Ejercicio

- Haz un **microsite** con una **encuesta**
 - **GET** / muestra el formulario
 - vista *form.pug*
 - **POST** / guarda el voto y muestra el resultado
 - vista *results.pug*

form.pug

```
body
  form(action='/form', method='POST')
    div
      label ¿Quién sería mejor presidente de EEUU?
    div
      label
        input(name='winner', type='radio', value='homer')
        span Homer Simpson
    div
      label
        input(name='winner', type='radio', value='fry')
        span Fry
    div
      label
        input(name='winner', type='radio', value='peter')
        span Peter Griffin

  button Estoy seguro!
```

results.pug

```
body
  table
    tr
      th Homer
      td #{homer}
    tr
      th Fry
      td #{fry}
    tr
      th Peter
      td #{peter}
```

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

```
app.use(express.json())
```

```
app.post('/data', (req, res) => {  
  console.log(req.body)  
  res.json({ status: 'ok' })  
})
```

```
app.listen(3000)
```

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

app.use(express.json())

app.post('/data', (req, res) => {
  console.log(req.body)
  res.json({ status: 'ok' })
})

app.listen(3000)
```

REST

REST

- Arquitectura para diseñar APIs
 - Organizar la información en “*recursos*”
 - *sustantivos!*
 - Cada recurso tiene una **url** asociada
 - `/users`, `/posts/12`, `/jeans/183/sizes`
 - Operaciones asignadas a los métodos HTTP estándar
 - GET, PATCH, POST, PUT, DELETE

REST

VERBO	PATH	INCORRECTO!	DESCRIPCIÓN
GET	<code>/users</code>	<code>/getAllUsers</code>	JSON con la lista de usuarios
GET	<code>/users/1</code>	<code>/getUser/1</code>	JSON con la info del usuario con ID 1
POST	<code>/users</code>	<code>/createUser</code>	Crear un nuevo usuario (JSON en body)
PUT	<code>/users/1</code>	<code>/updateUser</code>	Modificar el usuario con ID 1 (JSON en body)
DELETE	<code>/users</code>	<code>/deleteAllUsers</code>	Eliminar todos los usuarios
DELETE	<code>/users/1</code>	<code>/deleteUser</code>	Eliminar el usuario con ID 1

Ejercicio

- Crea una API con un recurso **/todolists** que implemente las 6 operaciones fundamentales
 - ***todo*list** es un objeto con las propiedades:
 - “title”, “createdAt”
 - almacenamiento en memoria

Ejercicio

- Añade a la API un recurso **/todolists/<id>/todos** que implemente las 6 operaciones fundamentales
 - un todo tiene las propiedades:
 - “todolistId”, “title”, “done”
 - almacena los ***todos*** en memoria

Estrategias de autenticación

Estrategias de autenticación

- Necesitamos saber **quién** está haciendo la petición
 - proteger recursos
 - controlar qué operaciones puede realizar
 - identificarle como dueño de los recursos que cree

Estrategias de autenticación

- La autenticación en una api se base en...
 - **añadir un token** a la petición
 - param
 - cookie
 - cabecera
 - **comprobar** que sea correcto desde un **middleware**

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

function private(req, res, next) {
  if (req.query.token === 's3cr3t')
    return next()
  else
    res.status(401).send('Not authorized')
}

app.get('/private', private, (req, res) => {
  res.send('Top secret!')
}))

app.listen(3000)
```

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

```
function private(req, res, next) {
  if (req.query.token === 's3cr3t')
    return next()
  else
    res.status(401).send('Not authorized')
}
```

```
app.get('/private', private, (req, res) => {
  res.send('Top secret!')
}))
```

```
app.listen(3000)
```



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

function private(req, res, next) {
  if (req.query.token === 's3cr3t')
    return next()
  else
    res.status(401).send('Not authorized')
}

app.get('/private', private, (req, res) => {
  res.send('Top secret!')
})

app.listen(3000)
```

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

```
function private(req, res, next) {
  if (req.query.token === 's3cr3t')
    return next()
  else
    res.status(401).send('Not authorized')
}
```

```
app.get('/private', private, (req, res) => {
  res.send('Top secret!')
}))
```

```
app.listen(3000)
```

Estrategias de autenticación

- Para utilizar **cookies** para enviar el token tenemos dos alternativas:
 - **cookie-parser**
 - **cookie-session**

```
app.use(require('cookie-parser')('c00k13-s3cr3t'))
```

```
function private(req, res, next) {  
  req.signedCookies.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    res.cookie('auth', true, { signed: true })  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```

```
app.use(require('cookie-parser')('c00k13-s3cr3t'))
```

```
function private(req, res, next) {  
  req.signedCookies.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    res.cookie('auth', true, { signed: true })  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```

```
app.use(require('cookie-parser')('c00k13-s3cr3t'))
```

```
function private(req, res, next) {  
  req.signedCookies.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    res.cookie('auth', true, { signed: true })  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```

```
app.use(require('cookie-parser')('c00k13-s3cr3t'))
```

```
function private(req, res, next) {  
  req.signedCookies.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    res.cookie('auth', true, { signed: true })  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```

```
app.use(require('cookie-session')({ secret: 's3cr3t' })))
```

```
function private(req, res, next) {  
  req.session.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    req.session.auth = true  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```



```
app.use(require('cookie-session')({ secret: 's3cr3t' })))
```

```
function private(req, res, next) {  
  req.session.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    req.session.auth = true  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```

```
app.use(require('cookie-session')({ secret: 's3cr3t' })))
```

```
function private(req, res, next) {  
  req.session.auth ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    req.session.auth = true  
    res.send('Welcome!')  
  } else {  
    res.send('Invalid password...')  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.send('Top secret!'))
```

Estrategias de autenticación

- Para una **API**, es más conveniente generar un token y enviarlo con cada petición
 - un endpoint para generar un token
 - el token se añade como...
 - parámetro en la url
 - cabecera

```
let token = null

function private(req, res, next) {
  req.query.token === token ? next() : res.send(401)
}

app.get('/login', (req, res) => {
  if (req.query.pass === 's3cr3t') {
    token = Math.random().toString(36)
    res.json({ token })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private,
  (req, res) => res.json({ top: 'secret' })))
```

```
let token = null
```

```
function private(req, res, next) {  
  req.query.token === token ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    token = Math.random().toString(36)  
    res.json({ token })  
  } else {  
    res.json({ error: 'Invalid password' })  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.json({ top: 'secret' })))
```

```
let token = null

function private(req, res, next) {
  req.query.token === token ? next() : res.send(401)
}

app.get('/login', (req, res) => {
  if (req.query.pass === 's3cr3t') {
    token = Math.random().toString(36)
    res.json({ token })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private,
  (req, res) => res.json({ top: 'secret' })))
```

```
let token = null
```

```
function private(req, res, next) {  
  req.query.token === token ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    token = Math.random().toString(36)  
    res.json({ token })  
  } else {  
    res.json({ error: 'Invalid password' })  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.json({ top: 'secret' })))
```

```
let token = null

function private(req, res, next) {
  req.query.token === token ? next() : res.send(401)
}

app.get('/login', (req, res) => {
  if (req.query.pass === 's3cr3t') {
    token = Math.random().toString(36)
    res.json({ token })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private,
  (req, res) => res.json({ top: 'secret' })))
```



```
let token = null

function private(req, res, next) {
  req.query.token === token ? next() : res.send(401)
}

app.get('/login', (req, res) => {
  if (req.query.pass === 's3cr3t') {
    token = Math.random().toString(36)
    res.json({ token })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private,
  (req, res) => res.json({ top: 'secret' })))
```

Estrategias de autenticación

- Pasar el token en la URL no es buena idea
- Lo más recomendable: incluir el token en la cabecera
Authorization
- Más adecuada al protocolo HTTP

```
let token = null
```

```
function private(req, res, next) {  
  req.get('Authorization') === `Bearer ${token}` ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    token = Math.random().toString(36)  
    res.json({ token })  
  } else {  
    res.json({ error: 'Invalid password' })  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.json({ top: 'secret' })))
```

```
let token = null
```

```
function private(req, res, next) {  
  req.get('Authorization') === `Bearer ${token}` ? next() : res.send(401)  
}
```

```
app.get('/login', (req, res) => {  
  if (req.query.pass === 's3cr3t') {  
    token = Math.random().toString(36)  
    res.json({ token })  
  } else {  
    res.json({ error: 'Invalid password' })  
  }  
})
```

```
app.get('/private', private,  
  (req, res) => res.json({ top: 'secret' })))
```

Estrategias de autenticación

- Podemos usar el **middleware** de protección directamente en **app**
 - proteger todas las rutas que **se asignen posteriormente**
 - dividir en “parte pública” y “parte privada”

```
let token = null

function private(req, res, next) {
  req.get('Authorization') === `Bearer ${token}` ? next() : res.send(401)
}

// parte pública

app.get('/public', (req, res) => res.json({ public: 'knowledge' }))
app.get('/login', (req, res) => { /* código omitido */ })

app.use(private)

// parte privada

app.get('/private', (req, res) => res.json({ top: 'secret' })))
```

```
let token = null
```

```
function private(req, res, next) {  
  req.get('Authorization') === `Bearer ${token}` ? next() : res.send(401)  
}
```

```
// parte pública
```

```
app.get('/public', (req, res) => res.json({ public: 'knowledge' })))  
app.get('/login', (req, res) => { /* código omitido */ })
```

```
app.use(private)
```

```
// parte privada
```

```
app.get('/private', (req, res) => res.json({ top: 'secret' })))
```

```
let token = null

function private(req, res, next) {
  req.get('Authorization') === `Bearer ${token}` ? next() : res.send(401)
}

// parte pública

app.get('/public', (req, res) => res.json({ public: 'knowledge' }))
app.get('/login', (req, res) => { /* código omitido */ })

app.use(private)
```

```
// parte privada

app.get('/private', (req, res) => res.json({ top: 'secret' })))
```



```
let token = null

function private(req, res, next) {
  req.get('Authorization') === `Bearer ${token}` ? next() : res.send(401)
}

// parte pública

app.get('/public', (req, res) => res.json({ public: 'knowledge' }))

app.use(private)

// parte privada

app.get('/login', (req, res) => { /* código omitido */ })
app.get('/private', (req, res) => res.json({ top: 'secret' })))
```

Estrategias de autenticación

- **JSON Web Token**
 - <https://jwt.io>
 - **Datos JSON encriptados y firmados** utilizados como **token**
 - Compacto y **auto-contenido!**
 - Verificar sin almacenar

```
$ npm install jsonwebtoken
```

Estrategias de autenticación

- El **cliente** tiene que:
 - pedir un token autenticándose con user y password
 - guardar el token en localStorage
 - enviarlo en la cabecera *Authorization* con **cada request**

Estrategias de autenticación

- El **servidor** tiene que:
 - **generar y enviar** un jwt token cuando el cliente se autentique
 - con info suficiente para identificar al usuario
 - **comprobar** que la **firma** del token es correcta en cada petición

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}

app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god'}, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}
```

```
app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god' }, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})
```

```
app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}

app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god' }, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```



```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}

app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god'}, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}
```

```
app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god'}, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})
```

```
app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}

app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god'}, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}

app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god'}, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

```
const jwt = require('jsonwebtoken')

function private(req, res, next) {
  const token = req.get('Authorization').match(/Bearer (.*)$/)[1]
  jwt.verify(token, 's3cr3t', (err, decoded) => {
    if (err) return res.send(401)
    req.jwt = decoded
    next()
  })
}

app.get('/login', (req, res) => {
  if (req.query.pass === 'jarl') {
    res.json({ token: jwt.sign({ user: 'god'}, 's3cr3t') })
  } else {
    res.json({ error: 'Invalid password' })
  }
})

app.get('/private', private, (req, res) => res.json({ hello: req.jwt.user }))
app.listen(3000)
```

Ejercicio

- Escribe un **sistema de autenticación** basado en **JWT**
 - una ruta **/login** para obtener el token
 - que compruebe los credenciales contra una tabla en **mysql**
 - un **middleware** para *proteger* rutas

Addendum

- **NUNCA** guardes una contraseña en la base de datos
- Genera una **sal secreta** y genera un *hash* de password + sal
- Utiliza un algoritmo de hash que **no sea reversible**
- En la BBDD guarda solo **la sal y el hash**

Addendum

- Para validar un *user* y *password*:
 - busca el usuario en la BBDD
 - saca su sal y genera el hash(salt + password)
 - comprueba el hash contra el que hay en la BBDD
 - si coinciden: la contraseña era correcta
 - si no: la contraseña **no** era correcta

Estrategias de autenticación

- Para implementar **flujos más complejos...**
 - mejor utilizar alguna librería
 - [Passport.js](#) es bastante completa y poco intrusiva

Testing

app.js

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

```
app.get('/', (req, res) => res.send(200))
```

```
app.listen(3000)
```

tests/app.test.js

```
const request = require('supertest')
require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request('http://localhost:3000').get('/').then(response => {
      console.log(response)
      expect(response.statusCode).toBe(200)
    })
  })
})
```

tests/app.test.js

```
const request = require('supertest')  
require('../app.js')
```

```
describe('The root path', () => {  
  test('Status code should be 200', () => {  
    return request('http://localhost:3000').get('/').then(response => {  
      console.log(response)  
      expect(response.statusCode).toBe(200)  
    })  
  })  
})
```

tests/app.test.js

```
const request = require('supertest')
require('../app.js')
```

```
describe('The root path', () => {
  test('Status code should be 200', () => {
    return request('http://localhost:3000').get('/').then(response => {
      console.log(response)
      expect(response.statusCode).toBe(200)
    })
  })
})
```

tests/app.test.js

```
const request = require('supertest')
require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request('http://localhost:3000').get('/').then(response => {
      console.log(response)
      expect(response.statusCode).toBe(200)
    })
  })
})
```

tests/app.test.js

```
const request = require('supertest')
require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request('http://localhost:3000').get('/').then(response => {
      console.log(response)
      expect(response.statusCode).toBe(200)
    })
  })
})
```


tests/app.test.js

```
const request = require('supertest')
require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request('http://localhost:3000').get('/').then(response => {
      console.log(response)
      expect(response.statusCode).toBe(200)
    })
  })
})
```

tests/app.test.js

```
const request = require('supertest')
require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request('http://localhost:3000').get('/').then(response => {
      console.log(response)
      expect(response.statusCode).toBe(200)
    })
  })
})
```

Testing

- Si ejecuto el test del ejemplo...
 - veo que pasa el caso de test
 - pero **no termina**
 - **jest** se queda abierto...
 - **¿POR QUÉ?**

Testing

- exportar una referencia al **servidor**
 - para saber dónde está escuchando
 - para poder apagarlo al terminar
- exportar una referencia a la **db**
 - para poder inspeccionar los datos
 - para poder terminar la conexión al terminar

app.js

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

```
app.get('/', (req, res) => res.send(200))
```

```
const server = app.listen(3000)
```

```
module.exports = { server }
```

app.js

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

```
app.get('/', (req, res) => res.send(200))
```

```
const server = app.listen(3000)
```

```
module.exports = { server }
```

tests/app.test.js

```
const request = require('supertest')
const { server } = require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request(server).get('/').then(response => {
      expect(response.statusCode).toBe(200)
    })
  })
})

afterAll(() => {
  server.close()
})
```

tests/app.test.js

```
const request = require('supertest')  
const { server } = require('../app.js')
```

```
describe('The root path', () => {  
  test('Status code should be 200', () => {  
    return request(server).get('/').then(response => {  
      expect(response.statusCode).toBe(200)  
    })  
  })  
})  
  
afterAll(() => {  
  server.close()  
})
```


tests/app.test.js

```
const request = require('supertest')
const { server } = require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request(server).get('/').then(response => {
      expect(response.statusCode).toBe(200)
    })
  })
})

afterAll(() => {
  server.close()
})
```

tests/app.test.js

```
const request = require('supertest')
const { server } = require('../app.js')

describe('The root path', () => {
  test('Status code should be 200', () => {
    return request(server).get('/').then(response => {
      expect(response.statusCode).toBe(200)
    })
  })
})
```

```
afterAll(() => {
  server.close()
})
```

Testing

- Para comprobar la **respuesta** podemos usar...
 - `response.statusCode`
 - `response.body`
 - `response.header['Header-Name']`

Testing con MySQL

Testing con MySQL

- Necesitamos...
 - tener un **entorno de testing** con una base de datos aislada
 - configurar la conexión para que utilice esa base de datos cuando ejecutamos el programa en **modo test**

config.js

```
const testConfig = {
  db: { host: 'localhost', user: 'root', database: 'test' }
}

const devConfig = {
  db: { host: 'localhost', user: 'root', database: 'devel' }
}

const productionConfig = { db: { /*... */ } }

const selectConfig = () => {
  switch (process.env.NODE_ENV) {
    case 'production': return productionConfig;
    case 'test': return testConfig;
    default: return devConfig;
  }
}

module.exports = selectConfig()
```

config.js

```
const testConfig = {
  db: { host: 'localhost', user: 'root', database: 'test' }
}

const devConfig = {
  db: { host: 'localhost', user: 'root', database: 'devel' }
}

const productionConfig = { db: { /*... */ } }

const selectConfig = () => {
  switch (process.env.NODE_ENV) {
    case 'production': return productionConfig;
    case 'test': return testConfig;
    default: return devConfig;
  }
}

module.exports = selectConfig()
```

config.js

```
const testConfig = {
  db: { host: 'localhost', user: 'root', database: 'test' }
}

const devConfig = {
  db: { host: 'localhost', user: 'root', database: 'devel' }
}

const productionConfig = { db: { /*... */ } }

const selectConfig = () => {
  switch (process.env.NODE_ENV) {
    case 'production': return productionConfig;
    case 'test': return testConfig;
    default: return devConfig;
  }
}

module.exports = selectConfig()
```


app.js

```
const config = require('./config.js')
const db = mysql.createConnection(config.db)

app.use(express.json())

app.post('/users', (req, res) => {
  db.query(
    'INSERT INTO users VALUES (null, ?)', [req.body.name],
    (err) => res.send(err ? 500 : 200)
  )
})

const server = app.listen(3000)

module.exports = { server, db }
```

app.js

```
const config = require('./config.js')
```

```
const db = mysql.createConnection(config.db)
```

```
app.use(express.json())
```

```
app.post('/users', (req, res) => {  
  db.query(  
    'INSERT INTO users VALUES (null, ?)', [req.body.name],  
    (err) => res.send(err ? 500 : 200)  
  )  
})
```

```
const server = app.listen(3000)
```

```
module.exports = { server, db }
```

app.js

```
const config = require('./config.js')
const db = mysql.createConnection(config.db)

app.use(express.json())

app.post('/users', (req, res) => {
  db.query(
    'INSERT INTO users VALUES (null, ?)', [req.body.name],
    (err) => res.send(err ? 500 : 200)
  )
})

const server = app.listen(3000)

module.exports = { server, db }
```

app.js

```
const config = require('./config.js')
const db = mysql.createConnection(config.db)

app.use(express.json())

app.post('/users', (req, res) => {
  db.query(
    'INSERT INTO users VALUES (null, ?)', [req.body.name],
    (err) => res.send(err ? 500 : 200)
  )
})

const server = app.listen(3000)

module.exports = { server, db }
```

Testing con MySQL

- El procedimiento habitual será
 - **borrar** la base de datos
 - **preparar** el entorno para el test
 - hacer la **petición**
 - consultar que el **estado de la data** sea el esperado

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```

test/app.test.js

```
const request = require('supertest')
```

```
const { server, db } = require('../app.js')
```

```
describe('The root path', () => {
```

```
  beforeEach(done => db.query('DELETE FROM users', done))
```

```
  test('Should create a new user', (done) => {
```

```
    const userData = { name: 'Homer' }
```

```
    request(server).post('/users').send(userData).then(response => {
```

```
      expect(response.statusCode).toBe(200)
```

```
      db.query('SELECT * FROM users', (err, rows) => {
```

```
        if (err) return done(err)
```

```
        expect(rows.length).toBe(1)
```

```
        expect(rows[0].name).toBe('Homer')
```

```
        done()
```

```
      })
```

```
    })
```

```
  })
```

```
}
```

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {
  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```


test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    const userData = { name: 'Homer' }
    request(server).post('/users').send(userData).then(response => {
      expect(response.statusCode).toBe(200)
      db.query('SELECT * FROM users', (err, rows) => {
        if (err) return done(err)
        expect(rows.length).toBe(1)
        expect(rows[0].name).toBe('Homer')
        done()
      })
    })
  })
})
```

test/app.test.js

```
const request = require('supertest')
const { server, db } = require('../app.js')

describe('The root path', () => {

  beforeEach(done => db.query('DELETE FROM users', done))

  test('Should create a new user', (done) => {
    /* código omitido */
  })
})

afterAll(() => {
  server.close()
  db.end()
})
```

Ejercicio

- Escribe una **API REST** para gestionar **listas de tareas...**
 - **registro y login** de usuarios
 - cada usuario tiene listas, cada lista tiene tareas
 - <https://app.swaggerhub.com/apis/redradix7/todolists>
 - utilizando **TDD**
 - *(reutiliza libremente código de ejercicios anteriores)*