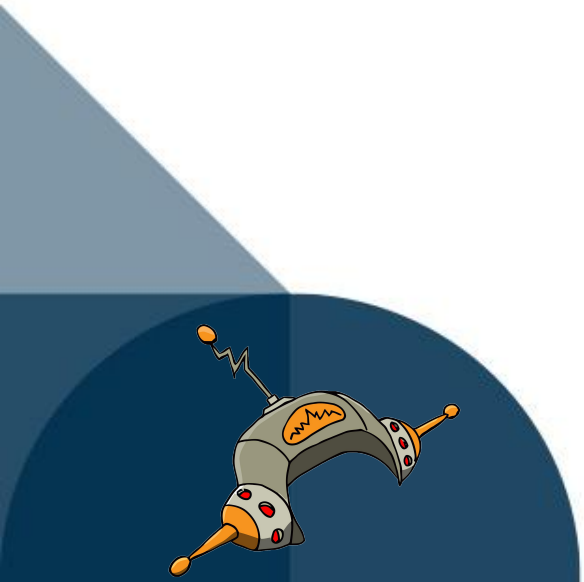

hapi

Introducción

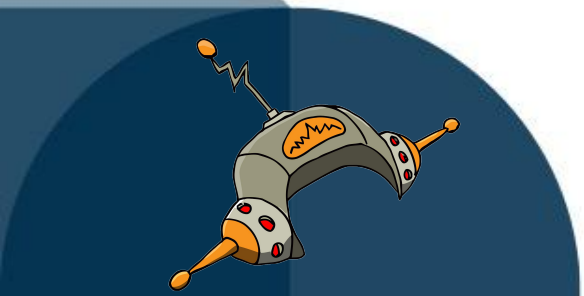
Qué es Hapi?

- Framework de Node.js para crear aplicaciones y servicios web
- Diseñado pensando en aplicativos modularizados
- Filosofía que contempla la separación de la configuración y la lógica de negocio
- Opinionado



Principales usos

- Aplicativos web
- APIs REST
- APIs GraphQL
- Proxies HTTP
- Integrador de múltiples backends



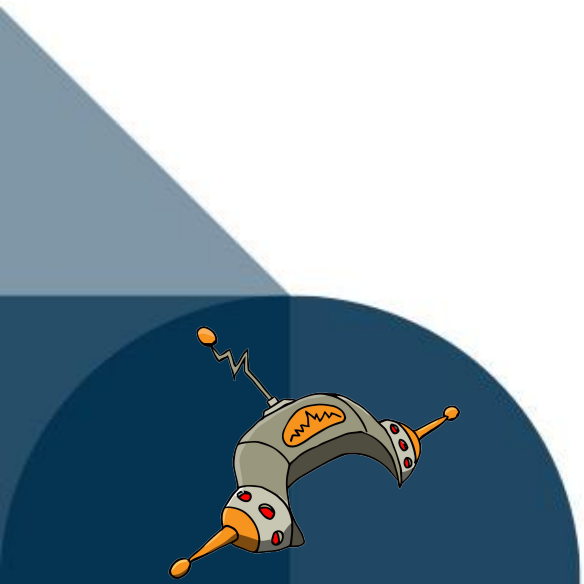
Prerrequisitos

- Conocimientos básicos de Node.js
- Conocimientos básicos de aplicativos web
- Deseable experiencia mínima con Express
- Deseable experiencia con manejo de asincronía en Node.js con Async/Await
- Deseable estar familiarizado con Firebase

Breve historia y estado actual

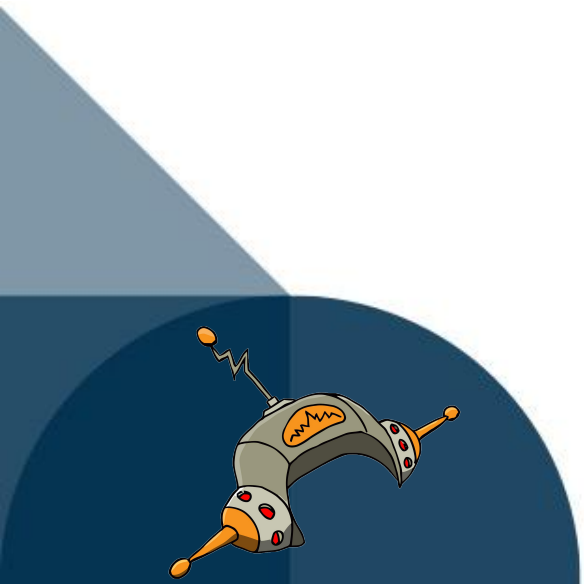
Creación

- Creado por Eran Hammer - Autor líder y editor de la especificación Oauth
- Nombre derivado de HTTP API server
- Creado en Walmart labs por el equipo de mobile
- Solución a los picos de alto tráfico en el Black Friday



Evolución

- Creado inicialmente como un middleware de Express
- Reemplazo de Express con Director
- Planteamiento filosófico de que es mejor tener configuración que código
- Recientemente rediseñado para usar los últimos avances del core de Node.js



Versionamiento

- Uso de Semantic versioning (Semver)
- Changelog automatizado y publicado en el sitio
- 17 versiones major hasta ahora
- Versión actual 17.5.5

Conceptos principales

Servidor

Unidad básica y principal del framework

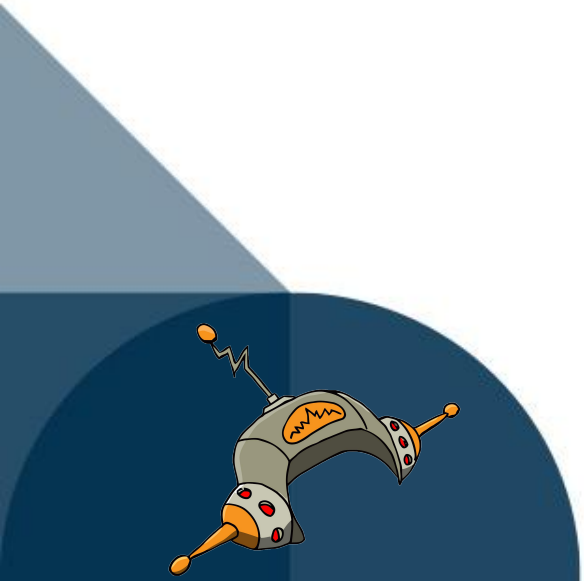
```
const Hapi = require('hapi')


const server = Hapi.server({
    port: 3000,
    host: 'localhost'
})
```

Ruta

Define puntos de interacción para el aplicativo

```
server.route({
    method: 'GET',
    path: '/hola',
    handler: (request, h) => {
        return 'Hola Mundo'
    }
})
```



Ruta > Method

Método HTTP por el cual un Browser o un cliente debe acceder a la ruta

```
server.route({
    method: 'GET',
    path: '/hola',
    handler: (request, h) => {
        return 'Hola Mundo'
    }
})
```

Ruta > Path

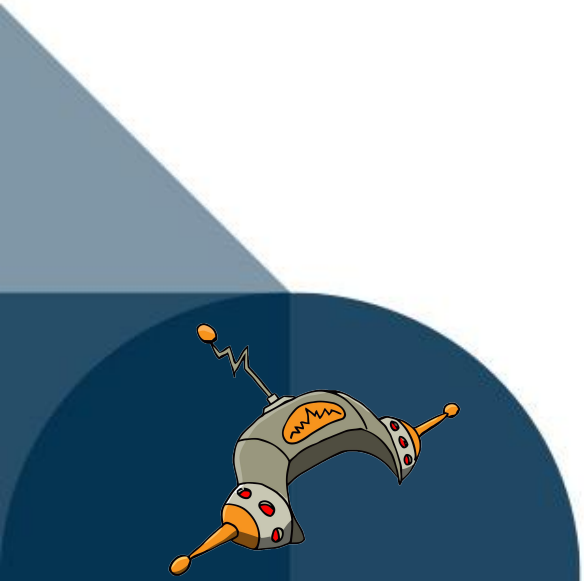
Define parte de la URL que será accedida por el cliente

```
server.route({
    method: 'GET',
    path: '/hola',
    handler: (request, h) => {
        return 'Hola Mundo'
    }
})
```

Ruta > Handler

Función que se ejecuta cuando se accede a una ruta

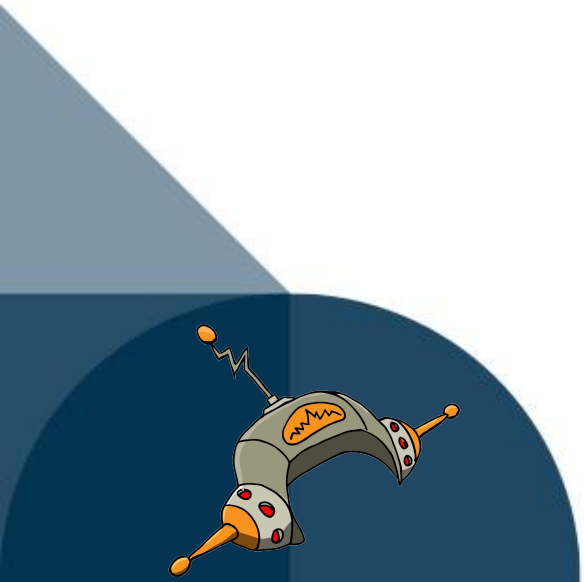
```
server.route({
    method: 'GET',
    path: '/hola',
    handler: (request, h) => {
        return 'Hola Mundo'
    }
})
```



Response

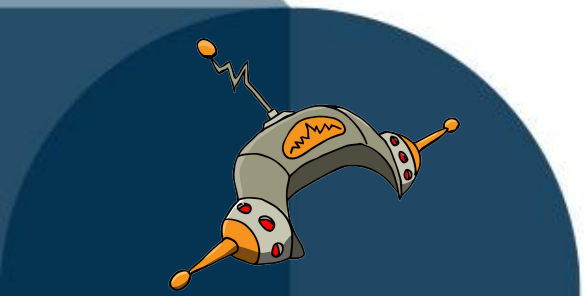
El objeto h

Segundo argumento del handler, es una colección de utilidades y propiedades relativas a enviar información de respuesta



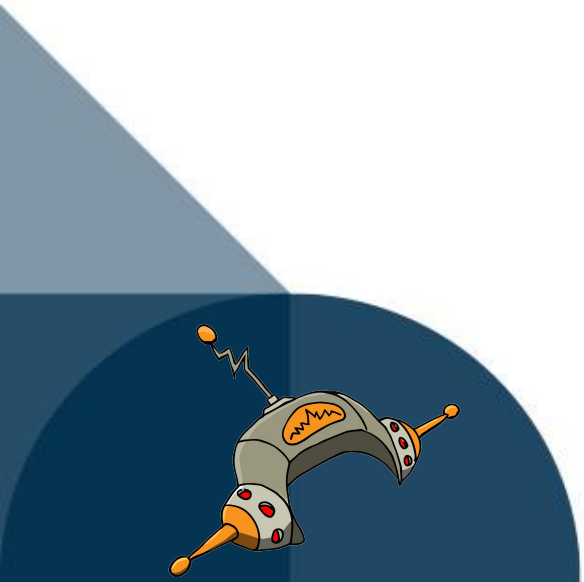
El objeto h > Métodos básicos

- h.response - Crea un objeto de respuesta
- h.redirect - Redirecciona una petición



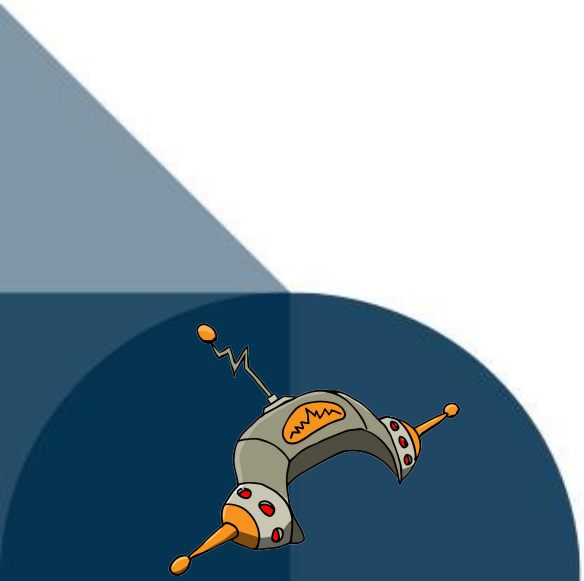
El objeto Response

Generado por el método **h.response**, permite definir las propiedades de la respuesta al Browser o al cliente que realizó la petición



El objeto Response > Métodos básicos

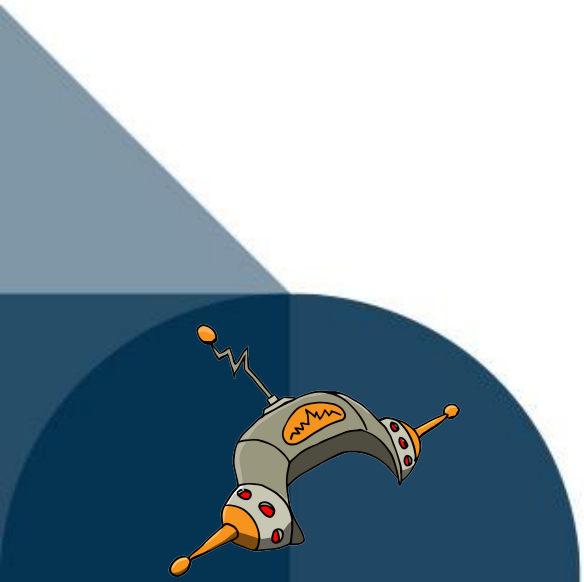
- response.header - Configura un encabezado en la respuesta
- response.type - Permite definir el tipo mime de la respuesta
- response.code - permite definir el código de estado de la respuesta (200, 300, 400….)



Request

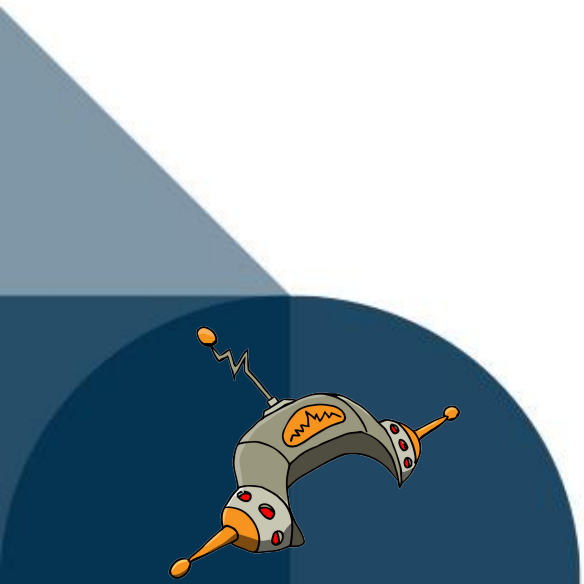
El objeto Request

Primer argumento del handler, colección de propiedades relativas a la petición del Browser o el cliente



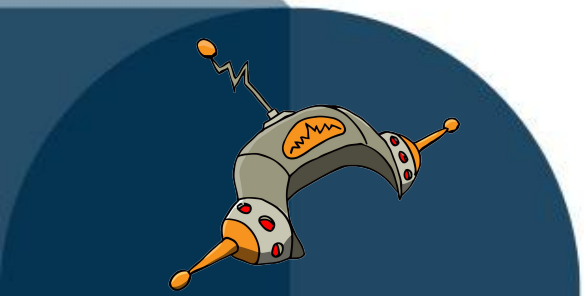
El objeto Request > Propiedades

- request.path - Ruta accedida
- request.method - Método usado en la petición
- request.params - Parámetros incluidos en la url
- request.query - Parámetros incluidos en el query de la URL
- request.payload - Parámetros recibidos en métodos POST y PUT



El objeto Request > Ciclo de vida

- onRequest
- onPreAuth
- onCredentials
- onPostAuth
- onPreHandler
- onPostHandler
- onPreResponse

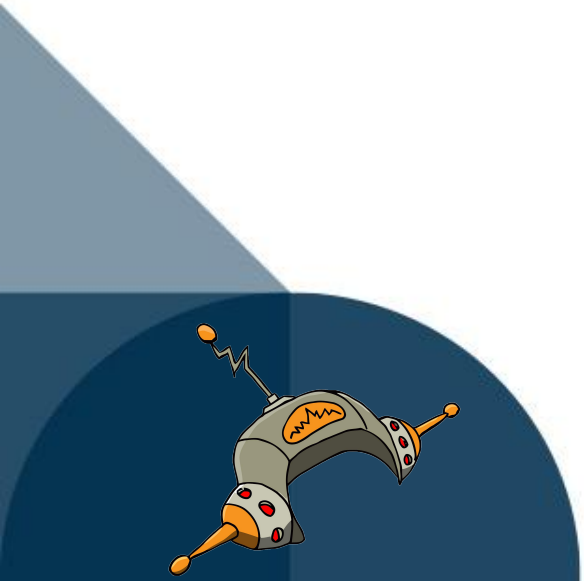


Plugins

Estructura de un plugin

Objeto que define funcionalidades adicionales a un servidor externo

```
const plugin = {
    name: 'miPlugin',
    version: '1.0.0',
    register: function (server, options) {
        ...
    }
}
```



Estructura de un plugin > Nombre

Define el nombre con el que se identificará el plugin

```
const plugin = {
    name: 'miPlugin',
    version: '1.0.0',
    register: function (server, options) {
        ...
    }
}
```

Estructura de un plugin > Versión

Dato opcional e informativo que puede ser usado por otros plugins

```
const plugin = {
    name: 'miPlugin',
    version: '1.0.0',
    register: function (server, options) {
        ...
    }
}
```

Estructura de un plugin > Registro

Función que recibirá el servidor inyectado y los argumentos iniciales

```
const plugin = {
    name: 'miPlugin',
    version: '1.0.0',
    register: function (server, options) {
        ...
    }
}
```