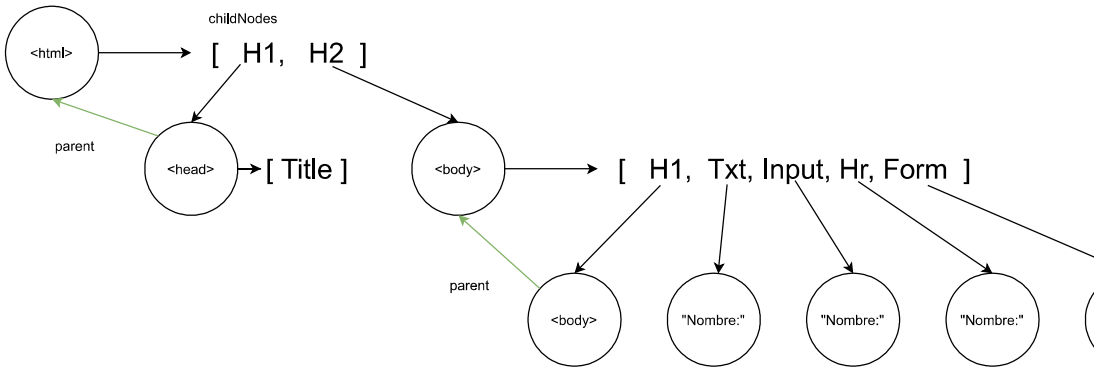
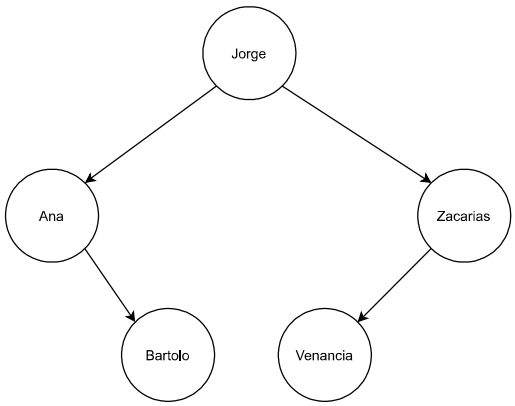
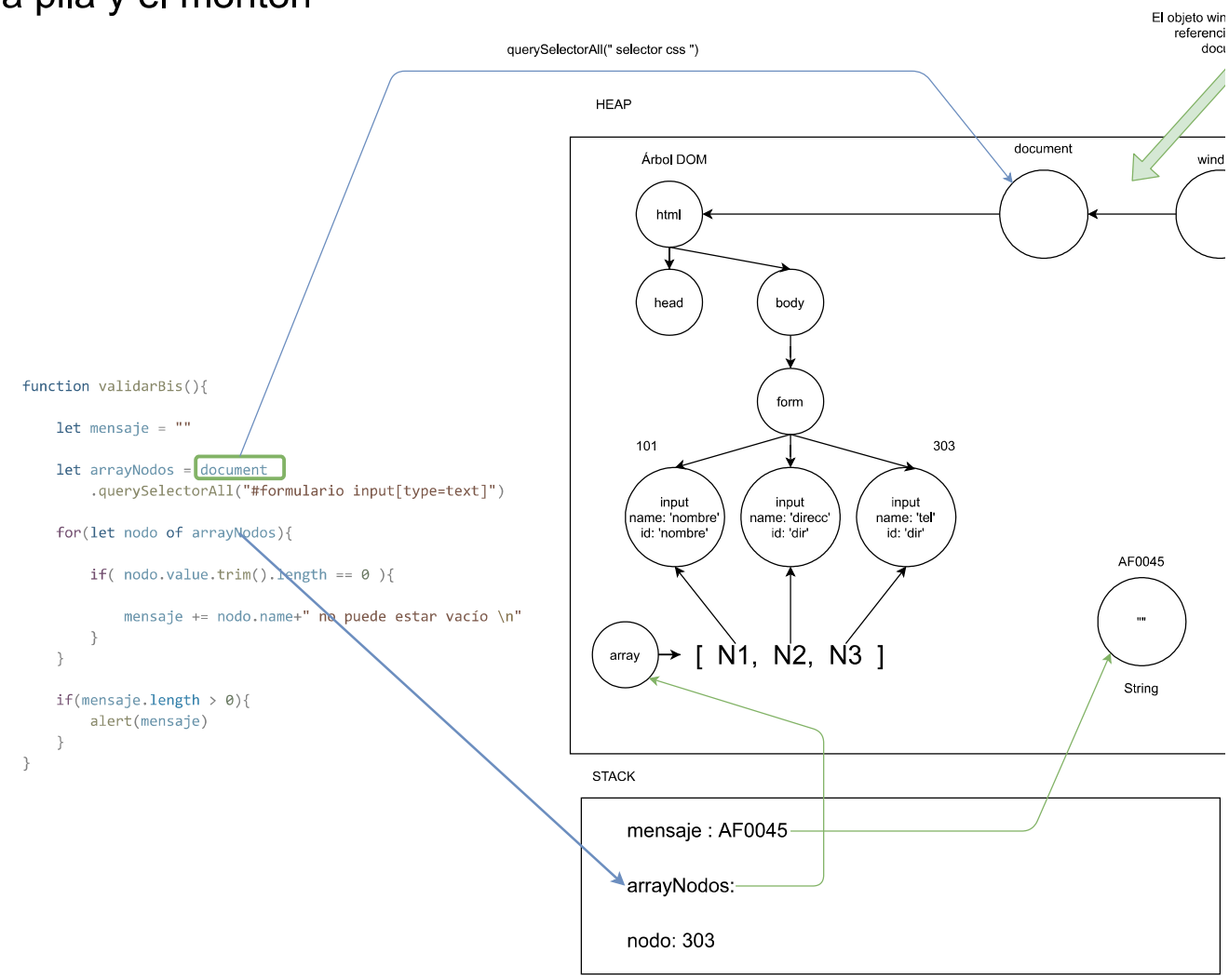
El árbol DOM

```
{
   type : "input",
   childNodes: [ nodos ],
   parentNode: nodo

}
```

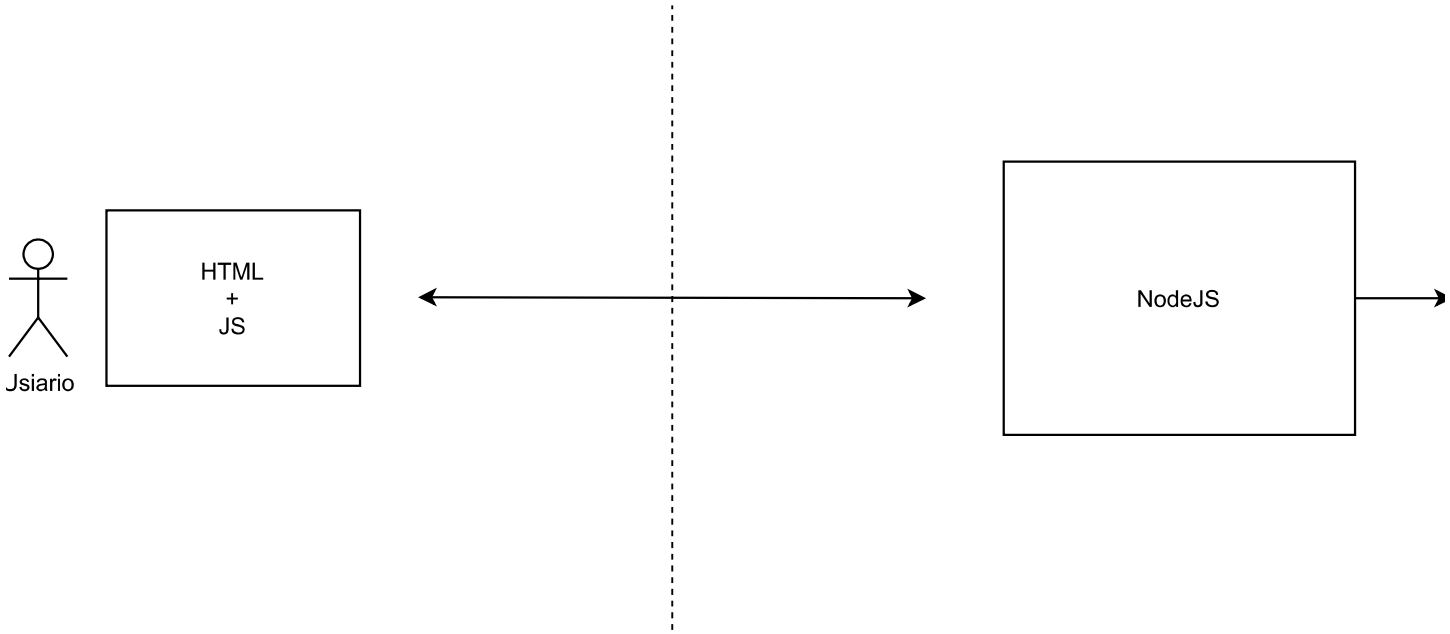




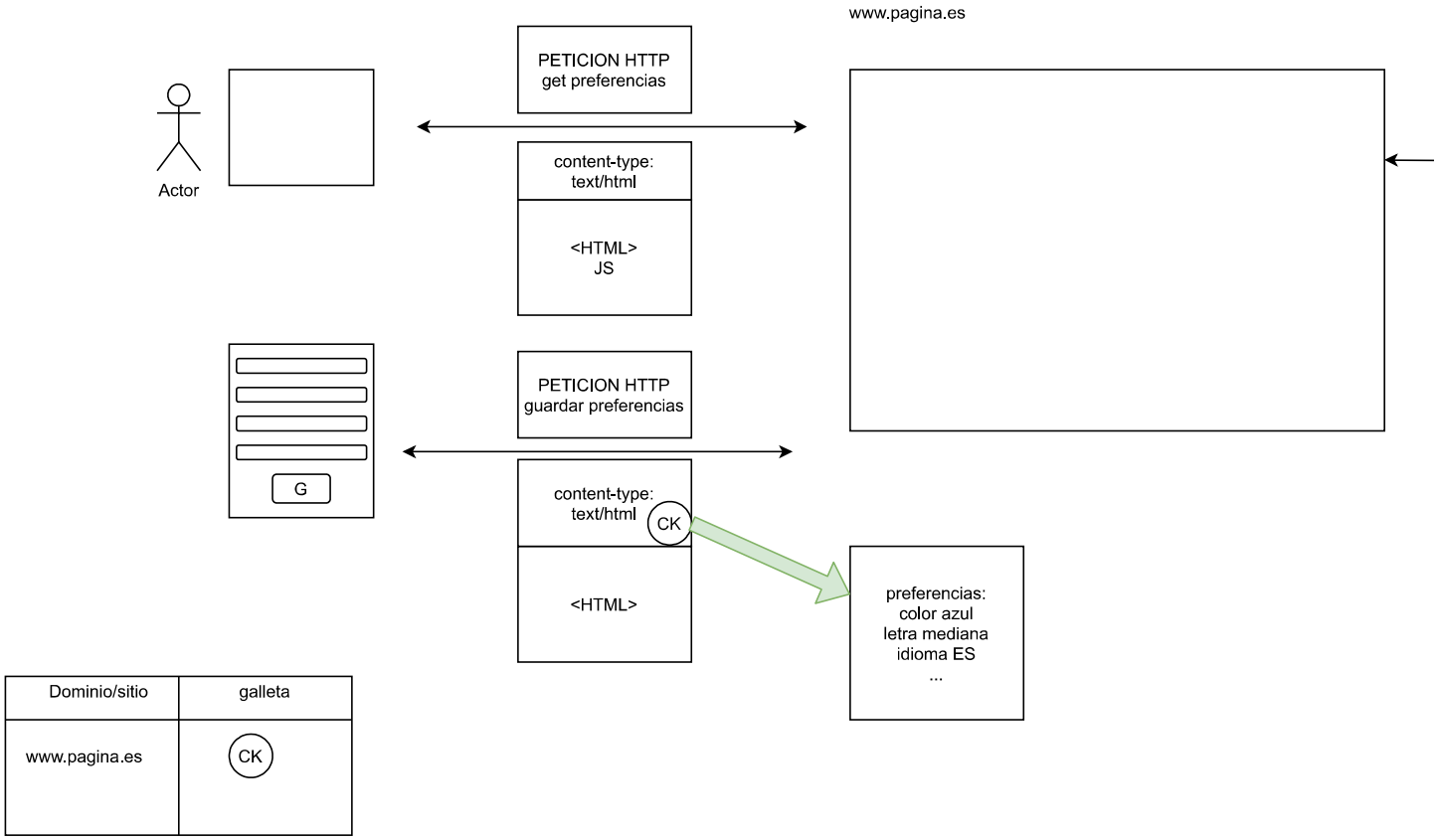
La pila y el montón




```
function validarBis(){

    let mensaje = ""

    let arrayNodos = document
        .querySelectorAll("#formulario input[type=text]")

    for(let nodo of arrayNodos){

        if( nodo.value.trim().length == 0 ){

            mensaje += nodo.name+" no puede estar vacío \n"
        }
    }

    if(mensaje.length > 0){
        alert(mensaje)
    }
}
```

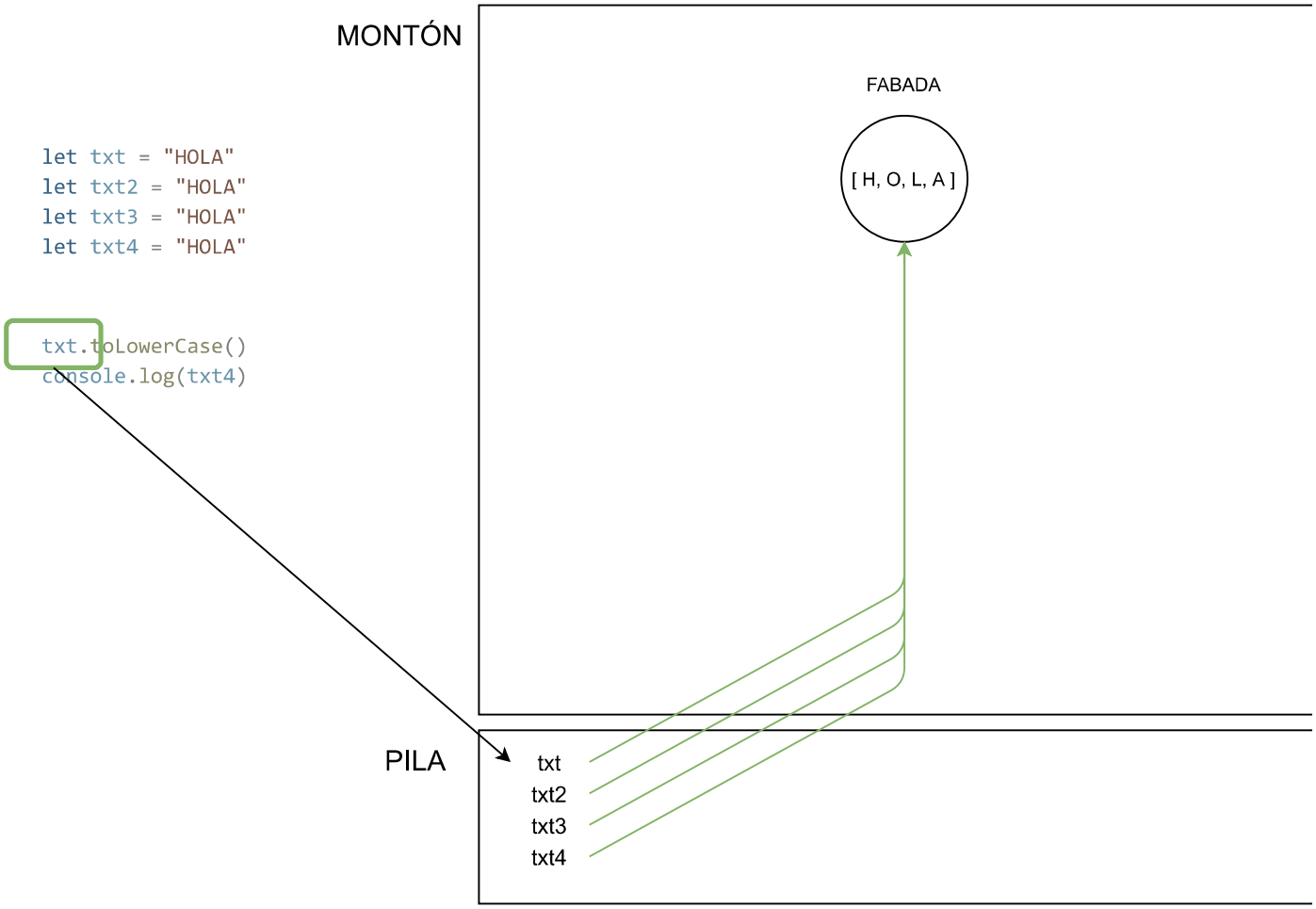

HTML
+
JS
Jsiario
NodeJS

Utilizando cookies para almacenar en el navegador
informacion recibida del servidor

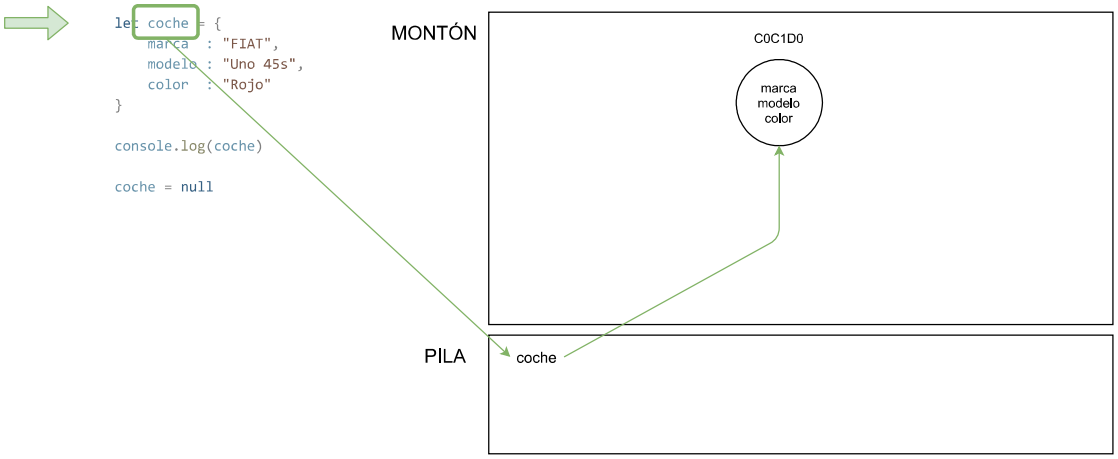


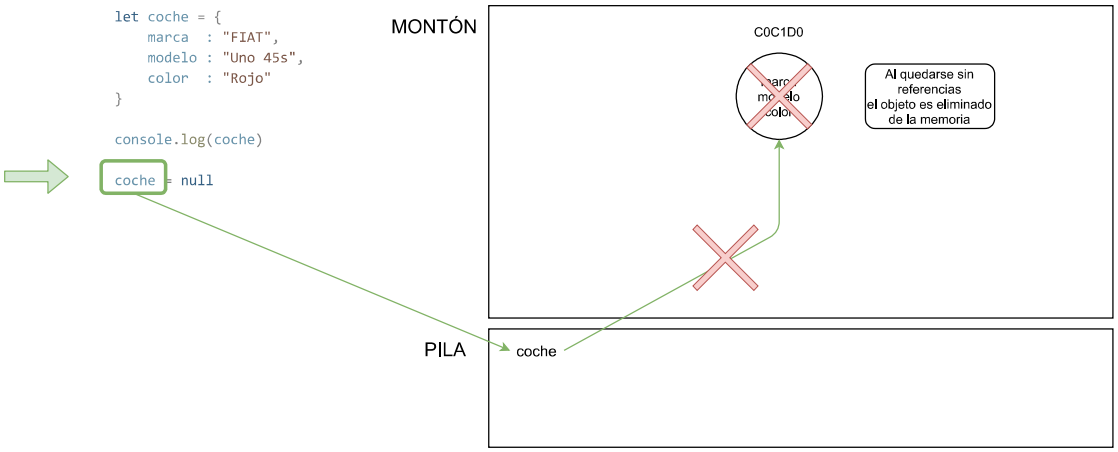
Dominio/sitio	galleta
www.pagina.es	CK

La vida secreta de los string



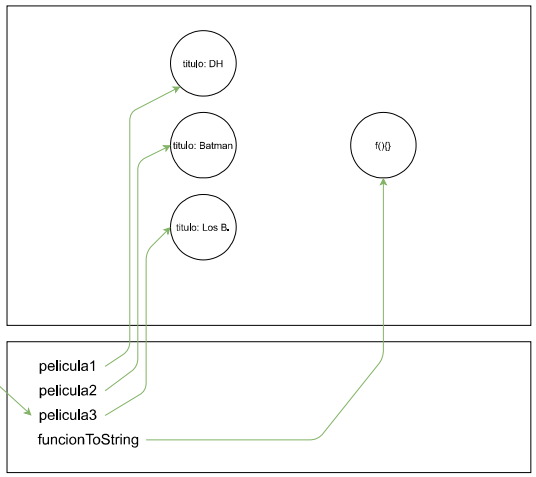
Recolección de basura en JS



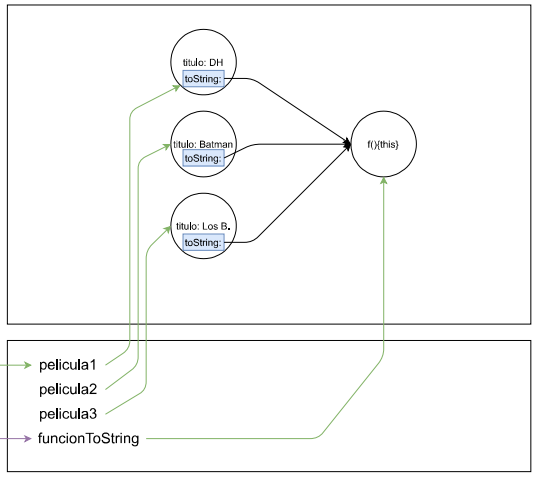


Objetos JS que comparten una función

```
console.log("======================================")
let pelicula1 = { titulo:"Die Hard" }
let pelicula2 = { titulo:"Batman (1989)" }
let pelicula3 = { titulo:"Los Bingueros" }

let funcionToString = function(){
    console.log("Gracias por inventar JS:",this)
    return "Titulo: "+this.titulo
}

pelicula1.toString = funcionToString
pelicula2.toString = funcionToString
pelicula3.toString = funcionToString
```



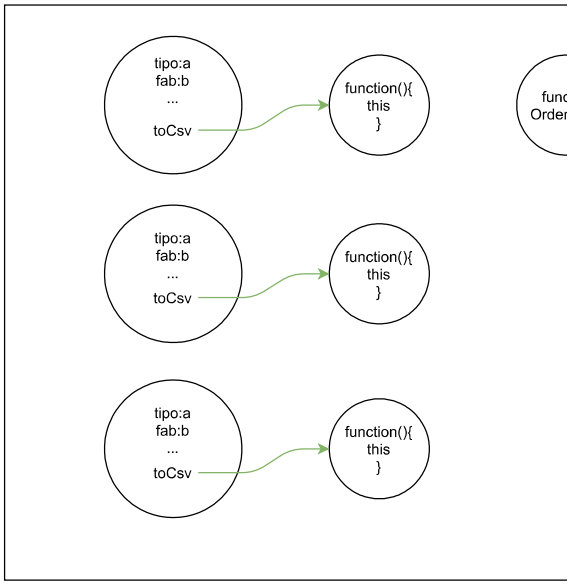

```
console.log("======================================")
let pelicula1 = { titulo:"Die Hard" }
let pelicula2 = { titulo:"Batman (1989)" }
let pelicula3 = { titulo:"Los Bingueros" }

let funcionToString = function(){
    console.log("Gracias por inventar JS:",this)
    return "Titulo: "+this.titulo
}

pelicula1.toString = funcionToString
pelicula2.toString = funcionToString
pelicula3.toString = funcionToString
```




```
console.log(pelicula1.toString()) //this será 'pelicula1'
console.log(pelicula2.toString()) //this será 'pelicula2'
console.log(pelicula3.toString()) //this será 'pelicula3'
```

Si declaramos las funciones dentro del prototipo cada objeto tendrá su propia copia de la funcion. Esto está muy mal

```
console.log("=========================================================")
function Ordenador(tipo, fabricante, procesador, memoria, discoDuro){
    this.tipo       = tipo
    this.fabricante = fabricante
    this.procesador = procesador
    this.memoria    = memoria
    this.discoDuro  = discoDuro
    this.toCSV = function(){
        return `${this.tipo},${this.fabricante},${this.procesador},
                ${this.memoria},${this.discoDuro}`
    }
}

let ordenador1 = new Ordenador("PC sobremesa","IBM","8088",512,"10Mb")
let ordenador2 = new Ordenador("Microordenador","AMSTRAD","z80",64,"Cassette")
let ordenador3 = new Ordenador("Portatil","Apple","Powerpc",1024,"250Mb")
```



ordenador1

ordenador2

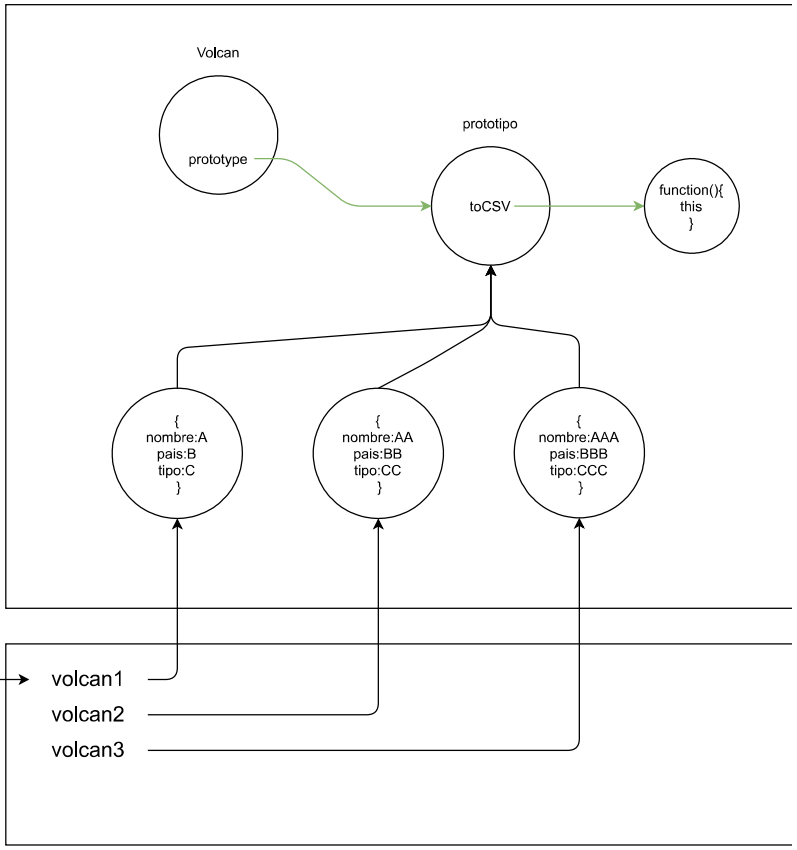
ordenador3

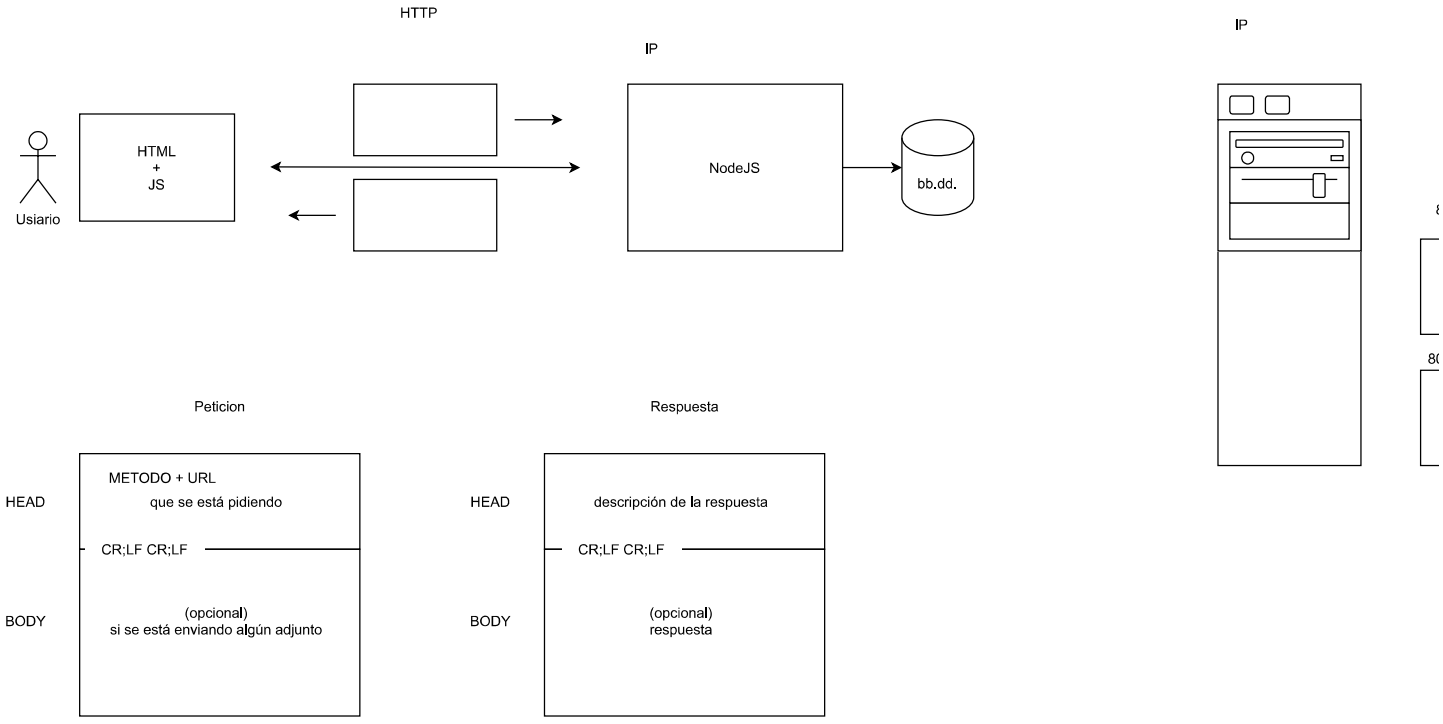
Añadiendo propiedades al prototipo

```
function Volcan(nombre,pais,tipo){
    this.nombre = nombre
    this.pais   = pais
    this.tipo   = tipo
}

Volcan.prototype.toCSV = function(){
    return `${this.nombre},${this.pais},${this.tipo}`
}
```

```
let volcan1 = new Volcan("Stromboli","Italia","Estromboliano")
let volcan2 = new Volcan("Mauna Kea","USA","Hawaiano")
let volcan3 = new Volcan("Misti","Perú","Estratovolcan")
```

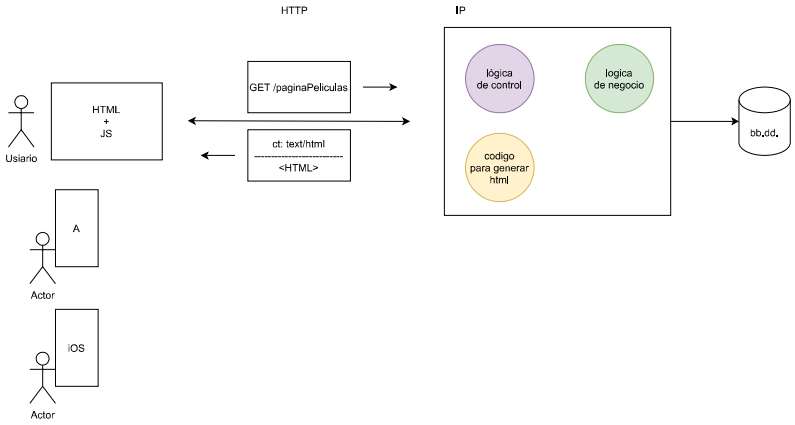
```
console.log(volcan1.toCSV())
console.log(volcan2.toCSV())
console.log(volcan3.toCSV())
```

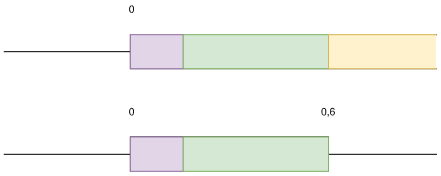



HTTP
IP
IP
HTML
+
JS
Usiario
NodeJS
bb.dd.

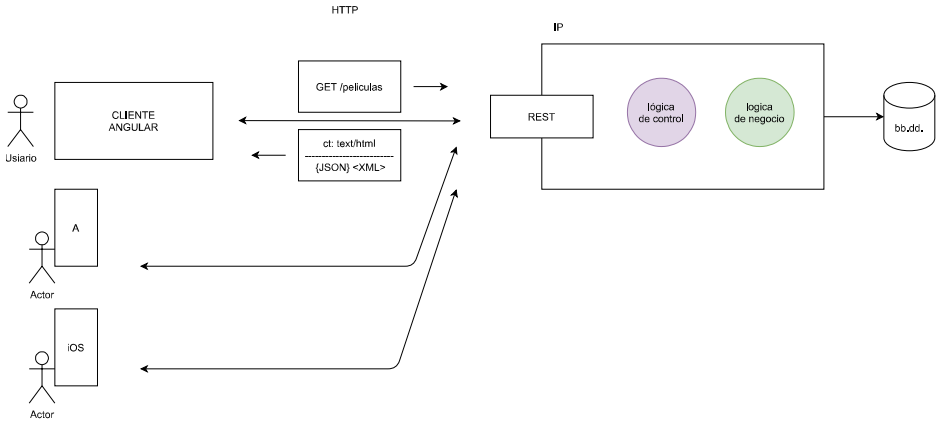
Peticion
Respuesta
METODO + URL
HEAD
que se está pidiendo
CR;LF CR;LF
BODY
(opcional)
si se está enviando algún adjunto
HEAD
descripción de la respuesta
CR;LF CR;LF
BODY
(opcional)
respuesta

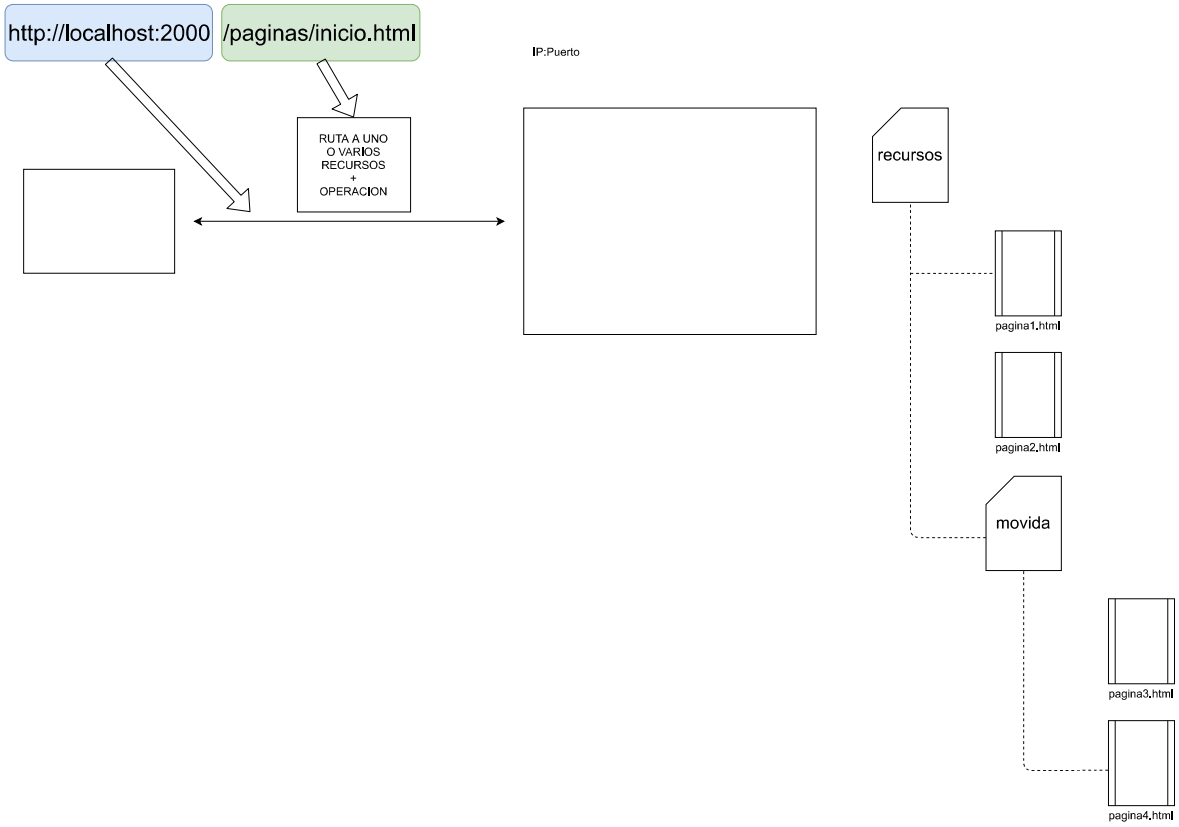
Aplicación web en el servidor





Arquitectura cliente-servidor



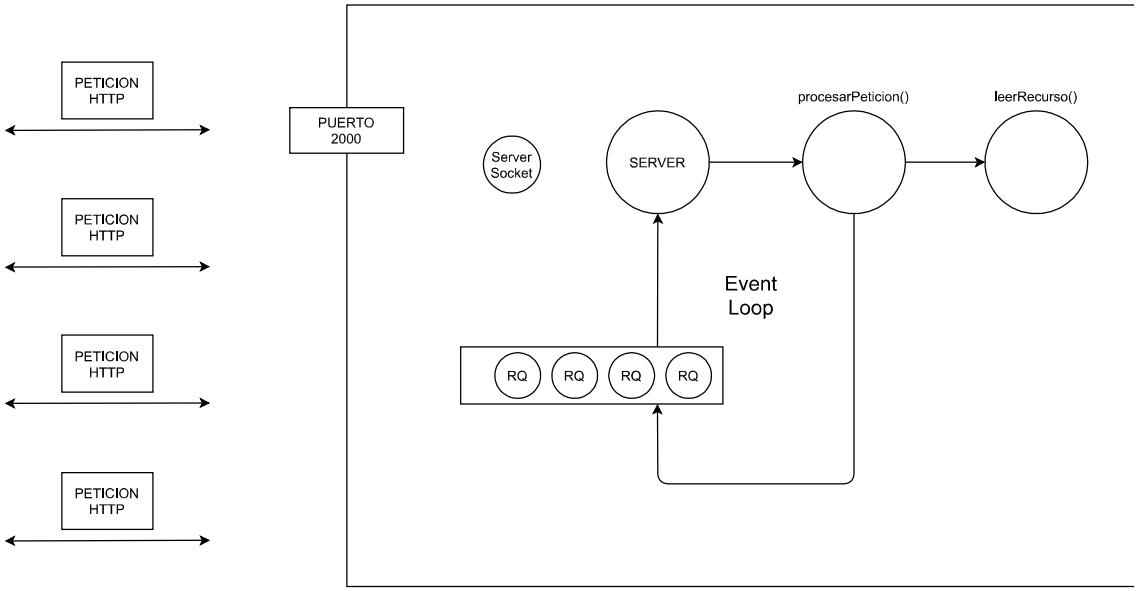


PETICIONES HTTP

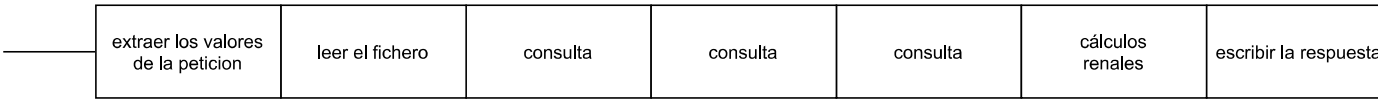
METODO + RUTA A UNO O VARIOS RECURSOS

GET /recursos/movida/pagina3.html

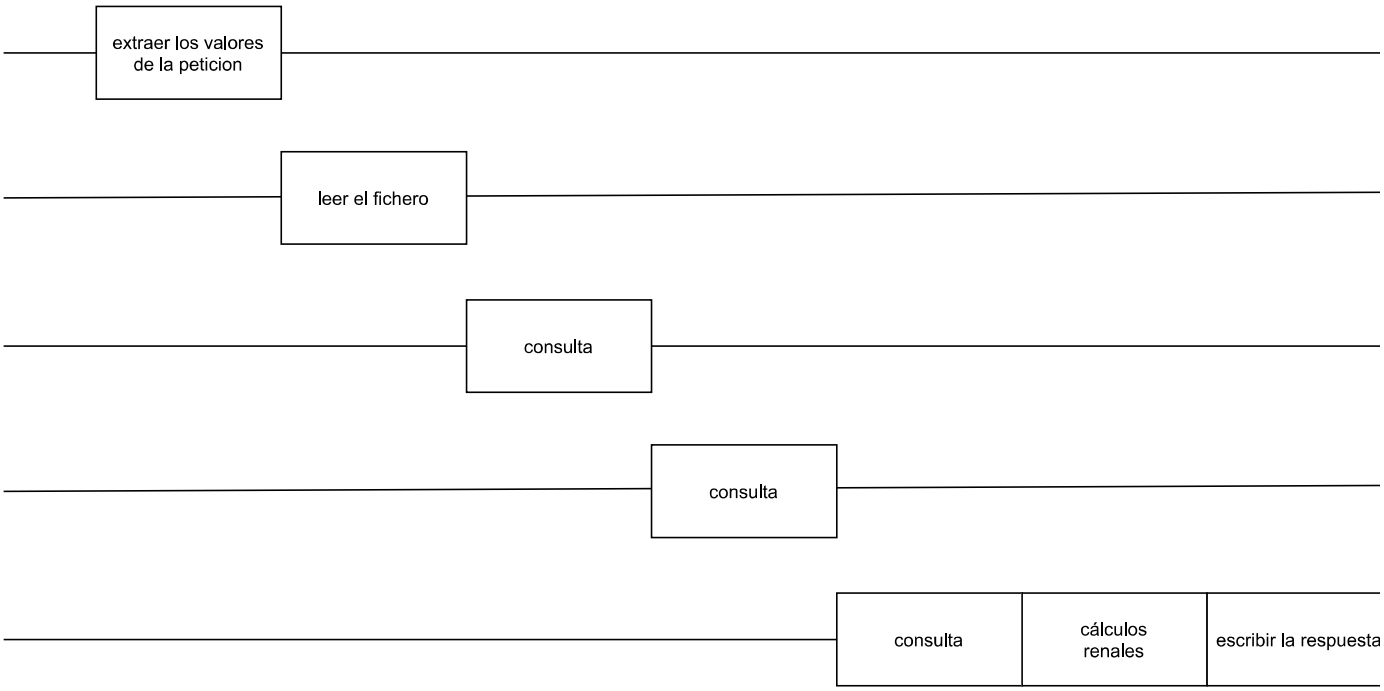
El event loop



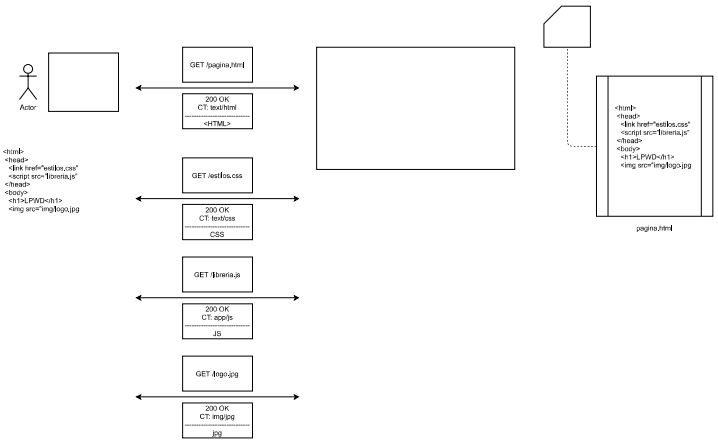
Un hilo para cada petición: Apache, IIS, JEE

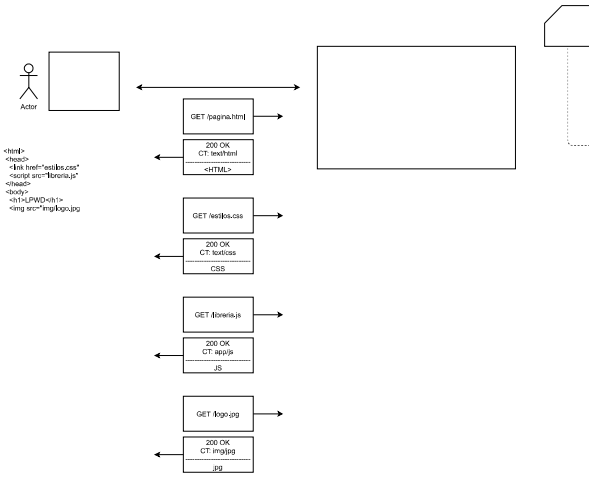


Node:

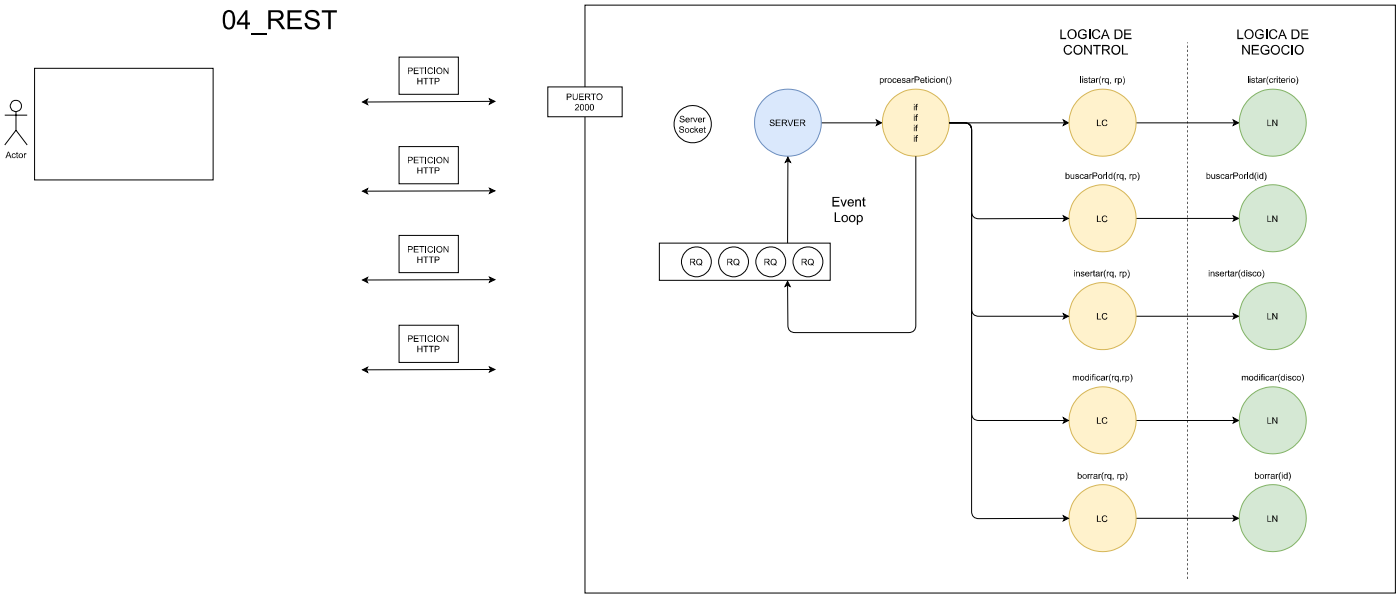


HTTP 2.0 y la multiplexación de los sockets

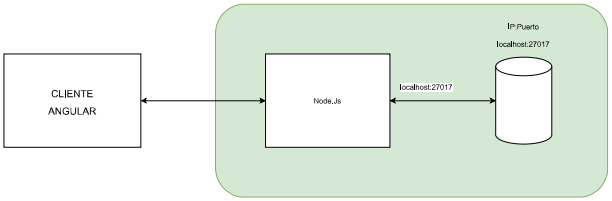




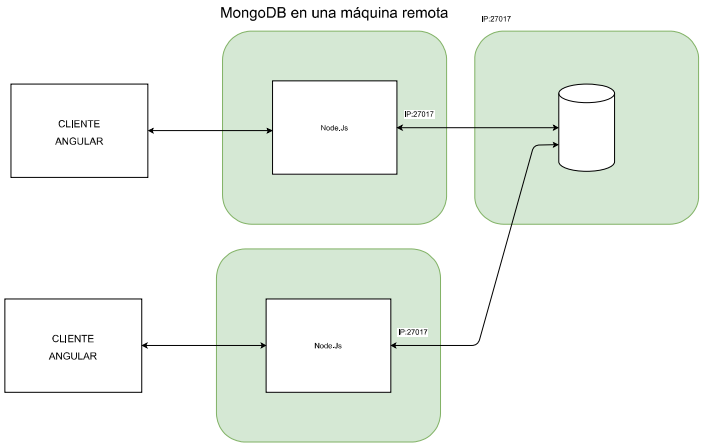
04_REST


Actor
PETICION HTTP
PETICION HTTP
PETICION HTTP
PETICION HTTP
PUERTO 2000
Server Socket
SERVER
procesarPeticion()
if
if
if
if
Event Loop
RQ
RQ
RQ
RQ
LOGICA DE CONTROL
LOGICA DE NEGOCIO
listar(rq, rp)
LC
listar(criterio)
LN
buscarPorId(rq, rp)
LC
buscarPorId(id)
LN
insertar(rq, rp)
LC
insertar(disco)
LN
modificar(rq,rp)
LC
modificar(disco)
LN
borrar(rq, rp)
LC
borrar(id)
LN

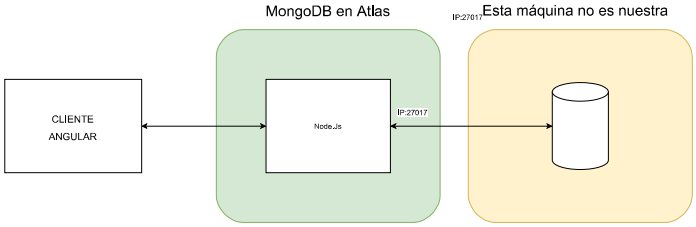
MongoDB en localhost



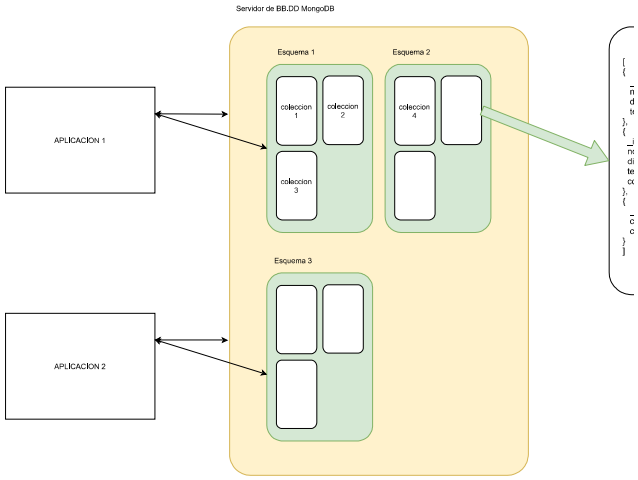
MongoDB en una máquina remota



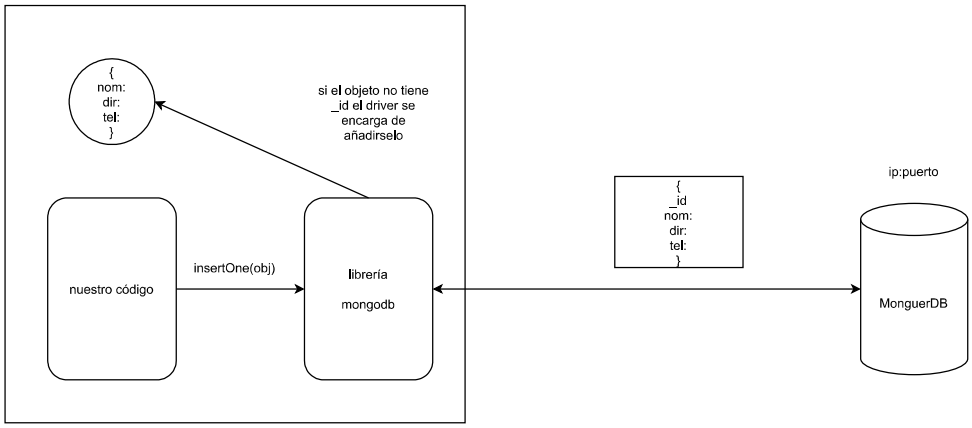
MongoDB en Atlas



Mitocondrias en una célula

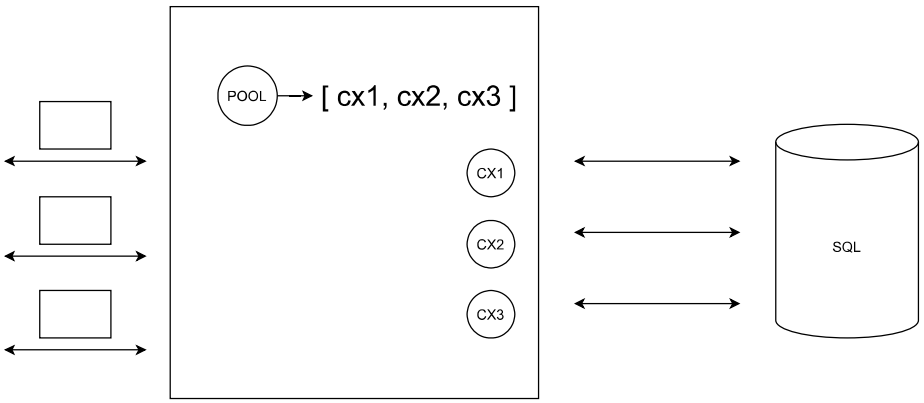


Con MongoDB es el driver el que asigna valor a _id si no está presente

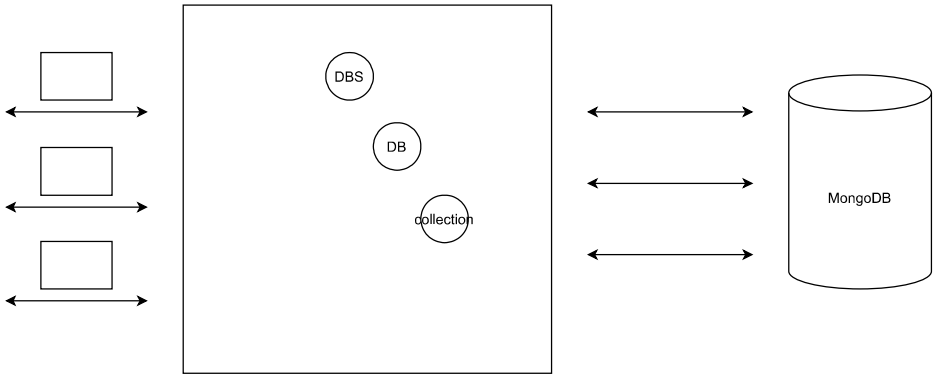


Conexiones a la base de datos, relacional VS MongoDB

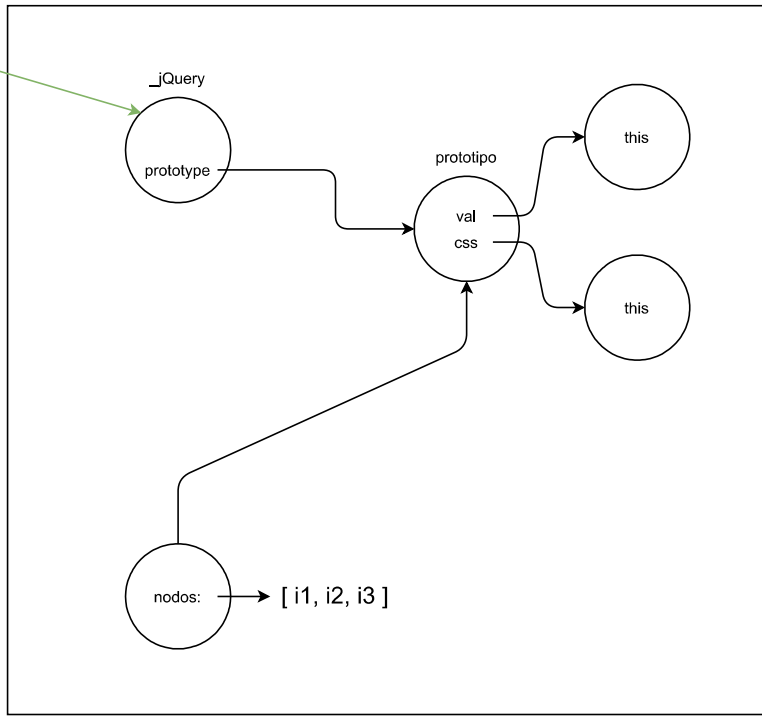
Con una base de datos relacional no podemos compartir las conexiones entre distintos hilos y hay que utilizar una diferente para procesar cada petición. PAra no crear y tirar a la basura las conexión se utiliza un pool



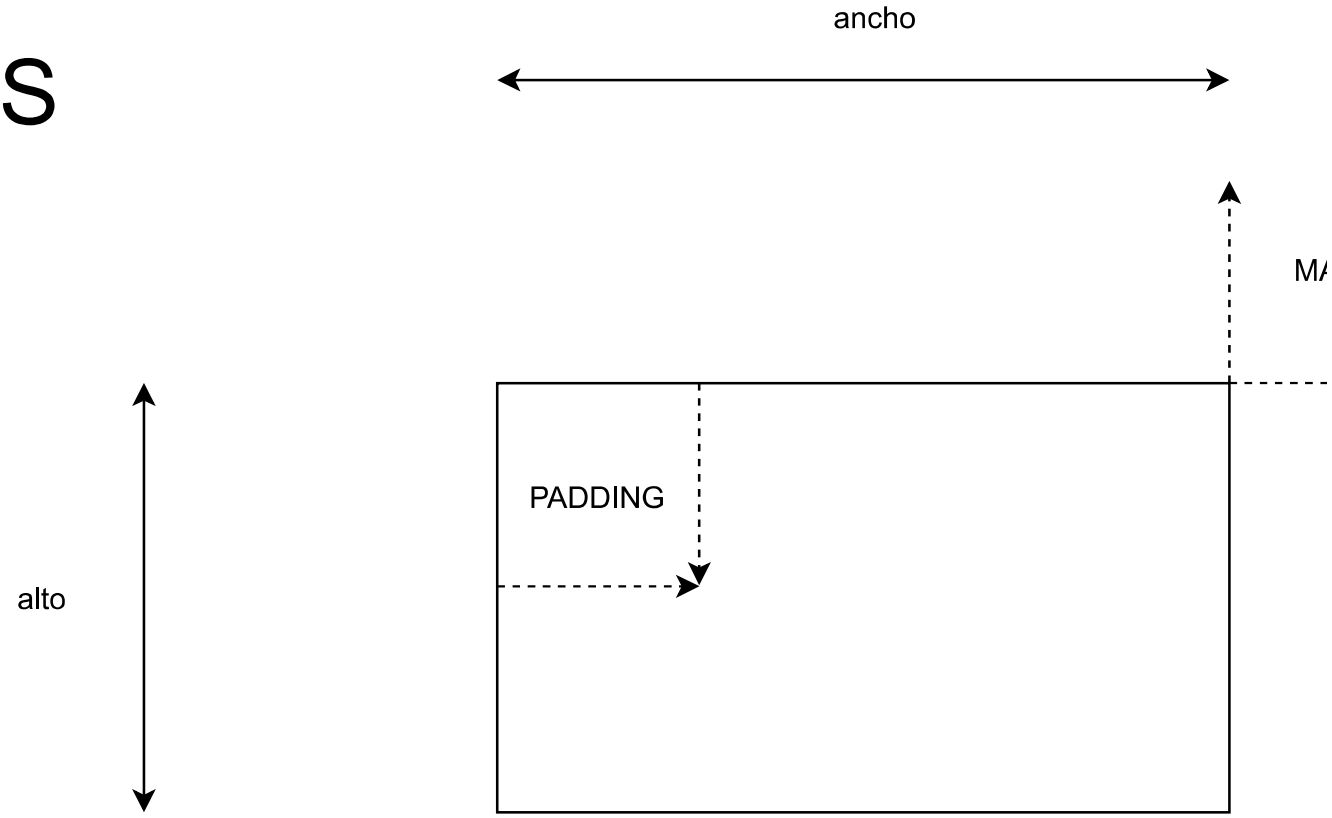
Con MongoDB obtenemos una únia conexión que al ser Thread Safe se puede compartir por todos los hilos que haya o haiga



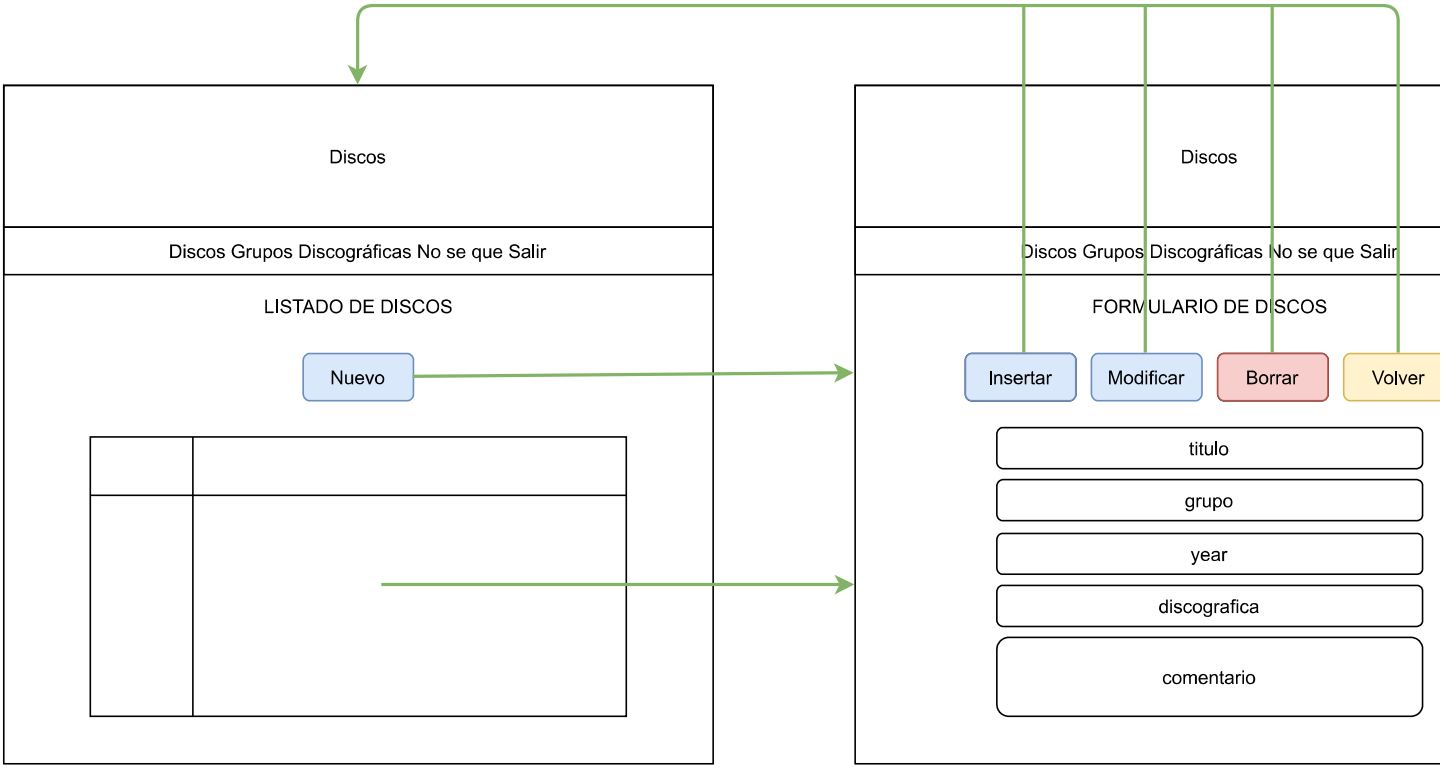
```
function _jQuery(selector){
    this.nodos = document.querySelectorAll(selector)
}
_jQuery.prototype.val = function(valor){
    for(let nodo of this.nodos){
        nodo.value = valor
    }
    return this
}
_jQuery.prototype.css = function(clave, valor){
    for(let nodo of this.nodos){
        nodo.style[clave] = valor
    }
    return this
}

let J = _jQuery

new J("input")
    .val("HABER QUE PASA")
    .css("color","pink")
```



CSS

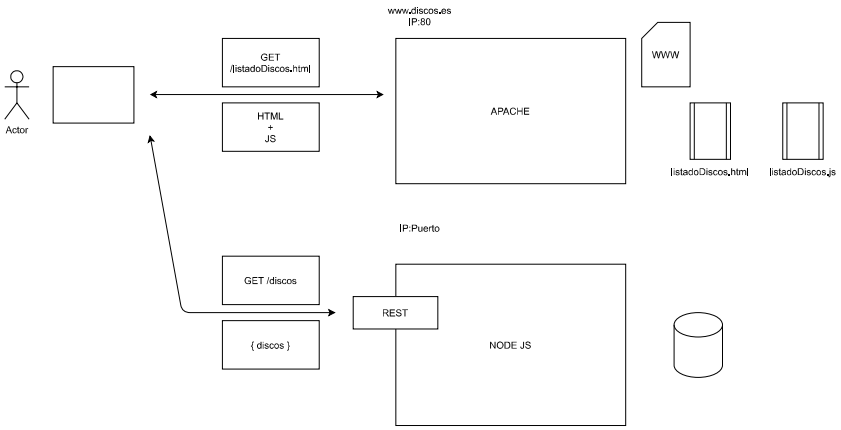


02_Node/04_REST

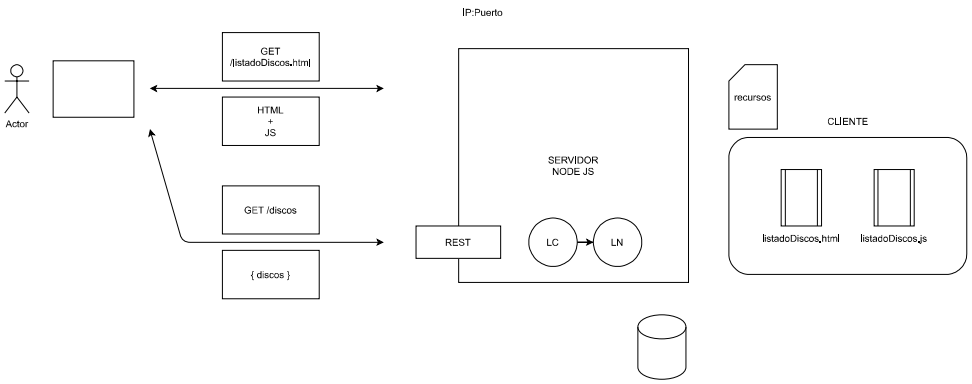


Maneras de desplegar una aplicación Node.js y el cliente web

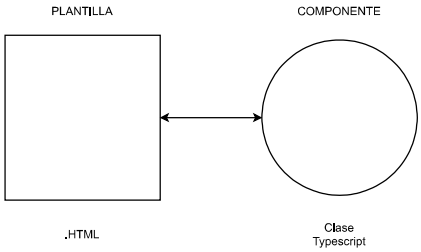
Lo mejor es delplegar el cliente en un servidor web, separandolo de la aplicación node

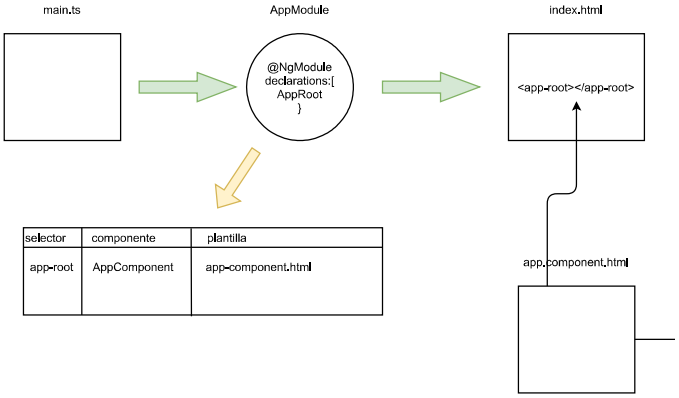


Pero tambien podemos programar la aplicación node para que entregue el contenido estático



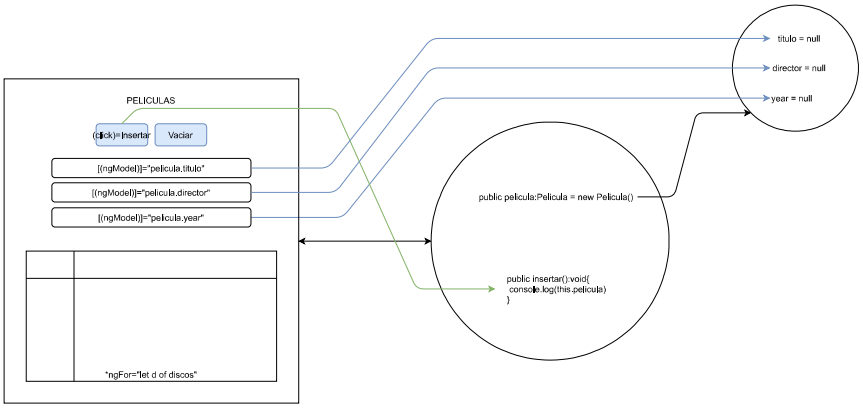
Componente en Angular



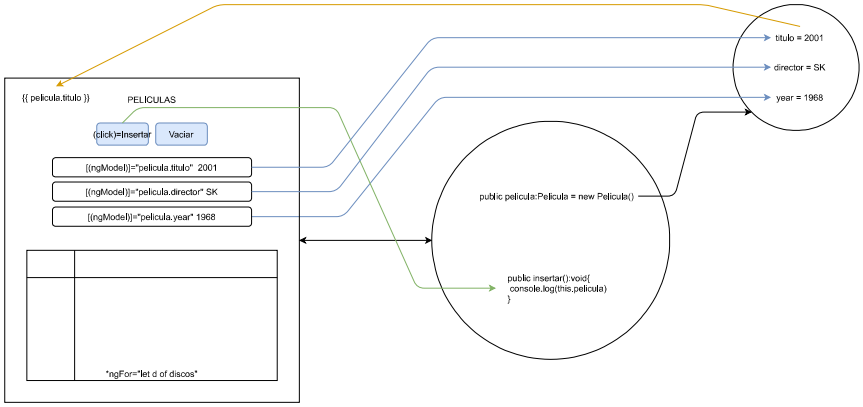


Bidirectional binding

Situación inicial



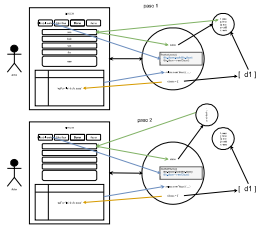
El usuario escribe algo en las cajas de texto



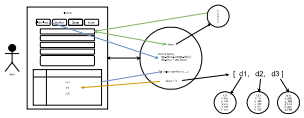
SITUACIÓN INICIAL

EL USUARIO ESCRIBE EN LAS CAJAS DE TEXTO

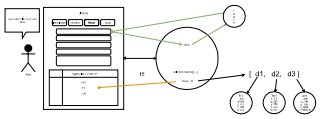
EL USUARIO PULSA EL BOTÓN INSERTAR

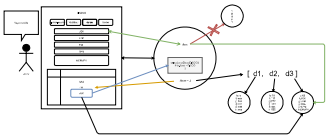


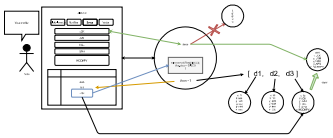
EL USUARIO YA HA INSERTADO VARIOS ÓBICOS

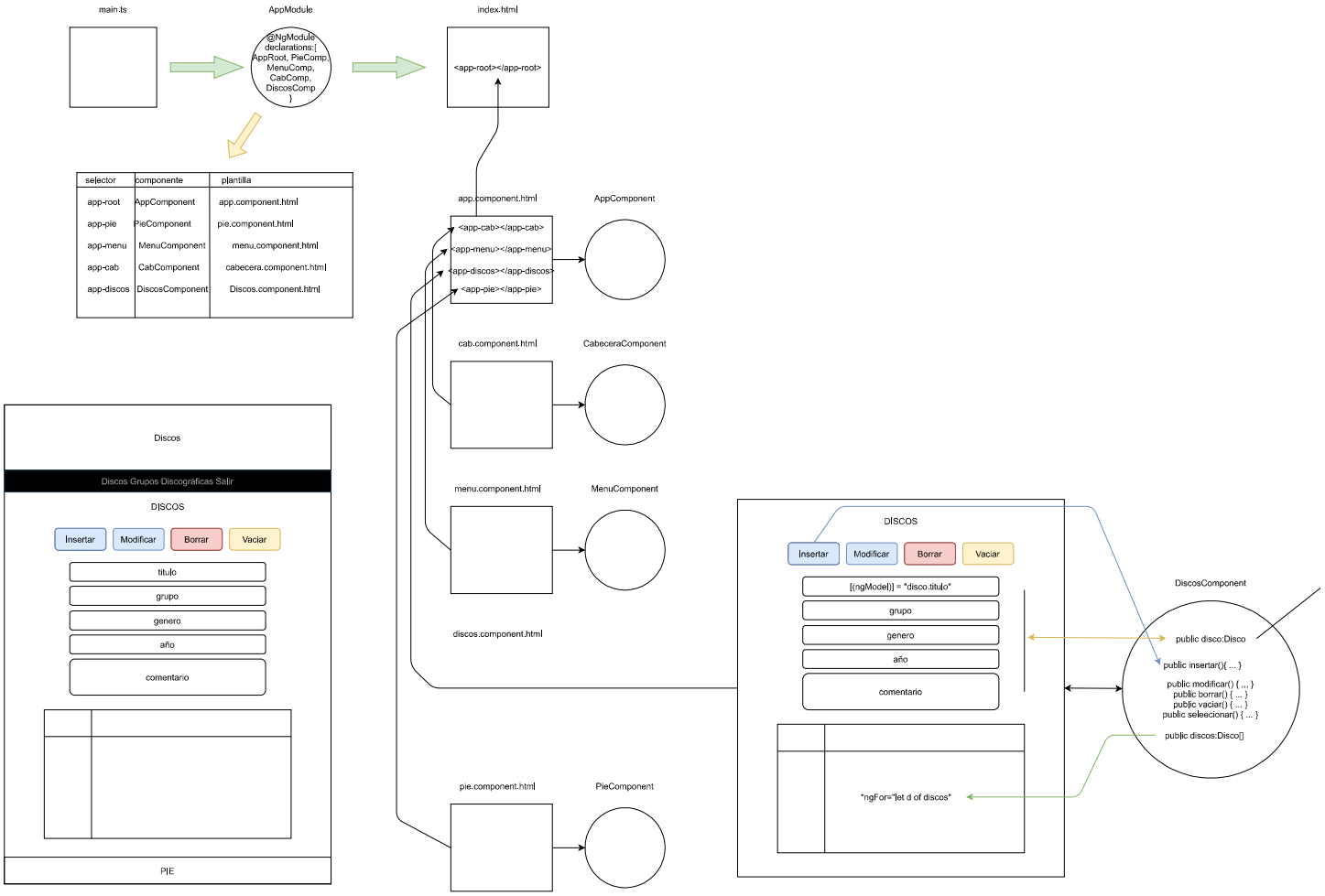


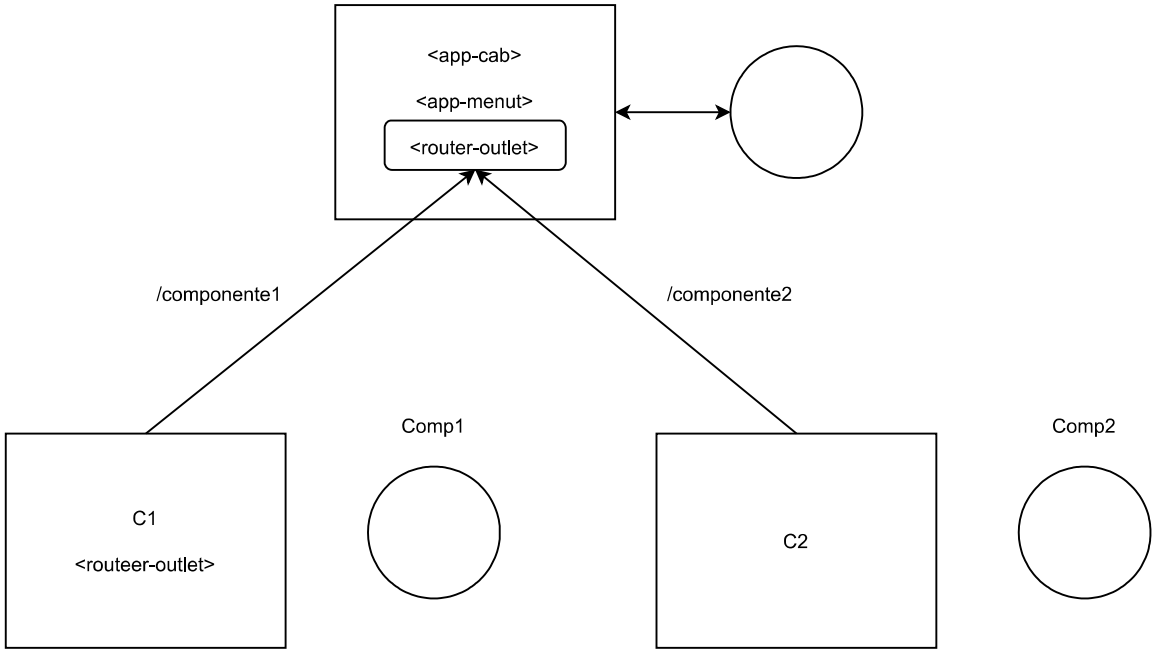
EL USUARIO HACE CLICK EN UNA FILA DE LA TABLA

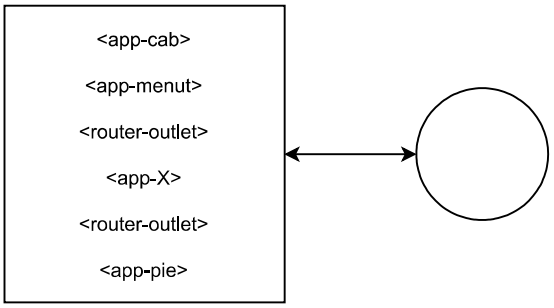




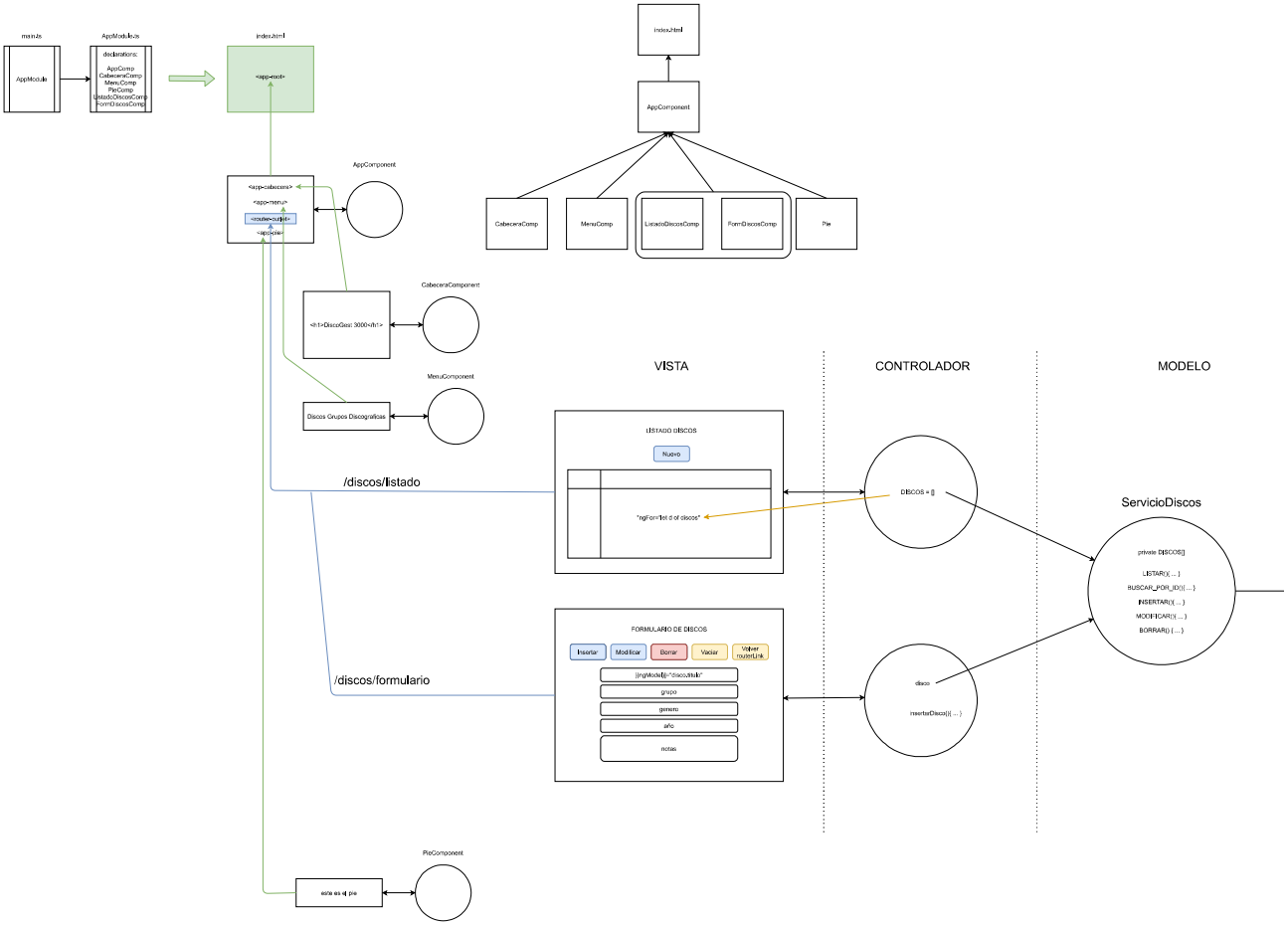



main.ts
AppModule
@NgModule
declarations:[
AppRoot, PieComp,
MenuComp,
CabComp,
DiscosComp
]
index.html
<app-root></app-root>
selector
componente
plantilla
app-root
AppComponent
app.component.html
app-pie
PieComponent
pie.component.html
app-menu
MenuComponent
menu.component.html
app-cab
CabComponent
cabecera.component.html
app-discos
DiscosComponent
Discos.component.html
app.component.html
AppComponent
<app-cab></app-cab>
<app-menu></app-menu>
<app-discos></app-discos>
<app-pie></app-pie>
cab.component.html
CabeceraComponent
menu.component.html
MenuComponent
discos.component.html
pie.component.html
PieComponent
Discos
Discos Grupos Discográficas Salir
DISCOS
Insertar
Modificar
Borrar
Vaciar
titulo
grupo
genero
año
comentario
PIE
DISCOS
Insertar
Modificar
Borrar
Vaciar
[(ngModel)] = "disco.titulo"
grupo
genero
año
comentario
"ngFor="let d of discos"
DiscosComponent
public disco:Disco
public insertar(){ ... }
public modificar() { ... }
public borrar() { ... }
public vaciar() { ... }
public seleecionar() { ... }
public discos:Disco[]

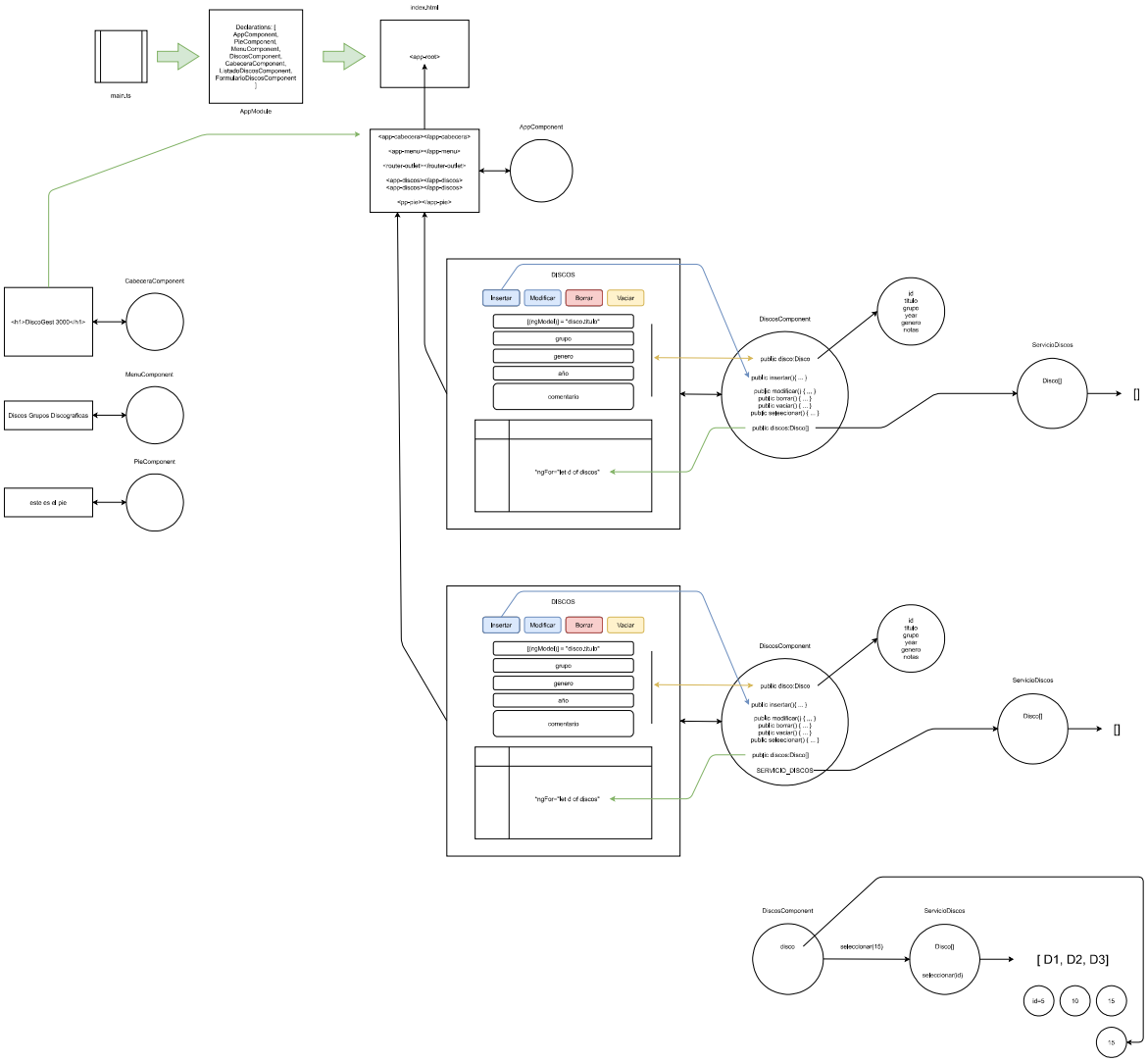

<app-cab>
<app-menut>
<router-outlet>
/componente1
/componente2
Comp1
Comp2
C1
<routeer-outlet>
C2

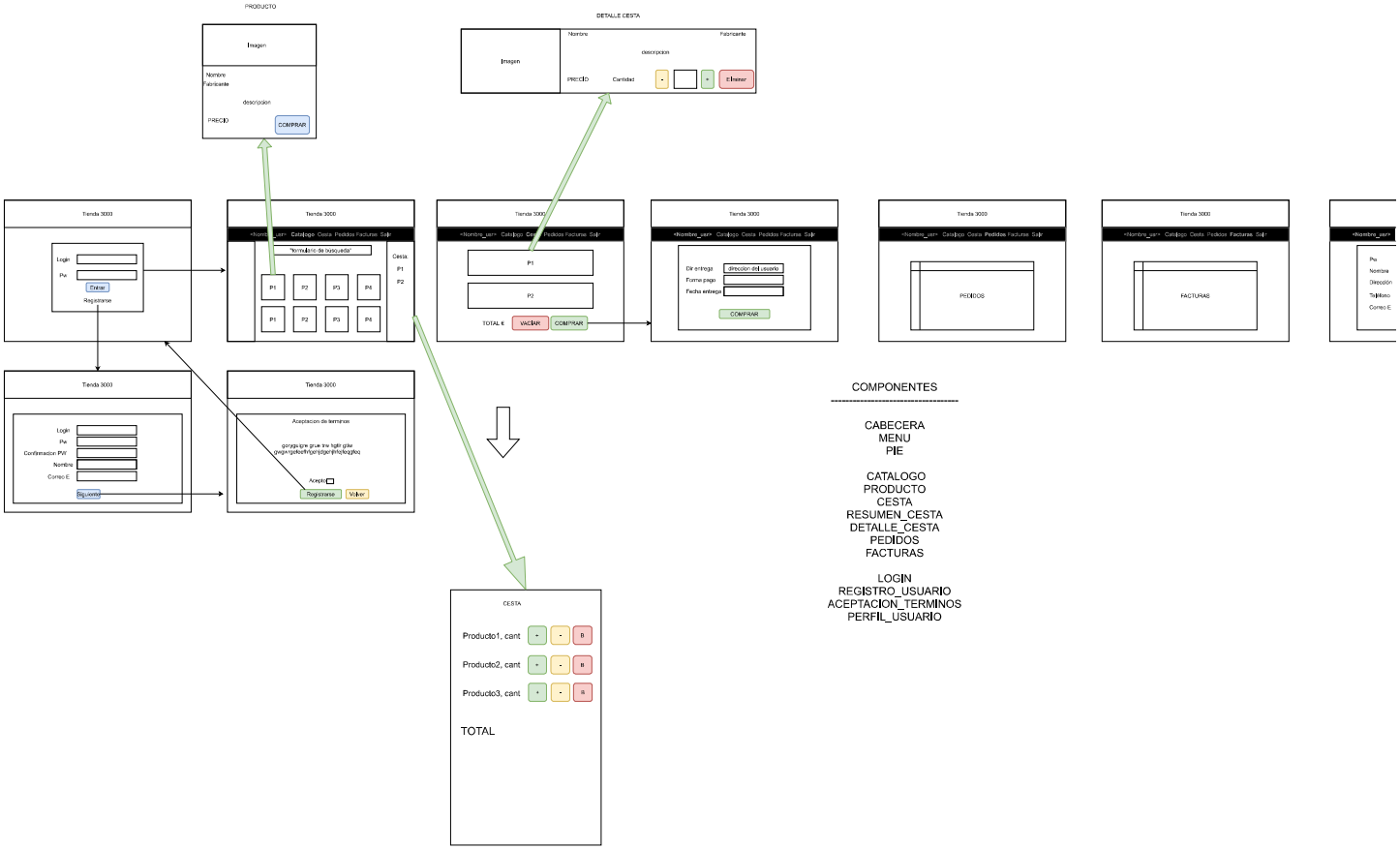

<app-cab>
<app-menut>
<router-outlet>
<app-X>
<router-outlet>
<app-pie>

ular/Ej04_Servicios



Ej04-Servicios





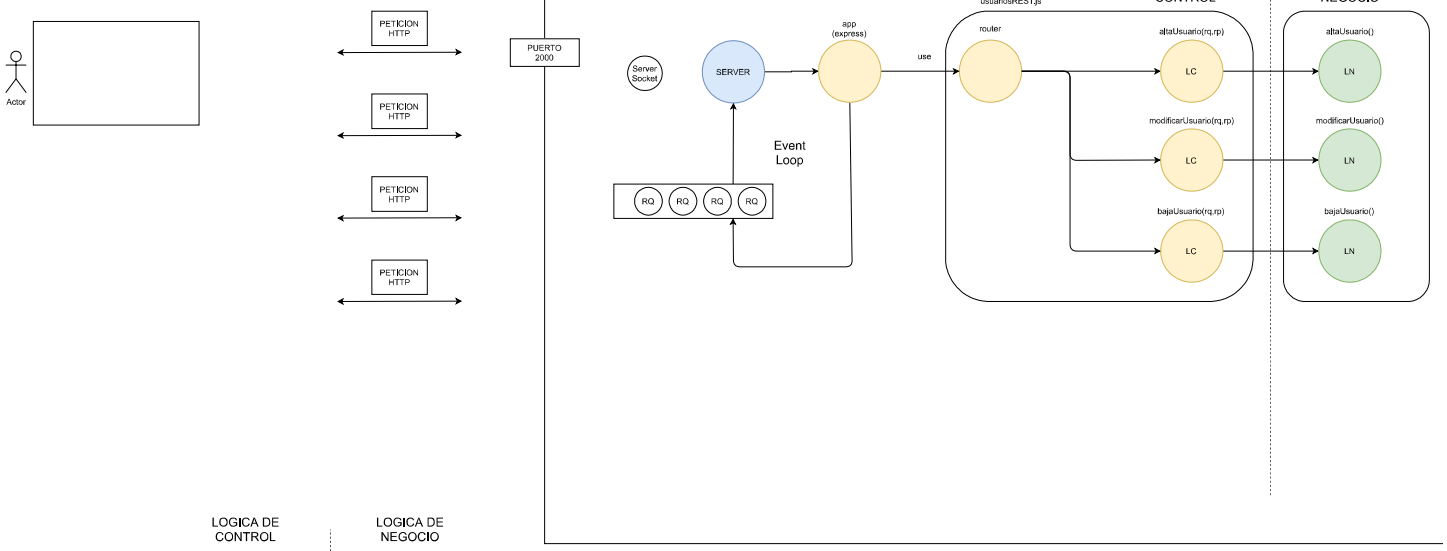
COMPONENTES

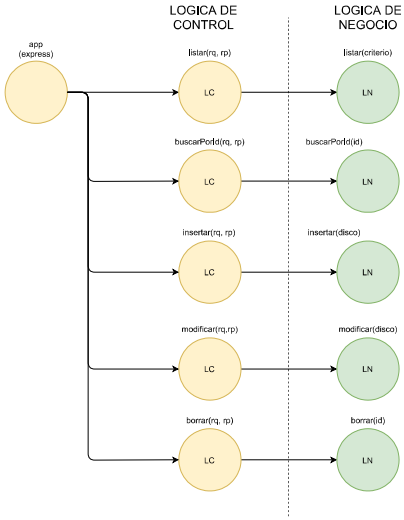
CABECERA
MENU
PIE

CATALOGO
PRODUCTO
CESTA
RESUMEN_CESTA
DETALLE_CESTA
PEDIDOS
FACTURAS

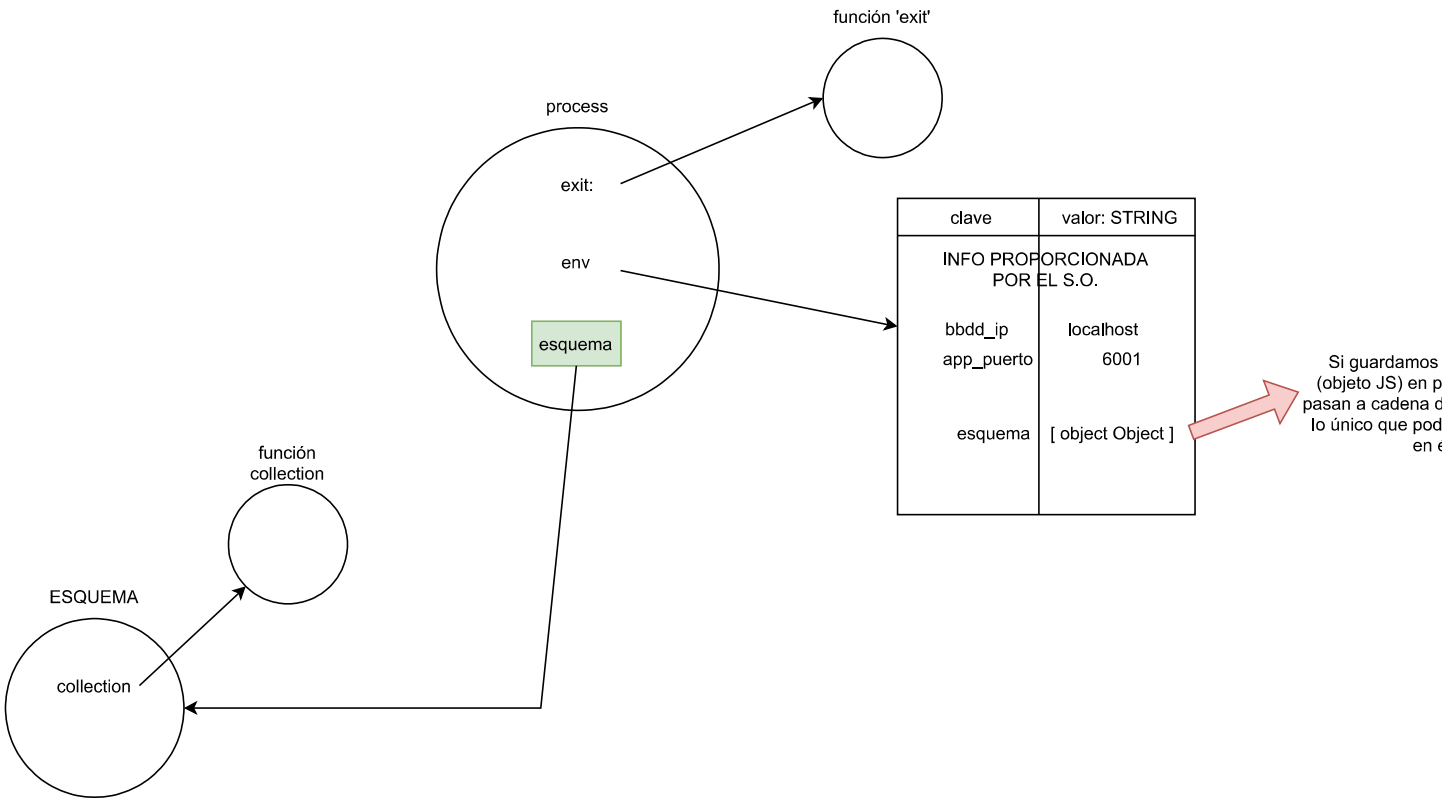
LOGIN
REGISTRO_USUARIO
ACEPTACION_TERMINOS
PERFIL_USUARIO

Tienda, lado del servidor

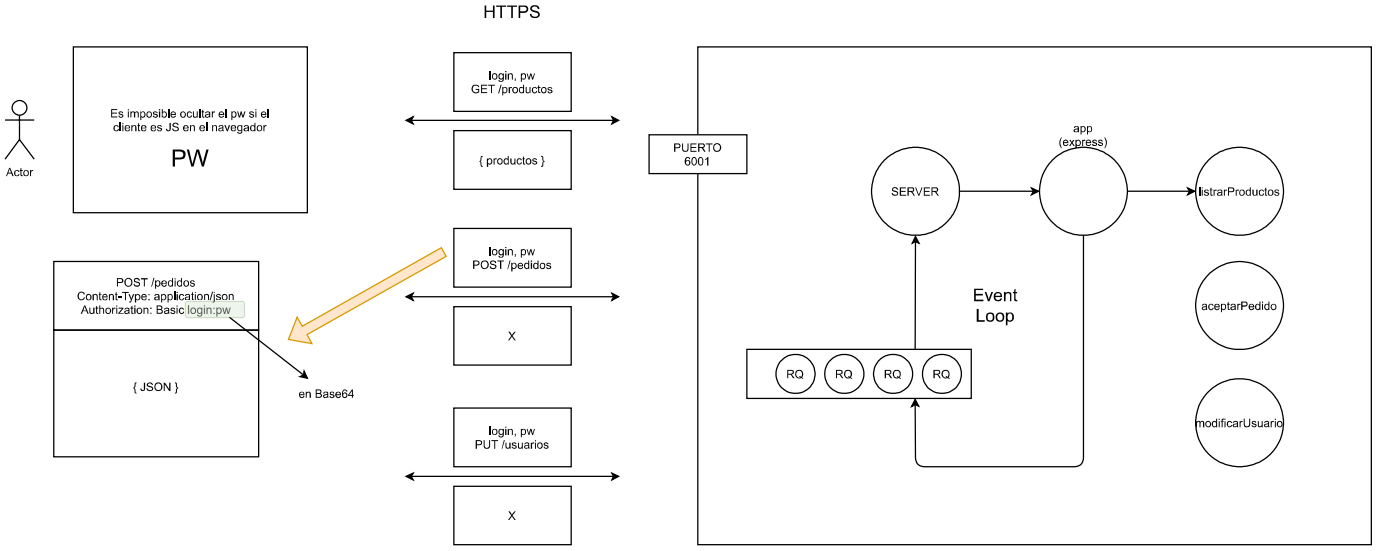




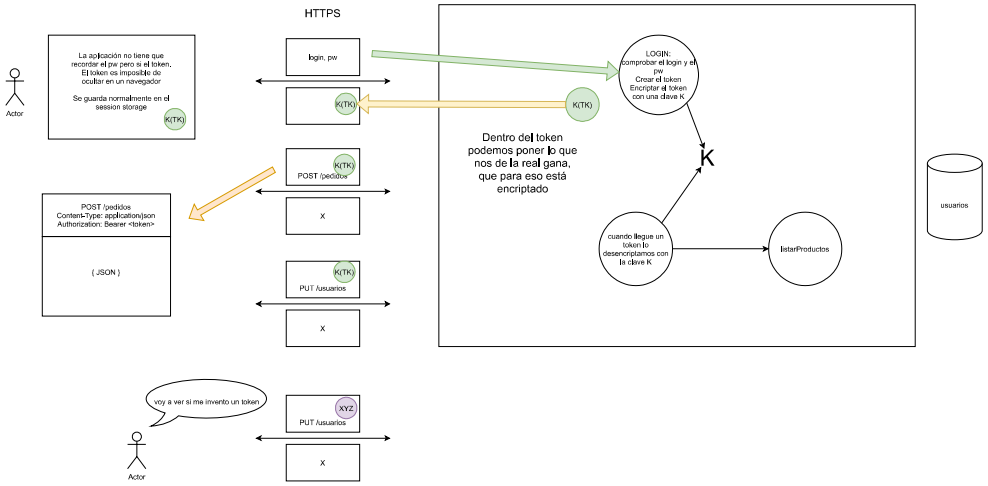
Node.js: el objeto Process



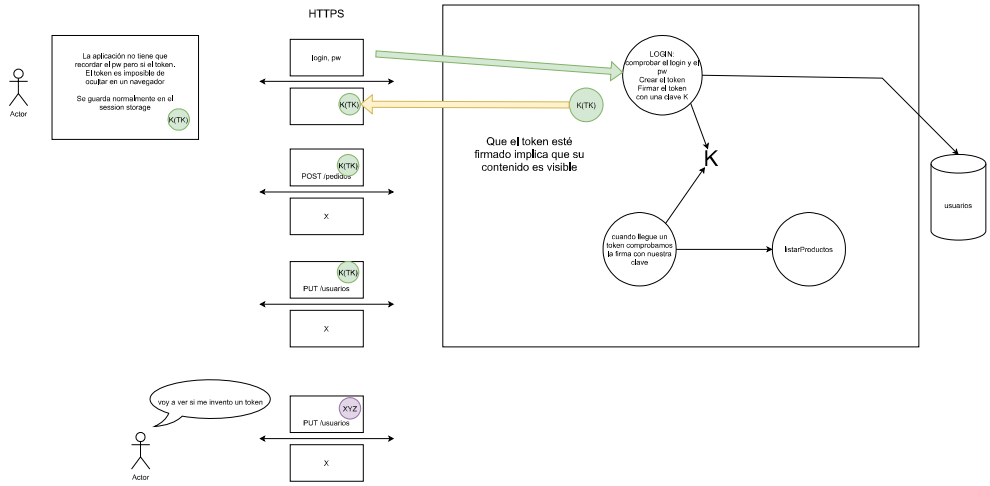
Autenticación sin token y sin estado (basic authentication)



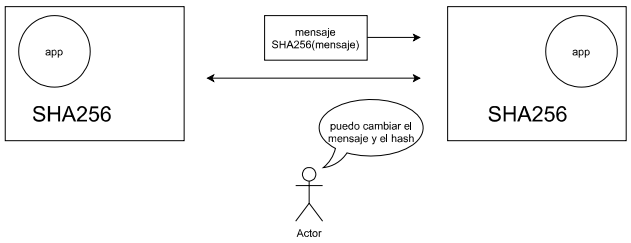
Autenticación con token y sin estado: Encriptando el token



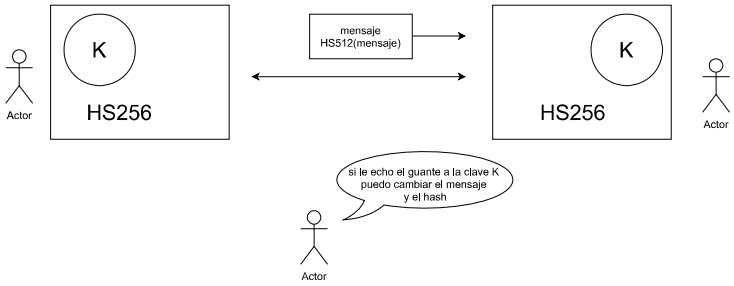
Autenticación con token y sin estado: Firmando el token



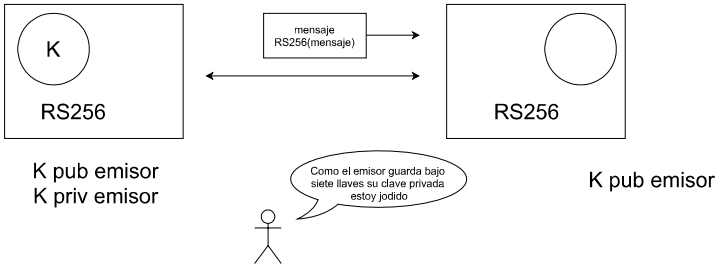
HASH de un mensaje, sin firma

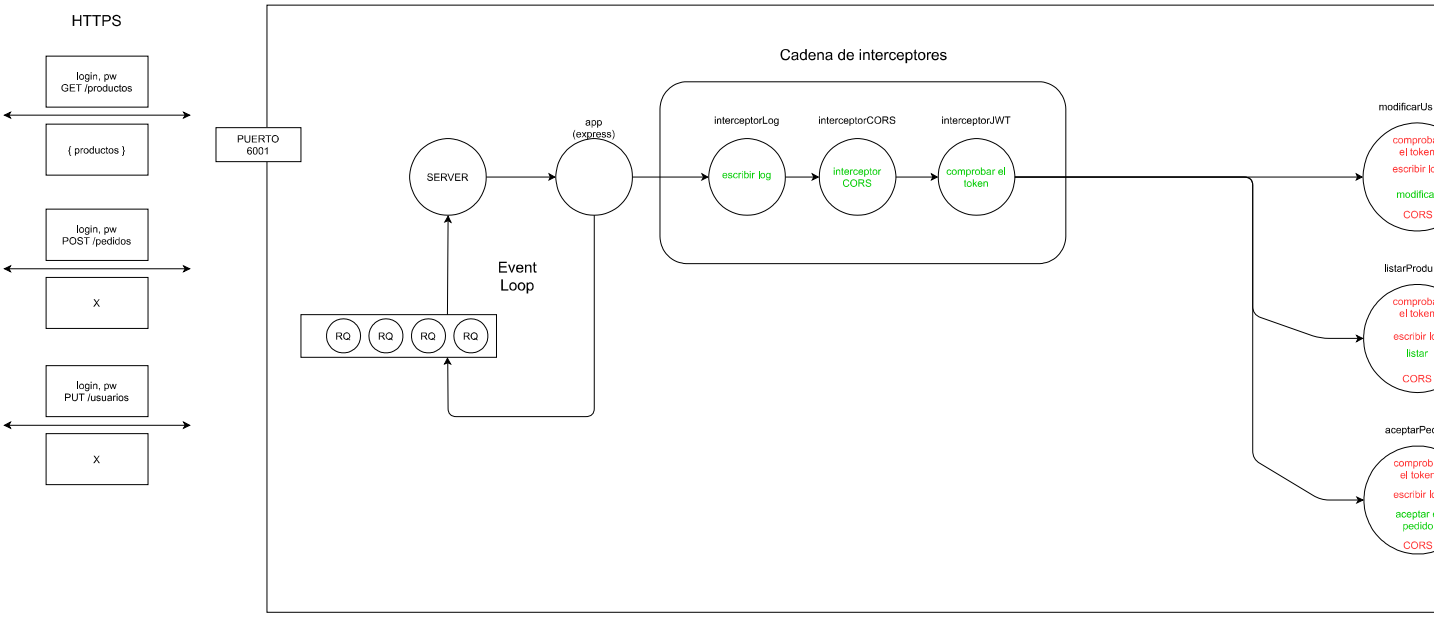


HASH de un mensaje, con firma simétrica

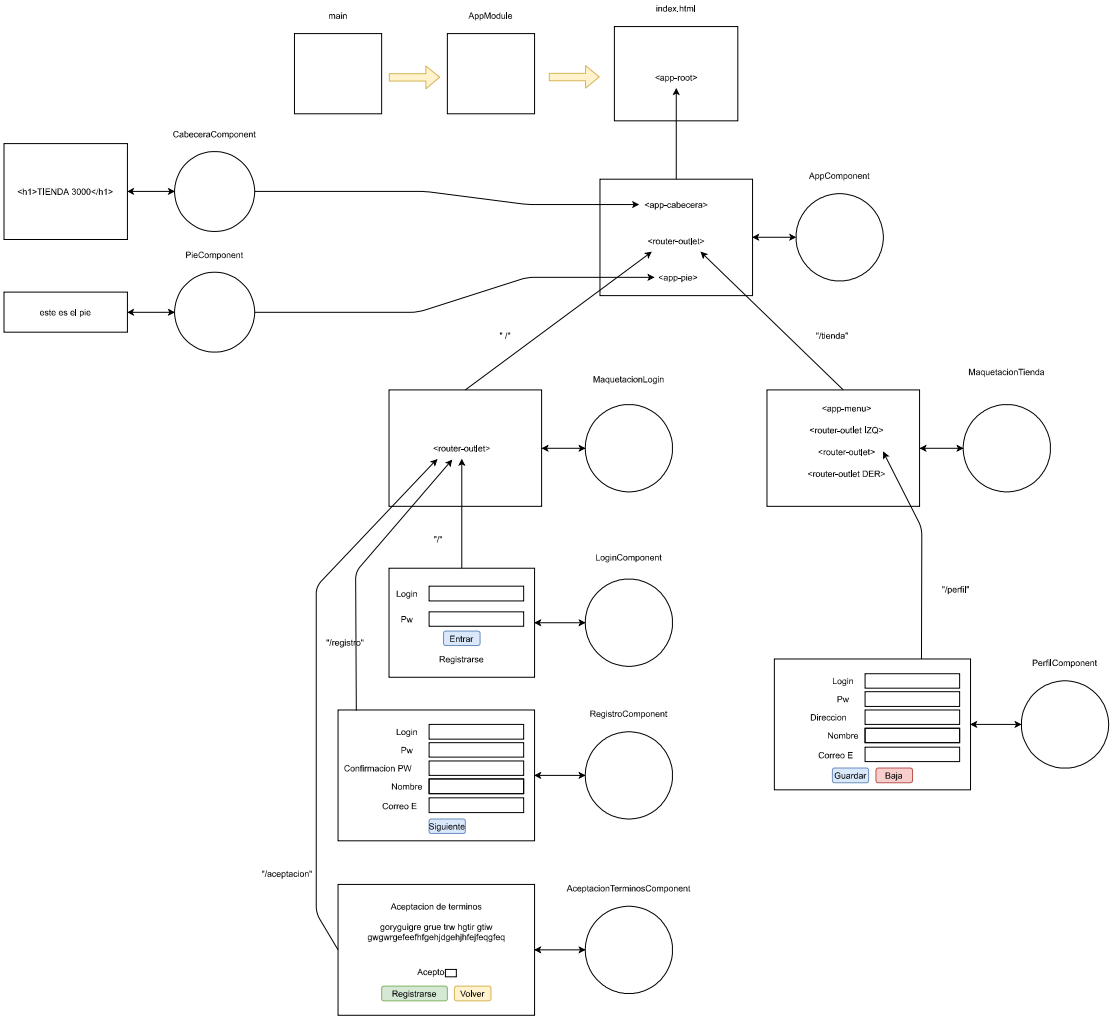


HASH de un mensaje, con firma asimétrica

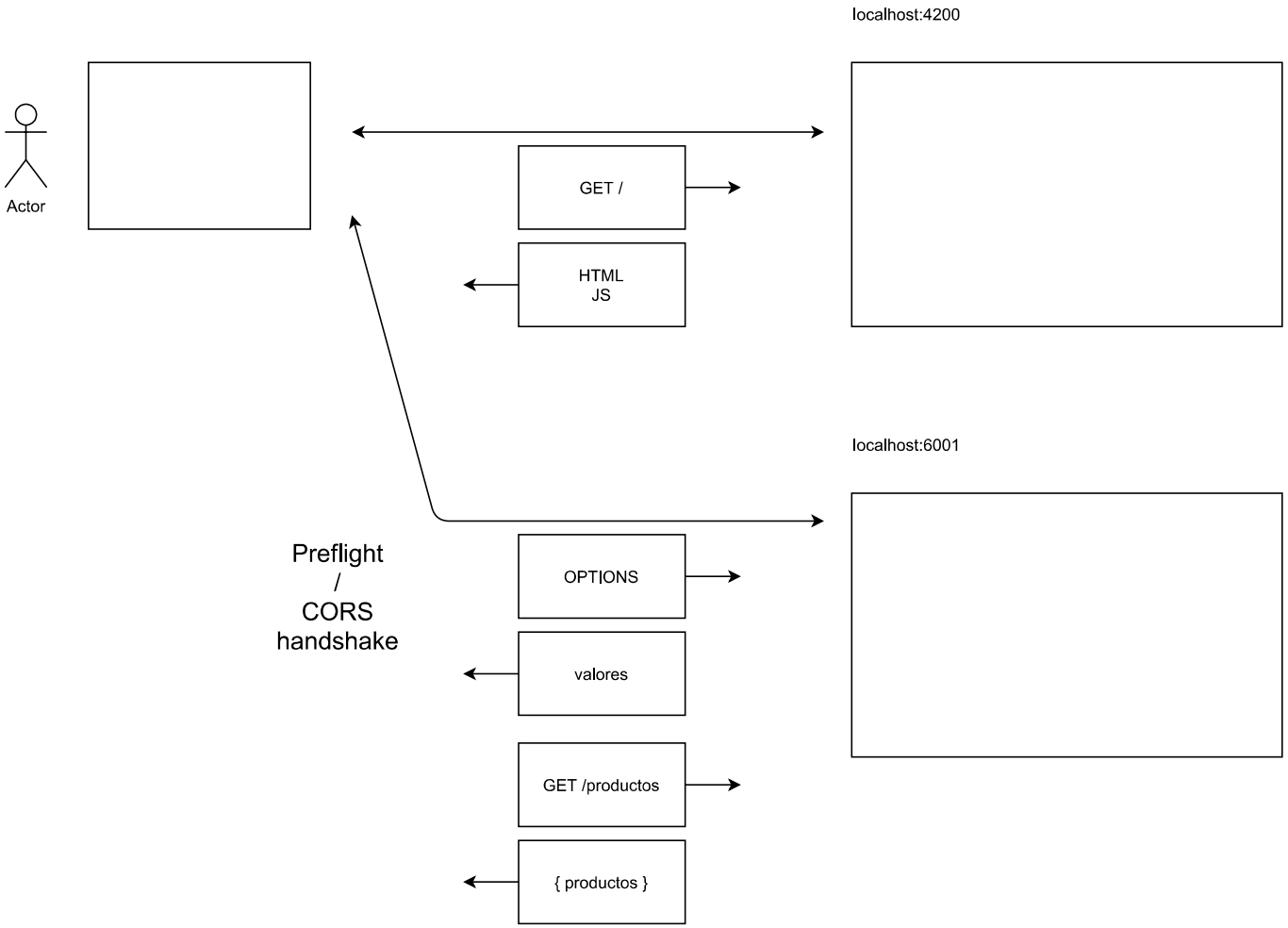



HTTPS
login, pw
GET /productos
{ productos }
login, pw
POST /pedidos
X
login, pw
PUT /usuarios
X
PUERTO
6001
SERVER
app
(express)
Event
Loop
RQ
RQ
RQ
RQ
Cadena de interceptores
interceptorLog
interceptorCORS
interceptorJWT
escribir log
interceptor
CORS
comprobar el
token
modificarUs
comproba
el token
escribir lo
modifica
CORS
listarProdu
comproba
el token
escribir lo
listar
CORS
aceptarPed
comprob
el token
escribir lo
aceptar
pedido
CORS

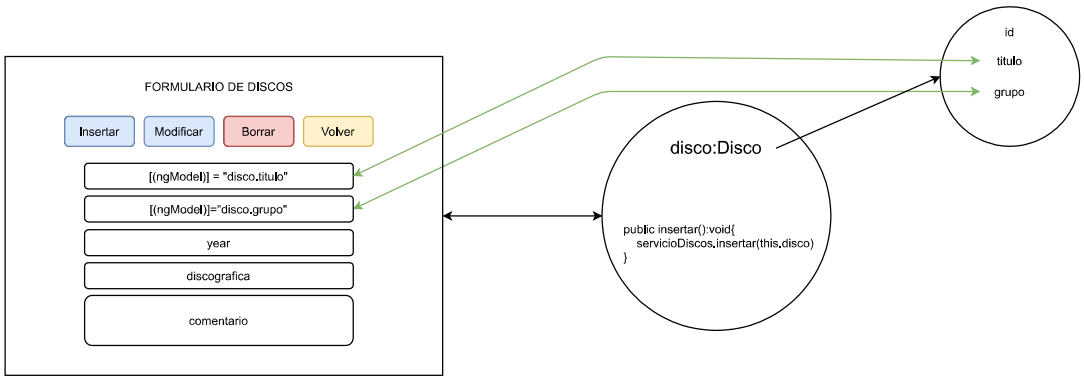
Maquetación de la tienda



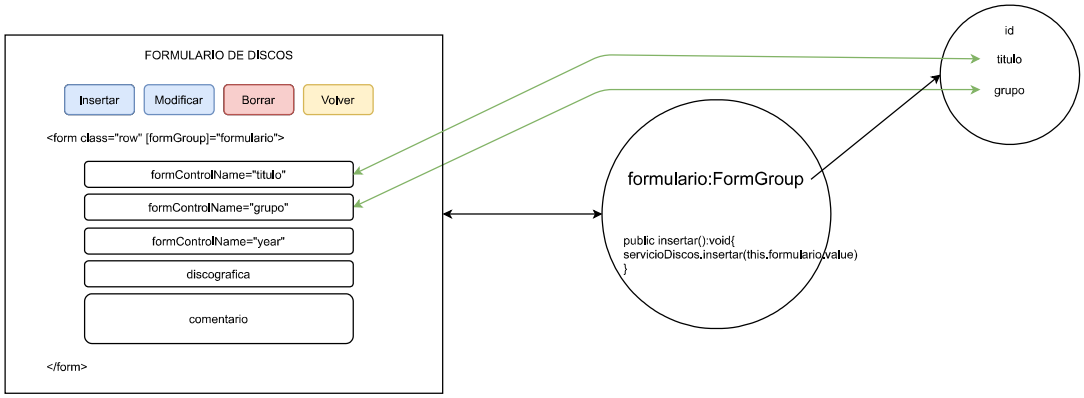
Cross Origin Resource Sharing (CORS)



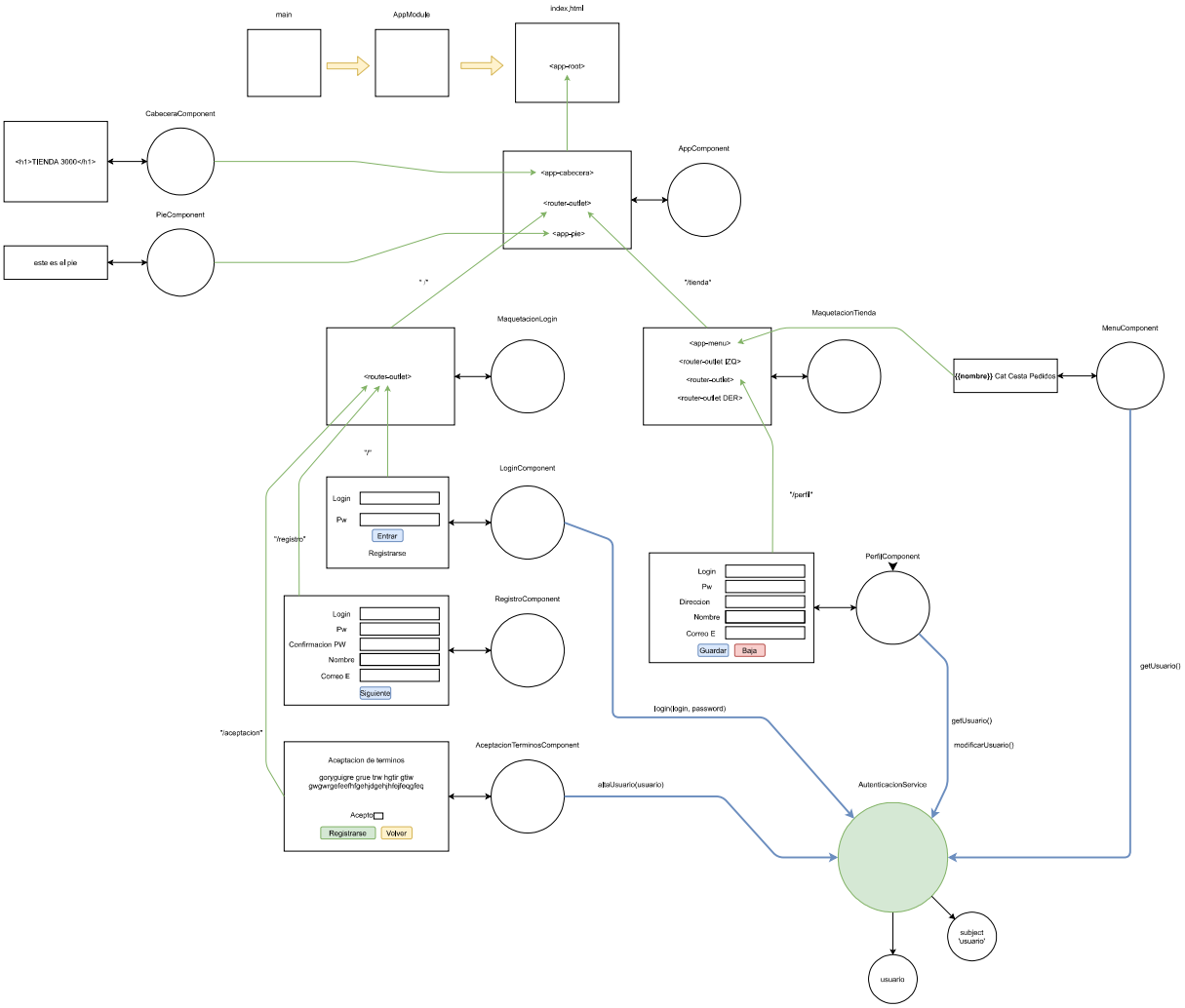
Con Bidirectional Binding



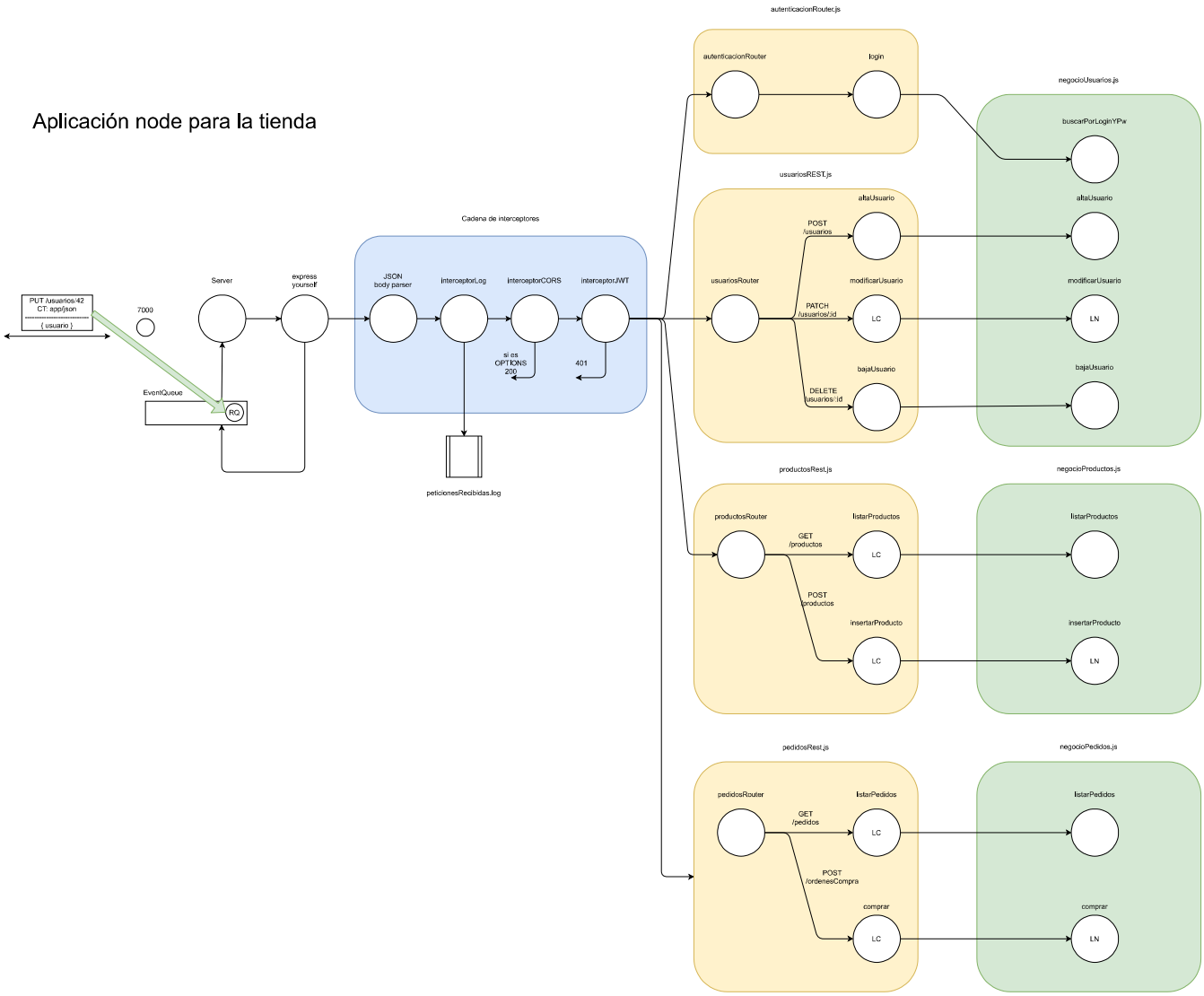
Con formularios reactivos



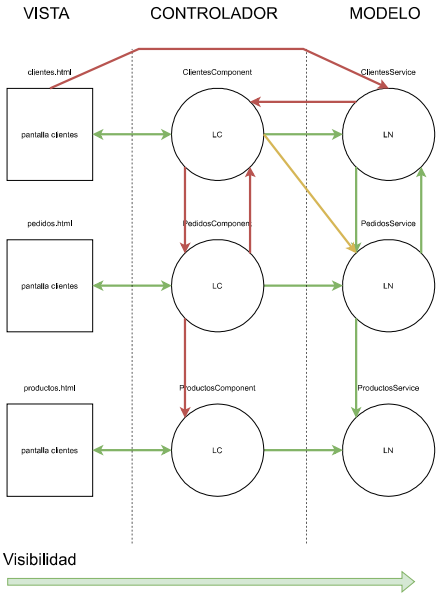
AutenticacionService, un servicio muy solicitado



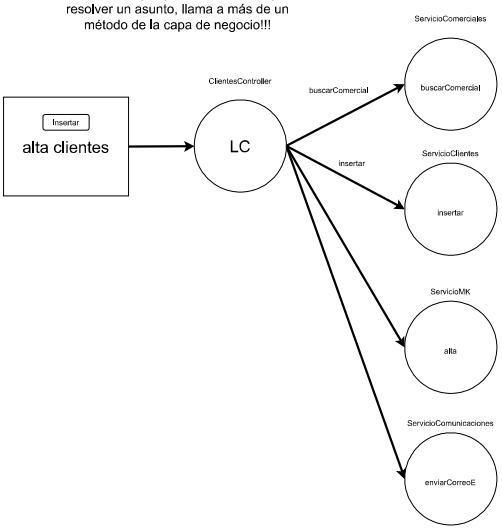
Aplicación node para la tienda



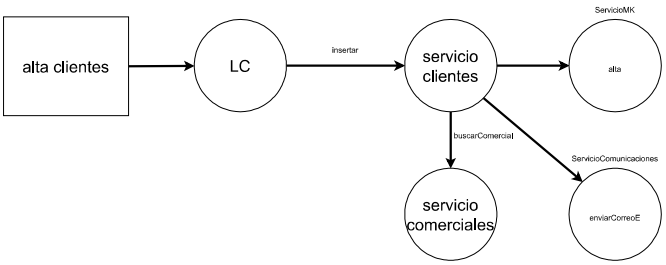
MVC, Capas, Visibilidad



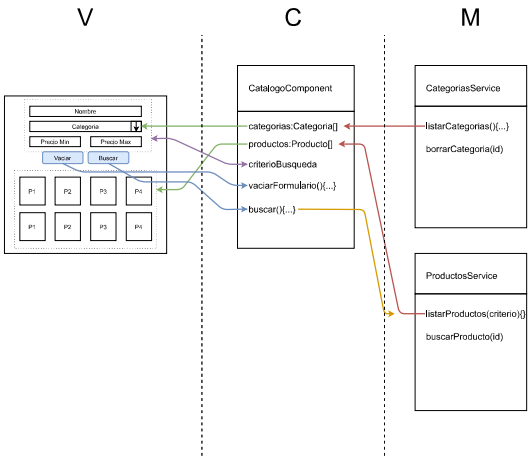
Esto está mal porque el controlador, para resolver un asunto, llama a más de un método de la capa de negocio!!!

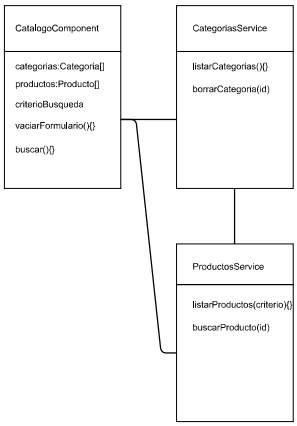


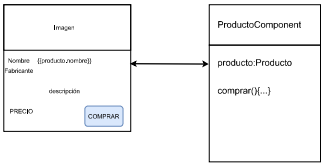
Así si

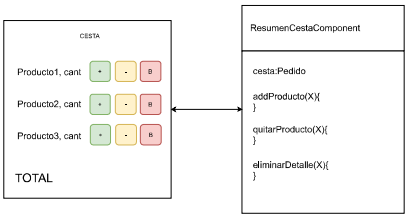


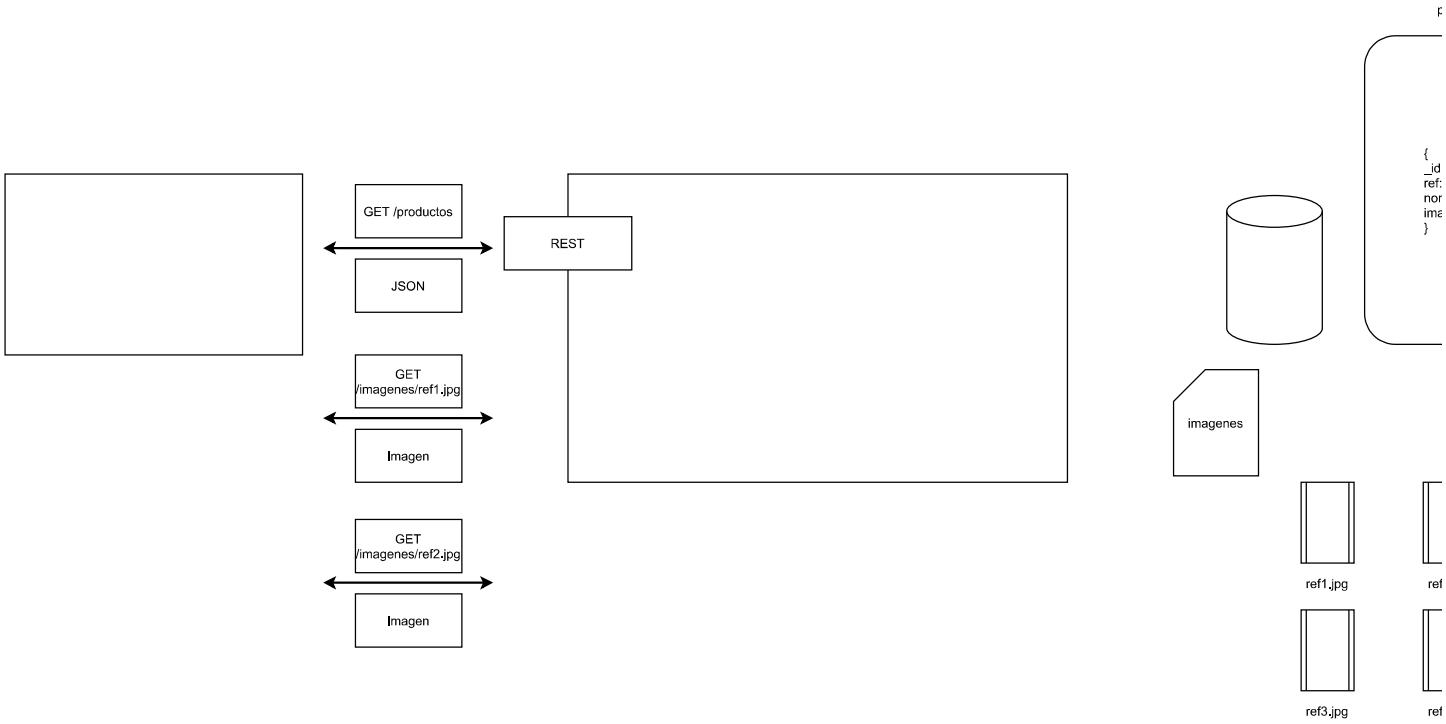
atalogoComponent



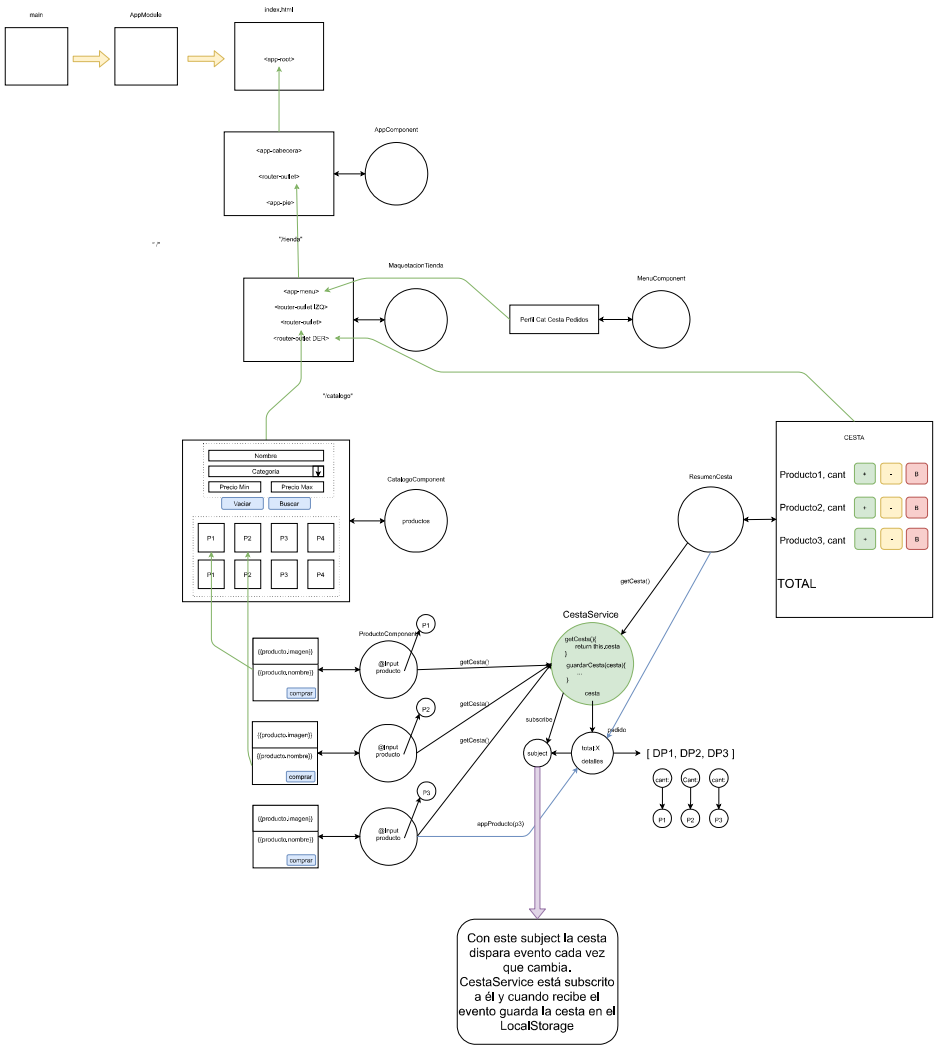




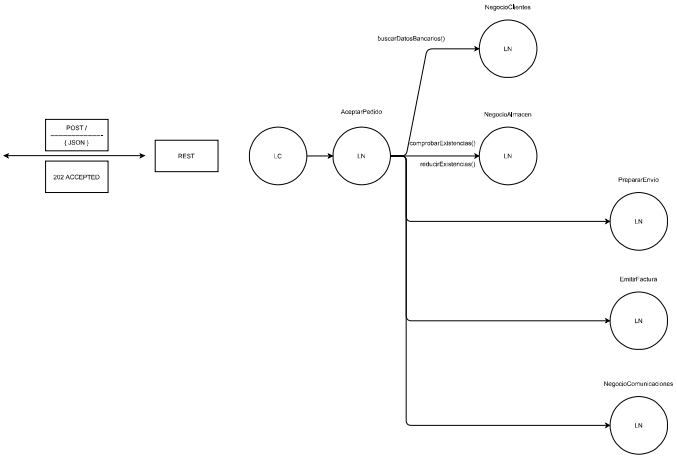



GET /productos
JSON
REST
GET /imagenes/ref1.jpg
Imagen
GET /imagenes/ref2.jpg
Imagen
imagenes
ref1.jpg
ref3.jpg

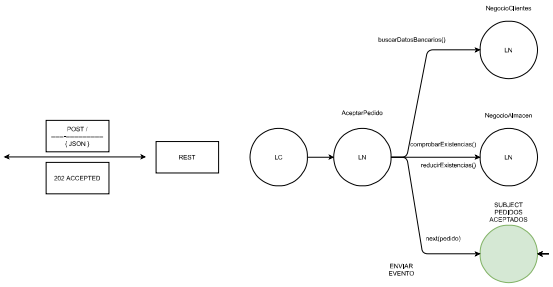
CestaService, otro servicio muy solicitado



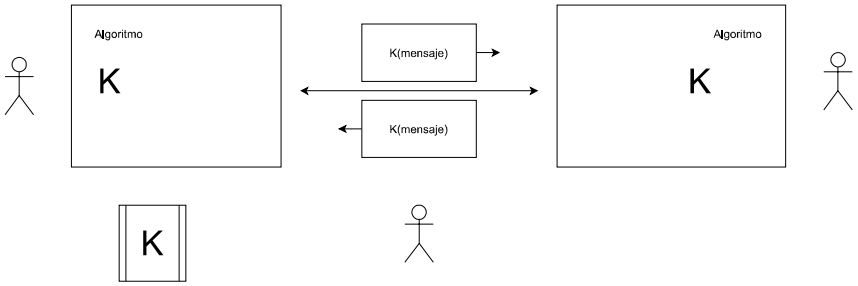
Sin eventos



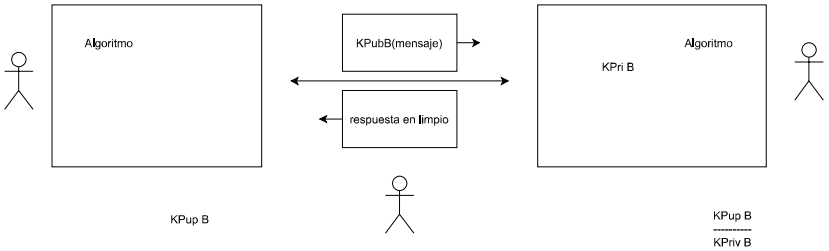
Con eventos

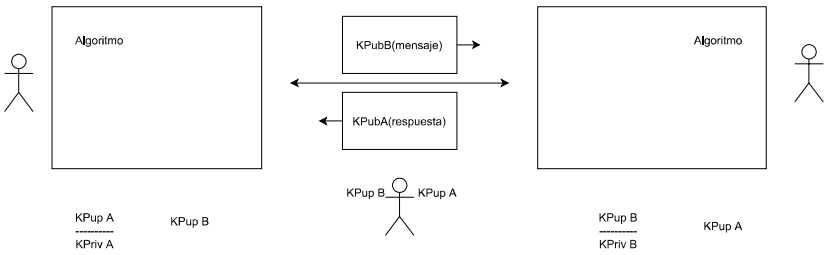


Algoritmos de encriptación simétricos

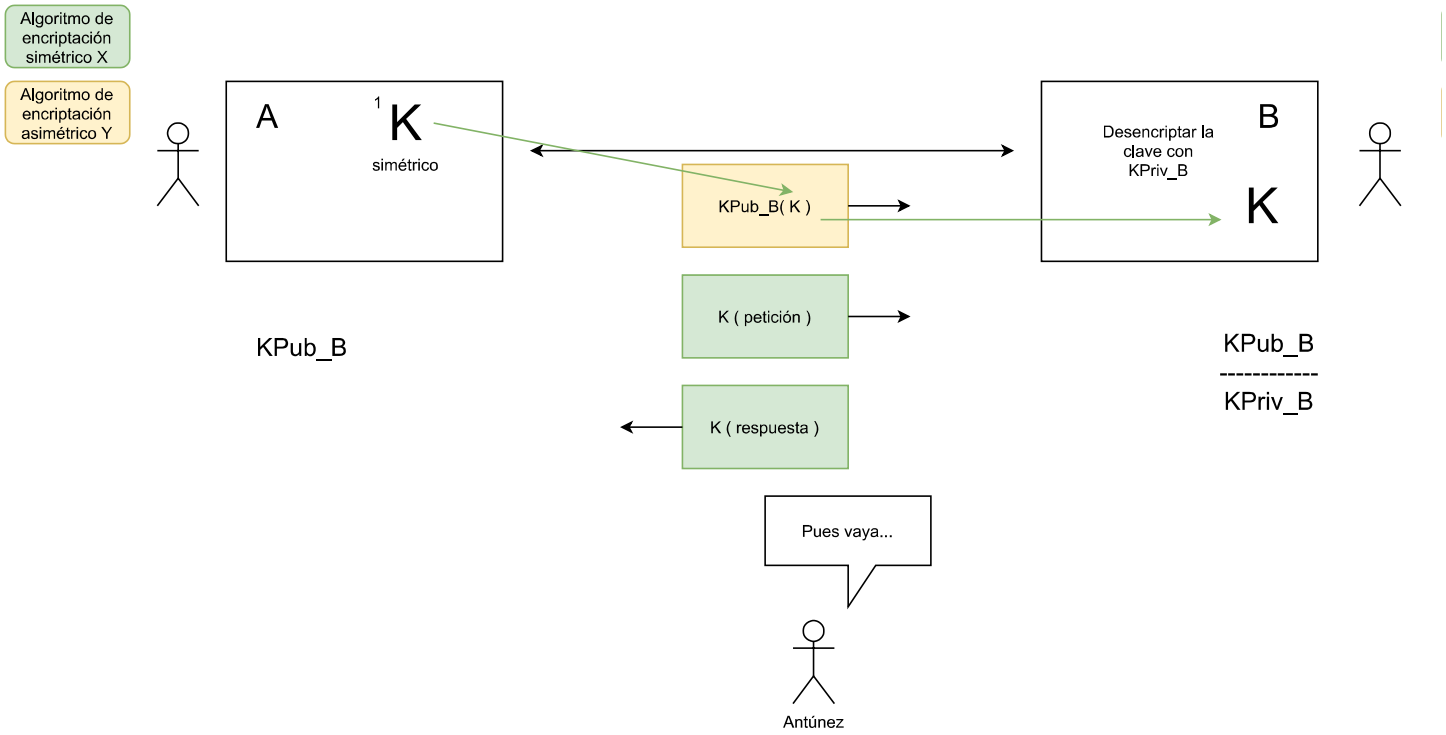


Algoritmos de encriptación asimétricos

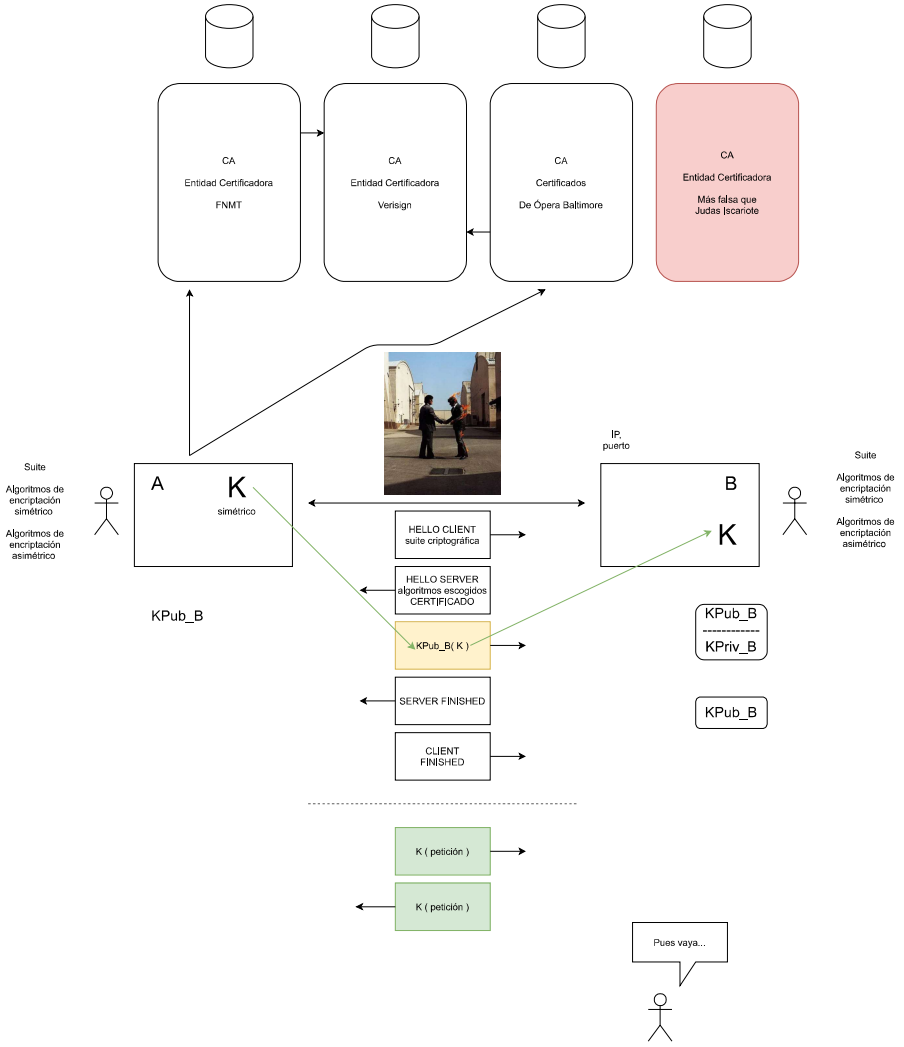




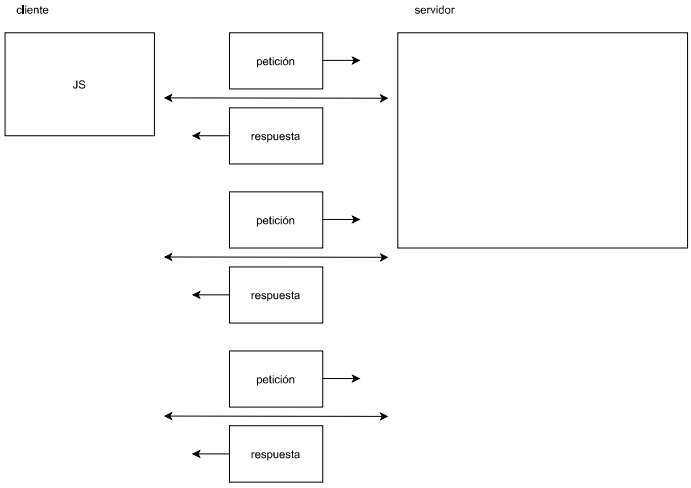
algoritmo simétrico con el que se encriptarán los mensajes



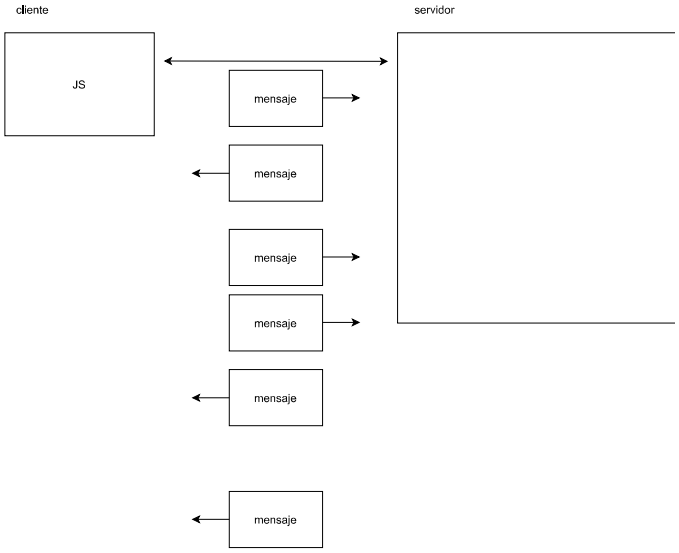
SSL/TLS Handshake



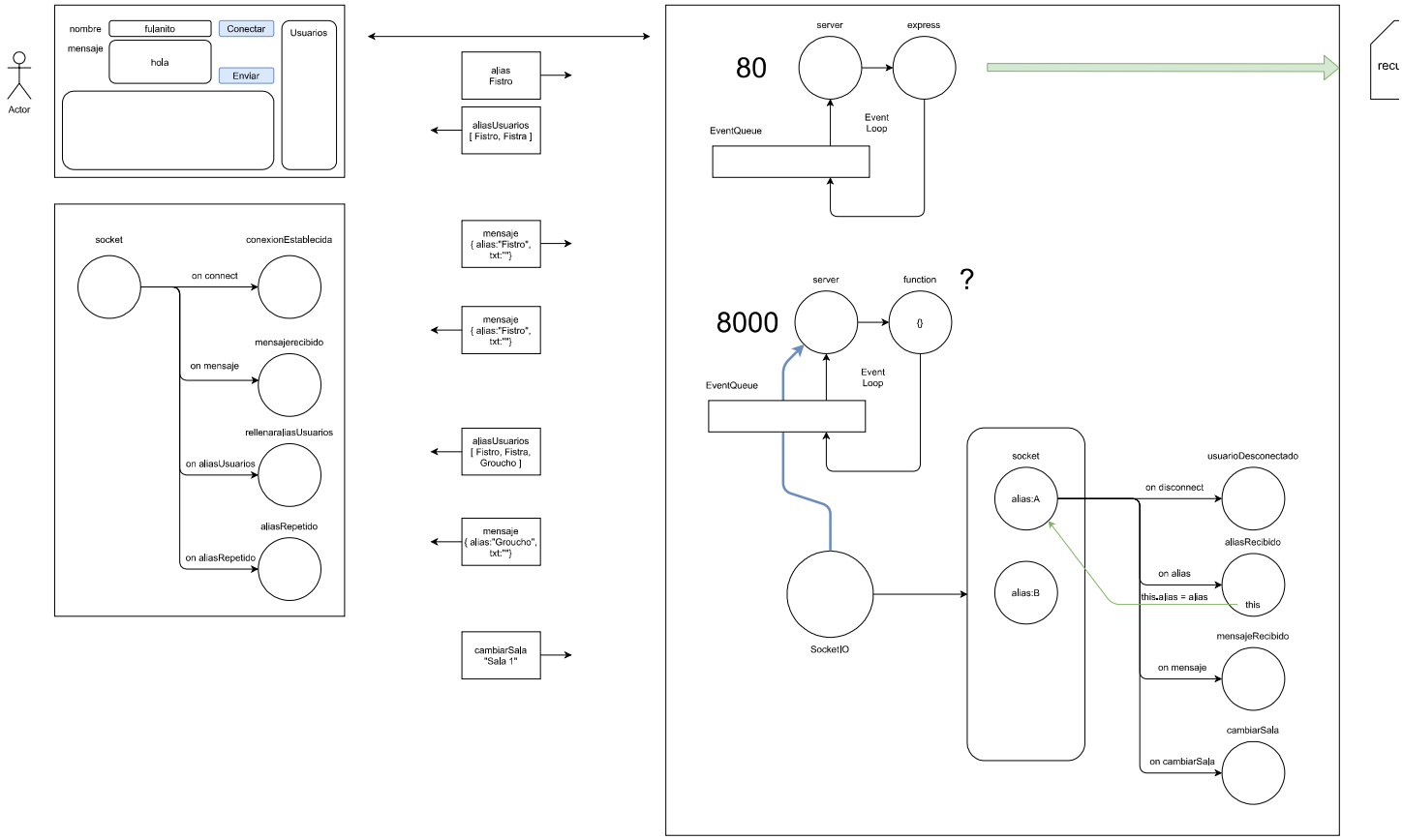
HTTP, petición-respuesta



Websockets

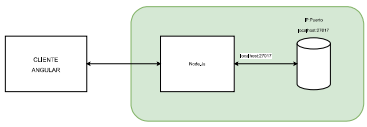


Chat 3000

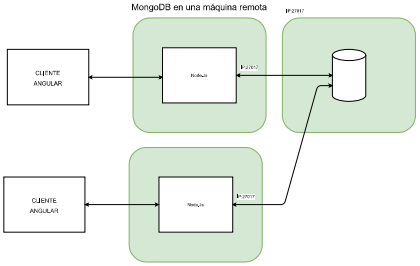

Actor
nombre
fulanito
Conectar
mensaje
hola
Enviar
Usuarios
socket
conexionEstablecida
on connect
mensajerecibido
on mensaje
rellenaraliasUsuarios
on aliasUsuarios
aliasRepetido
on aliasRepetido
alias
Fistro
aliasUsuarios
[Fistro, Fistra]
mensaje
{ alias:"Fistro",
txt:""}
mensaje
{ alias:"Fistro",
txt:""}
aliasUsuarios
[Fistro, Fistra,
Groucho]
mensaje
{ alias:"Groucho",
txt:""}
cambiarSala
"Sala 1"
80
server
express
EventQueue
Event
Loop
8000
server
function
?
{}
EventQueue
Event
Loop
SocketIO
socket
alias:A
alias:B
usuarioDesconectado
on disconnect
aliasRecibido
on alias
this.alias = alias
this
mensajeRecibido
on mensaje
cambiarSala
on cambiarSala

MongoDB Realm Serverless

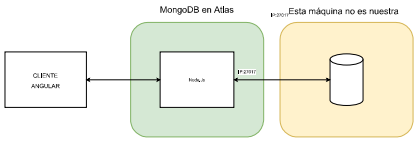
App Node y MongoDB en localhost



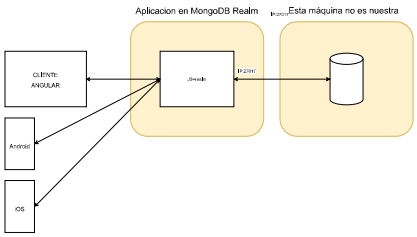
MongoDB en una máquina remota

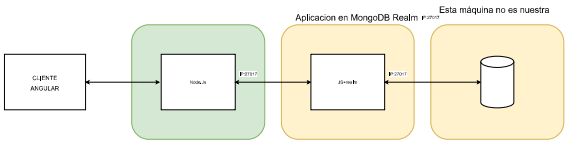


MongoDB en Atlas



Aplicacion en MongoDB Realm





MongoDB Realm QueryAnywhere