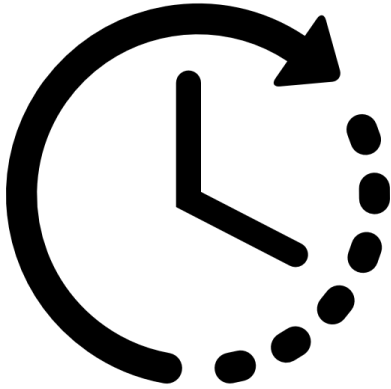




# Motivación

---



Datos en tiempo real



Datos sin conexión



Fácil escalado

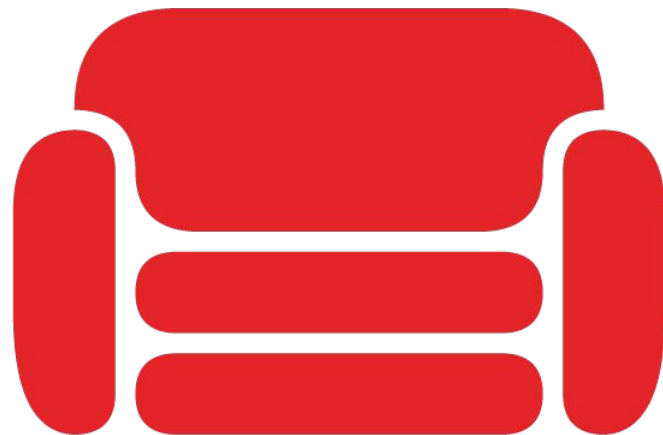
# Introducción

— — —

- Sistema creado en 2005 por Damien Katz
- Mantenido por Apache desde 2008
- Utilizado entre otros por **npm**

## Características:

- Documentos JSON sin esquema fijo
- API HTTP
- Fácil replicación



Databases

Database name ▼

Create Database

{ } JSON

Databases

Setup

Active Tasks

Configuration

Replication

Documentation

Verify

Your Account

Fauxton on Apache CouchDB  
v. 2.1.1

Name	Size	# of Docs	Actions
<a href="#">_global_changes</a>	71.7 KB	126	
<a href="#">_replicator</a>	2.3 KB	1	
<a href="#">_users</a>	17.4 KB	3 ⓘ	
<a href="#">my-workspace</a>	4.9 MB	8	

Showing 1–4 of 4 databases.

« 1 »

```
1 {  
2   "_id": "my-workspace",  
3   "_rev": "13-b71fbe8a5d7a0a8f23f11cdb670091c0",  
4   "title": "My nice workspace",  
5   "users": [  
6     "prz.lester@gmail.com",  
7     "lester_basket@hotmail.com",  
8     "test@test.com"  
9   ],  
10  "picture": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAlgAAAJYCAYAAAC  
11  "description": "A nice workspace",  
12  "admins": [  
13    "prz.lester@gmail.com"  
14  ],  
15  "type": "workspace",  
16  "timestamp": 1527687368582  
17 }
```

Documento JSON

# Revisiones

— — —

CouchDB está pensado para formar sistemas distribuidos facilitando la replicación de las bases de datos, por ello hace falta saber qué revisión de un documento tiene cada nodo del sistema, para esto se utiliza el campo **\_rev**.

Además los documentos no se bloquean, esto hace que sea más eficiente.

Debido a estas características se darán conflictos que habrá que resolver.

Esto también permite acceder a revisiones anteriores de un documento.

```
"_rev": "13-b71fbe8a5d7a0a8f23f11cdb670091c0"
```

# Conflictos

— — —

Los conflictos se dan cuando se escribe un documento con un campo **\_rev** distinto al que está almacenado en la base de datos. Cuando esto ocurra CouchDB elegirá una versión arbitraria como ganadora.

Para ver los conflictos de un documento se debe añadir el parámetro **?conflicts=true** al leer el documento, y así poder solucionarlos usando la estrategia que prefiramos, o incluso preguntando al usuario.

# Operaciones básicas



# Crear documento con \_id

— — —

Method	Address
PUT	http://localhost:5984/my-database/new-doc

Body	Response
<pre>{   "_id": "new-doc",   "hello": "world" }</pre>	<pre>{   "ok": true,   "id": "new-doc",   "rev": "1-15f65339921e497348be384867bb940f" }</pre>

# Crear documento sin \_id

— — —

Method	Address
POST	http://localhost:5984/my-database

Body	Response
<pre>{   "hello": "world" }</pre>	<pre>{   "ok": true,   "id": "e32f52806e996be9cbaf79db840207d4",   "rev": "1-15f65339921e497348be384867bb940f" }</pre>

# Leer documento

— — —

Method	Address
GET	http://localhost:5984/my-database/ <b>new-doc</b>

## Response

```
{
  "_id": "new-doc",
  "_rev": "1-15f65339921e497348be384867bb940f",
  "hello": "world"
}
```

# Eliminar documento

— — —

Method	Address
DELETE	http://localhost:5984/my-database/new-doc?rev=1-15f65339921e497348be384867bb940f

## Response

```
{  
  "ok": true,  
  "id": "new-doc",  
  "rev": "2-2d715caed35974bb33de24d1d6c95779"  
}
```

# Documentos de diseño


# Documentos de diseño

— — —

En las bases de datos pueden existir unos documentos especiales llamados **documentos de diseño**.

Estos documentos tienen parte del código de la aplicación y están escritos generalmente en Javascript aunque se puede utilizar casi cualquier lenguaje.

Los documentos de diseño contienen funciones que pueden ser de varios tipos, vamos a ver las más relevantes.



```

1 {
2   "_id": "_design/sofa", ← Determines the app URL
3   "_rev": "3157636749",
4
5
6   "language": "javascript", (for the web)
7
8
9   "validate_doc_update": "function (newDoc, oldDoc, userCtx) { ... }",
10                                     Application is stored as JSON data
11
12   "views": { ← Views field stores incremental
13     "comments": { map reduce functions
14       "map": "function(doc) { ... }";
15       "reduce": "function(keys, values, rereduce) { ... }";
16     }
17   },
18
19
20   "shows": { ← Shows functions transform
21     "post": "function(doc, req) { ... }"
22   },

```

# Funciones Show y List

---

Una función **show** permite devolver un documento con un formato determinado y una función **list** lo mismo pero con los documentos resultados de una vista.

Esto permite obtener resultados en cualquier formato como XML o CSV.

Anteriormente se aprovechaban estas funciones junto con la posibilidad de almacenar archivos para crear aplicaciones únicamente con CouchDB, eran conocidas como *CouchApps*, actualmente es algo fuertemente rechazado.

```
{db}/_design/{design-doc}/_show/{show-func}/{doc_id}  
{db}/_design/{design-doc}/_list/{list-func}/{view_name}
```



```
function(head, req){
  start({
    'headers': {
      'Content-Type': 'text/html'
    }
  });
  send('<html><body><table>');
  send('<tr><th>ID</th><th>Key</th><th>Value</th></tr>');
  while(row = getRow()){
    send('').concat(
      '<tr>',
      '<td>' + toJSON(row.id) + '</td>',
      '<td>' + toJSON(row.key) + '</td>',
      '<td>' + toJSON(row.value) + '</td>',
      '</tr>'
    ));
  }
  send('</table></body></html>');
}
```

# Funciones de actualización

---

Permiten ejecutar lógica del lado del servidor para crear o actualizar documentos, por ejemplo para añadir la fecha y hora o para incrementar un campo.

```
function(doc, req){  
  doc['world'] = 'hello';  
  doc['edited_by'] = req['userCtx']['name']  
  return [doc, 'Edited World!']  
}
```

Esta función crea un campo llamado 'world' y otro llamado 'edited\_by'. Devuelve un array con el documento actualizado y un mensaje deseado.

```
{db}/_design/{design-doc}/_update/{update-func}/{doc_id}
```

# Funciones de validación

— — —

Permite validar un documento antes de guardarlo en la base de datos.

Las funciones de validación serán llamadas cada vez que se cree o se actualice un documento.

Cómo parámetros reciben el documento a escribir, el usuario que lo escribe y el documento de seguridad de la base de datos.



# Funciones de Vista

---

Las vistas son la principal herramienta para hacer consultas de datos. Están formadas por dos funciones: **map** y **reduce**.

La función **map** recibe un documento emite claves de pares y valores.

La función **reduce** recibe una lista de documentos y devuelve un único valor.

CouchDB tiene incorporadas varias funciones reduce: sumatorio, número de resultados, máximo, mínimo y suma de los cuadrados.

```
{db}/_design/{design-doc}/_view/{view-name}
```

Vistas

# Vistas

— — —

Los campos emitidos por la función **map** son almacenados en un árbol-B, que es una estructura típica en computación, utilizada también por las bases de datos SQL para almacenar los índices.

Podemos pensar en los resultados de una vista como una tabla con una columna de claves y otra de valores, ordenada según las claves.

Al crear una vista se ejecuta una vez por cada documento de la base de datos, después se ejecuta cada vez que un documento es añadido o modificado.

# Vistas

---

```
function(doc) {  
  if(doc.date && doc.title) {  
    emit(doc.date, doc.title);  
  }  
}
```

Clave	Valor
"2009/01/15 15:52:20"	"Hello"
"2009/01/30 18:04:11"	"World"

Por ejemplo a partir de esta función podemos obtener una lista con los títulos de los documentos ordenados por fecha. Las vistas también almacenan el `_id` de los documentos, así que acceder al documento completo es trivial.

```
/blog/_design/doc/_view/by_date?include_docs=true
```

# Vistas

— — —

Las consultas se pueden restringir para una clave o para un rango de claves. También se pueden leer en orden inverso.

Una fecha determinada:

```
_view/by_date?key="2009/01/30 18:04:11"
```

Un rango de fechas:

```
_view/by_date?startkey="2010/01/01 00:00:00"&endkey="2010/02/00 00:00:00"
```

Orden inverso:

```
_view/by_date?startkey="2009/01/30 18:04:11"&descending=true
```



# Vistas

---

Las claves pueden ser cualquier tipo de datos válido de JSON, como arrays u objetos, lo que permite mayor control para la ordenación y agrupación de los resultados.

Aquí podemos ver los criterios que se siguen para la ordenación.

```
// special values sort before all other types
null
false
true

// then numbers
1
2
3.0
4

// then text, case sensitive
"a"
"A"
"aa"
"b"
"B"
"ba"
"bb"

// then arrays. compared element by element until different.
// Longer arrays sort after their prefixes
["a"]
["b"]
["b", "c"]
["b", "c", "a"]
["b", "d"]
["b", "d", "e"]

// then object, compares each key value in the list until different.
// larger objects sort after their subset objects.
{a:1}
{a:2}
{b:1}
{b:2}
{b:2, a:1} // Member order does matter for collation.
// CouchDB preserves member order
// but doesn't require that clients will.
// this test might fail if used with a js engine
// that doesn't preserve order
{b:2, c:2}
```

# Vistas

---

La claves complejas se pueden utilizar para la lectura de datos relacionados.

Con esta función podremos leer un post con todos sus comentarios. También se podría hacer más compleja para ordenar por fecha por ejemplo.

```
function(doc) {  
  if (doc.type == "post") {  
    emit([doc._id, 0], doc);  
  } else if (doc.type == "comment") {  
    emit([doc.post, 1], doc);  
  }  
}
```

Clave	Valor
['mypost', 0]	{ ...Post }
['mypost', 1]	{ ...Comentario }
['mypost', 1]	{ ...Comentario }
['other_post', 0]	{ ...Post }
['other_post', 1]	{ ...Comentario }

[/blog/\\_design/docs/\\_view/comments?startkey=\['mypost'\]&endkey=\['mypost', 2\]](/blog/_design/docs/_view/comments?startkey=['mypost']&endkey=['mypost', 2])

# Vistas

---

Se pueden agrupar los resultados de un reduce con el parámetro **?group=true**.

También se pueden paginar los resultados con los parámetros **?limit=x** y **?skip=y**.

```
{ "rows": [  
  { "key": "bike", "value": 1 },  
  { "key": "couchdb", "value": 3 },  
  { "key": "drums", "value": 1 },  
  { "key": "hypertext", "value": 1 },  
  { "key": "music", "value": 1 },  
  { "key": "mustache", "value": 1 },  
  { "key": "philosophy", "value": 1 }  
]}
```

Resultado de **\_sum** usando **group**

# Mango Queries

# Mango Queries

---

Para los usuarios que no estén familiarizados con las consultas **MapReduce** se creó una sintaxis inspirada en las consultas de MongoDB, llamada **Mango**.

Para usarla hay que crear índices utilizando la ruta **{db}/\_index**, para después ejecutar las consultas utilizando la ruta **{db}/\_find**.

Al crear un index realmente se crean vistas en un documento de diseño.



# Mango Queries

---

```
{
  "index": {
    "fields": ["foo"]
  },
  "name" : "foo-index",
  "type" : "json"
}
```

Cuerpo de una petición **POST** a **/\_index**.  
Se quiere crear un índice para el campo **foo**.

```
{
  "selector": {
    "year": {"$gt": 2010}
  },
  "fields": ["_id", "_rev", "year", "title"],
  "sort": [{"year": "asc"}],
  "limit": 2,
  "skip": 0,
  "execution_stats": true
}
```

Cuerpo de una petición **POST** a **/\_find**.  
Se quieren obtener los campos **\_id**, **\_rev**, **year** y **title** de los documentos con años mayores que 2010.

Seguridad

# Autenticación

— — —

- Dos tipos de usuarios: administradores y normales.
- Base de datos de autenticación (**\_users**)
- Dos métodos de autenticación:
  - Básica (`http://email:password@127.0.0.1:5984`)
  - Cookies (POST con usuario y contraseña a `/_session`)



# Base de datos de autenticación

— — —

Existe una base de datos especial **\_users** para gestionar los usuarios.

Solo los administradores pueden listar todos los usuarios, y cada usuario puede acceder únicamente a su propio documento, aunque existe la posibilidad de definir campos públicos.

El campo de roles solo lo puede modificar un administrador del sistema.

Para crear un usuario sólo hay que crear un documento con al menos el nombre y la contraseña en dicha base de datos y couchdb se encarga de encriptar la contraseña.

```
1 - {
2   "_id": "org.couchdb.user:prz.lester@gmail.com",
3   "_rev": "150-13976d53df70df9c840f9fb441a3aa79",
4   "name": "prz.lester@gmail.com",
5   "roles": [
6     "test-member",
7     "test-admin",
8     "my-workspace-member",
9     "my-workspace-admin"
10  ],
11  "type": "user",
12  "firstName": "Lester",
13  "lastName": "Perez",
14 - "workspaces": [
15    "test",
16    "my-workspace"
17  ],
18  "password_scheme": "pbkdf2",
19  "iterations": 10,
20  "derived_key": "01f3a71fe92de231d427834499c2fcb9036c0d95",
21  "salt": "2862146f1f1a75c1b8bec79d848af190"
22 }
```

# Autorización

---

Las bases de datos tienen un documento **\_security**.

Este documento tiene dos campos: **members** y **admins**, en ellos se definen los usuarios y/o roles que pertenecen a cada uno de los dos grupos.

Una importante limitación es que la seguridad solo se puede controlar para bases de datos completas, por lo que es habitual tener una base de datos para cada usuario.

```
{
  "admins": {
    "names": [
      "admin_1"
    ],
    "roles": [
      "db_admins"
    ]
  },
  "members": {
    "names": [
      "user_1"
    ],
    "roles": [
      "db_users"
    ]
  }
}
```

# Ecosistema



Servicio de base de datos en la **nube**.  
Crearon y donaron el sistema que permite  
ejecutar CouchDB como **clúster** y las **mango**  
**queries**. Incorporados en la versión 2.0.



Proyecto en el que trabaja actualmente el  
creador de CouchDB, resultado de la fusión de  
CouchDB y Membase, tiene muy poco que ver  
con CouchDB a pesar del nombre.  
**Tiene librerías nativas para Android e iOS.**



Implementación de CouchDB en Javascript.  
Permite la sincronización con bases de datos  
locales tanto en **navegador** como en **Node**,  
esto facilita el acceso a los datos sin conexión  
y en tiempo real.

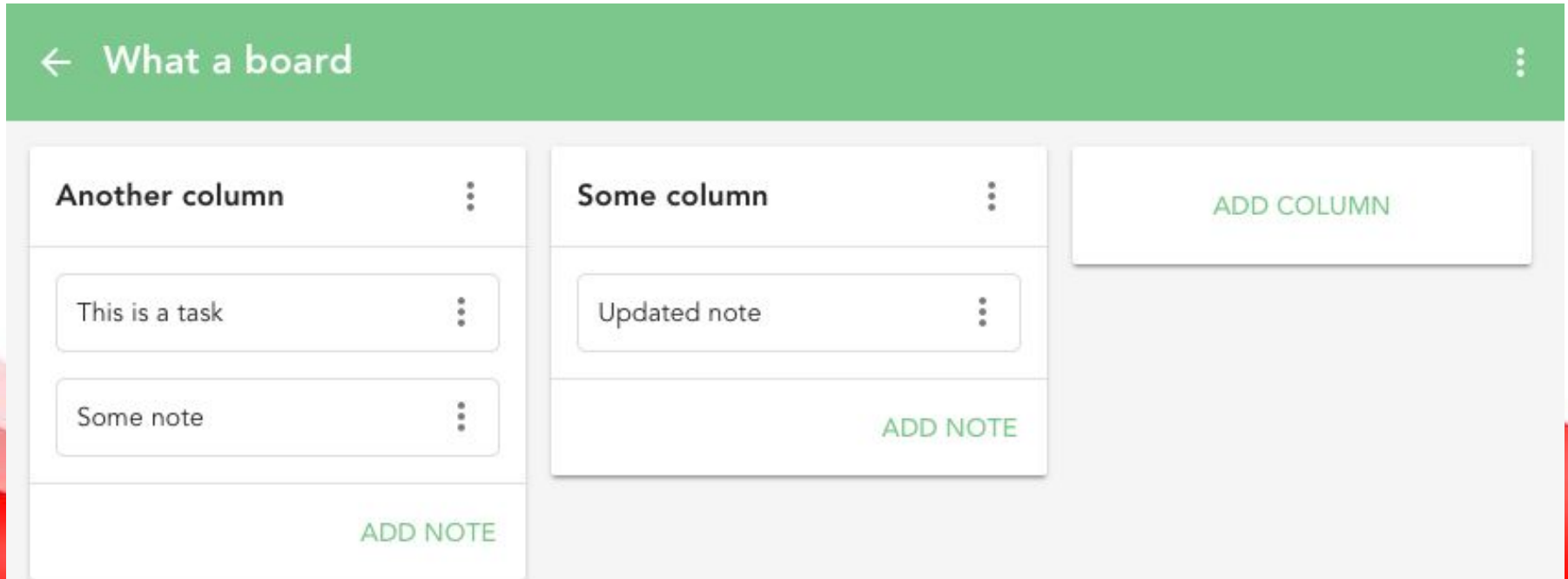
Todos comparten el mismo protocolo de replicación

# Caso Práctico: Task Manager

# Task Manager

- Front: <https://github.com/lesterbx/task-manager>
- Back: <https://github.com/lesterbx/task-manager-back>

Como caso práctico he construido una aplicación progresiva inspirada en Trello.



# Características

— — —

Los usuarios crean **espacios de trabajo** que pueden utilizar con otros usuarios.

Estos espacios de trabajo contienen **tableros**, que a su vez contienen **columnas**, que contienen **notas**. Las columnas y las notas se pueden arrastrar por el tablero.

La base de datos se almacena de forma offline en los dispositivos gracias a **pouchdb**, de esta forma los usuarios puedan trabajar también sin conexión.



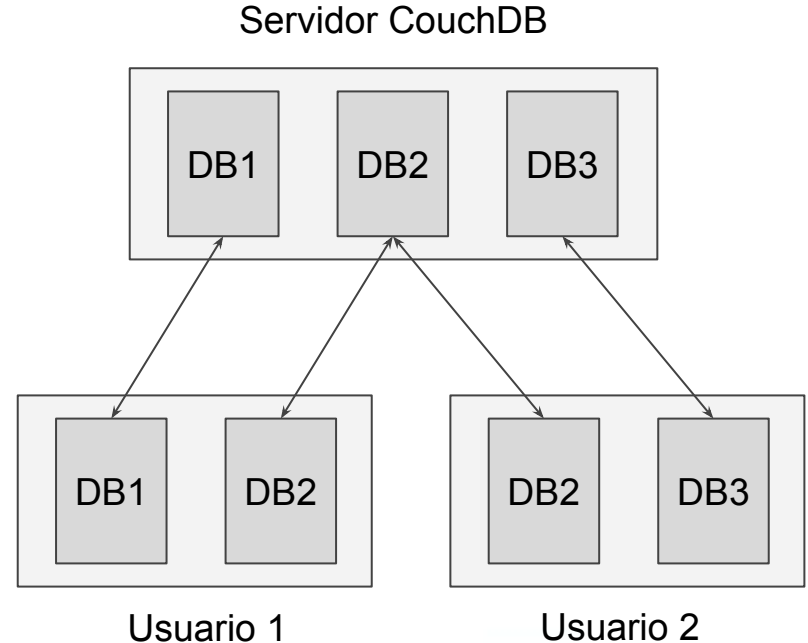


# Bases de datos

---

Para adaptarme a la limitación de seguridad sobre las bases de datos de CouchDB he estructurado la aplicación de forma que cada espacio de trabajo es una base de datos a la que todos sus miembros tienen acceso.

Esto también facilita la sincronización de las mismas para el acceso offline.

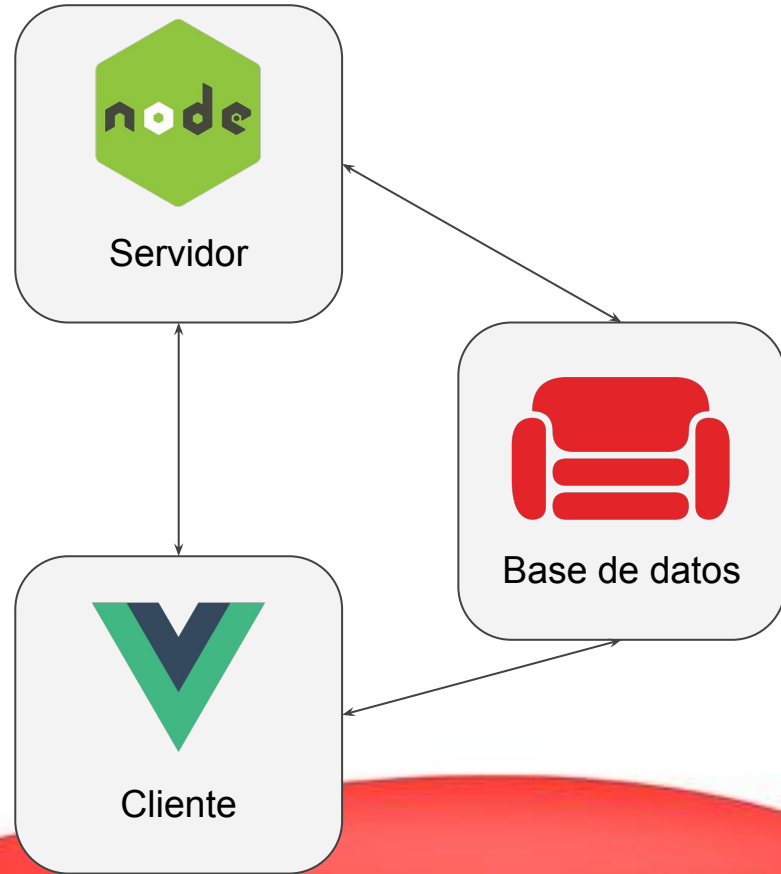


# Arquitectura

---

La aplicación consta de dos partes:

- **Ciente:** Desarrollado con Vue, se comunica directamente con la base de datos, salvo para algunas acciones que utiliza un servidor auxiliar.
- **Servidor:** Desarrollado con Node, sirve para crear espacios de trabajo, añadir o eliminar usuarios, y añadir o eliminar administradores.



# Documento de diseño

— — —

Las bases de datos para los workspace contienen un documento de diseño que se añade en el servidor al crear el workspace.

Este documento se llama **\_design/workspace**, y contiene una función de validación y una función para añadir un timestamp.

También contienen tres vistas, una para los tableros del workspace, otra para las columnas y notas de un tablero, y la última para la lista de notas de una columna.

```

1 {
2   "_id": "_design/workspace",
3   "_rev": "1-848d81c6f259b4cc93cc22df4a446be5",
4   "language": "javascript",
5   "validate_doc_update": "function (newDoc, savedDoc) {\n      if (!newDoc._deleted)
6   "updates": {
7     "timestamp": "function (doc, req) {\n      doc.timestamp = Date.now()\n      return
8   },
9   "views": {
10    "boards": {
11      "map": "function (doc) {\n      if (doc.type === 'board') {\n      emit(doc
12    },
13    "boardContent": {
14      "map": "function (doc) {\n      if (doc.type === 'column') {\n      emit(
15    },
16    "columnNotes": {
17      "map": "function (doc) {\n      if (doc.type === 'note') {\n      emit(doc.co
18    }
19  }
20 }

```

Documento de diseño del workspace

```

1 function (newDoc) {
2   if (!newDoc._deleted) {
3     function required(field, message) {
4       if (!newDoc[field] || newDoc[field] === '') {
5         throw ({ forbidden: message })
6       }
7     }
8     if (newDoc.type === 'workspace') {
9       required('title', 'Missing title')
10      required('users', 'Missing users')
11      required('picture', 'Missing picture')
12      required('admins', 'Missing admins')
13      required('description', 'Missing description')
14    } else if (newDoc.type === 'board') {
15      required('title', 'Missing title')
16    } else if (newDoc.type === 'column') {
17      required('title', 'Missing title')
18      required('boardID', 'Missing board id')
19      required('position', 'Missing position')
20    } else if (newDoc.type === 'note') {
21      required('text', 'Missing text')
22      required('columnID', 'Missing column id')
23      required('position', 'Missing position')
24    }
25  }
26 }
27

```

Función de validación

```

1 function (doc) {
2   doc.timestamp = Date.now()
3   return [doc, 'Timestamped document']
4 }

```

Función de actualización

```

views: {
  boards: {
    map: function (doc) {
      if (doc.type === 'board') {
        emit(doc.timestamp, null);
      }
    }
  },
  boardContent: {
    map: function (doc) {
      if (doc.type === 'column') {
        emit(doc.boardID, null)
      } else if (doc.type === 'note') {
        emit(doc.boardID, null);
      }
    }
  },
  columnNotes: {
    map: function (doc) {
      if (doc.type === 'note') {
        emit(doc.columnID, null)
      }
    }
  }
}

```

Leer todos los tableros:

`/_view/board?include_docs=true`

Columnas y notas de un tablero:

`/_view/boardContent?key=my-board&include_docs=true`

Notas de una columna:

`/_view/columnNotes?key=my-column&include_docs=true`

# Seguridad

— — —

Las bases de datos tienen usuarios normales y administradores, los administradores son los únicos que pueden añadir o eliminar usuarios, así como darles permisos de administrador o revocarlos.

Para esto se añaden los roles correspondientes a los usuarios, por eso hace falta el servidor auxiliar ya que los roles de usuario solo se pueden modificar desde una cuenta de administrador de CouchDB.

Para la autenticación he utilizado el plugin **pouchdb-authentication**



```
{
  admins: {
    roles: ['my-workspace-admin']
  },
  members: {
    roles: ['my-workspace-member']
  }
}
```

Documento **\_security** de un workspace

```
{
  _id: "org.couchdb.user:prz.lester@gmail.com",
  _rev: "3-d481bdeb37b0f37cec784774a617e604",
  type: "user",
  name: "prz.lester@gmail.com",
  firstName: "Lester",
  lastName: "Perez",
  roles: [
    "my-workspace-member",
    "my-workspace-admin"
  ],
  workspaces: [
    "my-workspace"
  ]
}
```

Documento de un usuario con los roles correspondientes para el workspace



# Resolución de conflictos

— — —

Para la resolución de conflictos he utilizado el plugin **pouch-resolve-conflicts**, con este plugin simplemente hay que crear una función para resolver los conflictos, que en mi caso es muy sencilla, conservo la actualización con el timestamp más reciente.

Para ello en el servidor escucho los cambios en todas base de datos y si hay conflictos los resuelvo con la función de resolución.



```
const listenDBConflicts = (dbname) => {  
  let db = new PouchDB(`${dbhost}/${dbname}`)  
  db.changes({  
    live: true,  
    include_docs: true,  
    conflicts: true  
  }).on('change', ({ doc }) => {  
    db.resolveConflict(doc, resolveConflict)  
  })  
}
```

```
const resolveConflict = (a, b) => a.timestamp > b.timestamp ? a : b
```

Resuelvo los conflictos en base al timestamp

Escucha de cambios en la base de datos

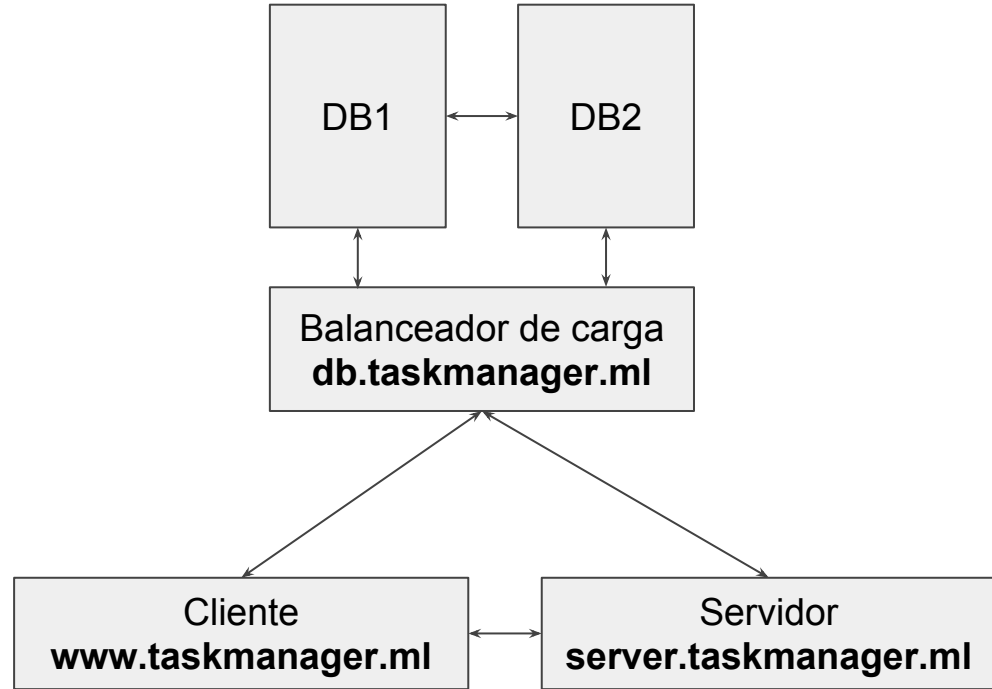
# Despliegue

---

La aplicación se ha desplegado en los **Servicios Web de Amazon**, utilizando el dominio gratuito **taskmanager.ml**.

Para la base de datos se han creado dos máquinas utilizando la configuración de clúster de CouchDB.

Las máquinas para la base de datos están detrás de un balanceador de carga.



# Conclusión

# Conclusión

— — —

Me ha parecido un sistema gestor muy fácil de usar, de hecho el lema es “**Relax!**”.

Debido a sus peculiaridades hay que utilizarlo para casos de usos apropiados, por ejemplo no es recomendable para datos profundamente relacionados.

Personalmente me ha gustado que se trabaje con documentos JSON y a través de HTTP, ya que prácticamente todos los lenguajes tienen librerías para esto.

Posiblemente la característica más valorada es la replicación, he llegado incluso a leer que si no fuera por la replicación CouchDB ya no existiría.

# Referencias

— — —

- Documentación oficial CouchDB  
(<http://docs.couchdb.org>)
- CouchDB, la guía definitiva  
(<http://guide.couchdb.org>)
- Tutorial para desplegar cluster de CouchDB en AWS  
(<https://hackernoon.com/running-a-couchdb-2-0-cluster-in-production-on-aws-with-docker-50f745d4bdbc>)
- 10 errores comunes con CouchDB (vídeo)  
(<https://www.youtube.com/watch?v=BKQ9kXKoHS8>)
- Relaciones con CouchDB  
(<https://wiki.apache.org/couchdb/EntityRelationship>)