

Escola Tècnica Superior d’Enginyeria  
Electrònica i Informàtica La Salle

Trabajo Fin de Máster

Máster en Programción Web de Alto Rendimiento

Alumno Profesor Ponente

Daniel Salgado Población Víctor Caballero Codina

ACTA DEL EXAMEN

DEL TRABAJO FIN DE MÁSTER

Reunido el Tribunal calificador en la fecha indicada, el alumno

D. Daniel Salgado Población

expuso su Trabajo Fin de Máster, titulado:

Social Access Controller

Acabada la exposición y contestadas por parte del alumno las objeciones formuladas por los Sres. miembros del tribunal, éste valoró dicho Trabajo con la calificación de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL

Social Access Controller

Universitat Ramon Llull – La Salle Barcelona

Daniel Salgado Población

1 de septiembre de 2019

# Resumen

Internet of Things (Iot) es la agrupación e interconexión de dispositivos y objetos a través de una red. Es uno de los anticipos tecnológicos más destacados en los últimos tiempos. Cada vez está más presente en el día a día y su aplicabilidad en el futuro está fuera de cualquier discusión.

Este nuevo marco anticipa varios retos, entre los que se suele destacar la accesibilidad y la manera de compartir (o compartición) de los objetos físicos conectados a la red llamados Web Things (*WT*). El W3Consortium propone un modelo común para describir contrapartes virtuales de los dispositivos físicos del Internet of Things.

Este trabajo replica un controlador basado en el Web of Things que permite compartir las contrapartes virtuales entre los diferentes amigos de una red social. Para ello y, al no disponer de dispositivos físicos, también se ha creado un software emulador de dispositivos del Web of Things.

Para abordar la compartición de los *WTs* hemos creado el **Social Access Controller (SAC)** hemos implementado un sistema para compartir Wts.esta seguridad y delegarla en redes ya construidas para esto, como puede ser Facebook. Es esta aplicabilidad la que hemos querido implementar SAC. Donde poder compartir con otras personas información de internet of things (Iot) .

# Abstract

Contenido

[Resumen 6](#_Toc18410921)

[Abstract 7](#_Toc18410922)

[Introducción 11](#_Toc18410923)

[Motivación 11](#_Toc18410924)

[Problemas de compartición de *WT* a resolver 11](#_Toc18410925)

[Requisitos 11](#_Toc18410926)

[Objetivos 12](#_Toc18410927)

[Diseño General 14](#_Toc18410928)

[Acople al modelo *WT* del W3Consortium 14](#_Toc18410929)

[Diseño de Software. Arquitectura Hexagonal 15](#_Toc18410930)

[Endpoints 17](#_Toc18410931)

[Lanzamiento Peticiones HTTP 18](#_Toc18410932)

[Diseño de Base de Datos 18](#_Toc18410933)

[Datos de prueba (Fixtures) 21](#_Toc18410934)

[Usuarios de prueba en Facebook 21](#_Toc18410935)

[WT de pruebas Iot\_emulator: 22](#_Toc18410936)

[Datos pruebas SAC 22](#_Toc18410937)

[Sadfasd 23](#_Toc18410938)

[Tests 24](#_Toc18410939)

[Iot\_emulator 24](#_Toc18410940)

[SAC 24](#_Toc18410941)

[Backend Iot\_emulator 25](#_Toc18410942)

[Estructura básica *WT* 25](#_Toc18410943)

[Esquema Base de datos Iot\_emulator 26](#_Toc18410944)

[Arquitectura REST Iot\_emulator 27](#_Toc18410945)

[Arquitectura hexagonal de Iot\_emulator 28](#_Toc18410946)

[Seguridad Iot\_emulator 30](#_Toc18410947)

[Frontend SAC 31](#_Toc18410948)

[Raíz del proyecto 31](#_Toc18410949)

[Mapa web para *Owner* 32](#_Toc18410950)

[Admin de un *WT* 33](#_Toc18410951)

[Ver listado de *Friends* 34](#_Toc18410952)

[Resultado de compartición 34](#_Toc18410953)

[Mapa web para *Friend* 34](#_Toc18410954)

[Página de Error y de Éxito 35](#_Toc18410955)

[Backend SAC 36](#_Toc18410956)

[Autenticación Delegada en Facebook 36](#_Toc18410957)

[Proceso de Login 36](#_Toc18410958)

[Crear Owner en SAC 38](#_Toc18410959)

[Crear Actions en SAC 38](#_Toc18410960)

[Acciones sobre los WTs 38](#_Toc18410961)

[API SAC 39](#_Toc18410962)

[Esquema Base de datos SAC 44](#_Toc18410963)

[Arquitectura REST SAC 46](#_Toc18410964)

[Arquitectura Hexagonal SAC 47](#_Toc18410965)

[SEGURIDAD 52](#_Toc18410966)

[Tecnologías usadas 54](#_Toc18410967)

[PHPStorm: 54](#_Toc18410968)

[Facebook 54](#_Toc18410969)

[Sistema operativo 54](#_Toc18410970)

[github 56](#_Toc18410971)

[aws ec1 56](#_Toc18410972)

[nginx 56](#_Toc18410973)

[html5 56](#_Toc18410974)

[javascript 56](#_Toc18410975)

[jquery 57](#_Toc18410976)

[Jquery-UI 57](#_Toc18410977)

[moustache 57](#_Toc18410978)

[symfony 4 57](#_Toc18410979)

[npm 57](#_Toc18410980)

[mysql 57](#_Toc18410981)

[Doctrine 57](#_Toc18410982)

[Análisis resultados 58](#_Toc18410983)

[Relación con Asignaturas del máster 58](#_Toc18410984)

[Frameworks 59](#_Toc18410985)

[Emprendiduría 59](#_Toc18410986)

[Entorno Web 59](#_Toc18410987)

[Seguridad 59](#_Toc18410988)

[Frontend 59](#_Toc18410989)

[PHP 59](#_Toc18410990)

[Bases de Datos 59](#_Toc18410991)

[Conclusiones 59](#_Toc18410992)

[Instalación 59](#_Toc18410993)

[---- INSTALATION SYSTEM REQUIREMENTS 59](#_Toc18410994)

[------- INSTALACION SAC 60](#_Toc18410995)

[------------ INSTALACION IOT\_EMULATOR 61](#_Toc18410996)

[Mejoras 62](#_Toc18410997)

[Glosario 63](#_Toc18410998)

[Bibliografía 64](#_Toc18410999)

# Introducción

No hace falta qe en introducción este los conceptos (o sí) Introducimos los coceptos. ES UN RESUMEN PERO MÁS LARGO

Internet of Things (Iot) es la agrupación e interconexión de dispositivos y objetos a través de una red. Estos objectos físicos conectados a la red tienen una representación virtual llamada Web Things (*WT*). En este trabajo nos hemos querido poner en práctica una manera en la que compartir las contrapartes virtuales de los dispositivos físicos *WT* entre diferentes amigos de una red social.

Hemos seguido la manera de compartir los *WT* sugerido en el trabajo [link]] mediante un Social Access Controller (SAC). Al compartir mediante SAC hemos conseguido compartir *Wts* con las siguientes características. SAC actúa de proxy de compartición permitiendo compartir de manera granulada ciertas partes de los *WTs*. SAC administra el acceso a los *WTs* basándose en la estructura de redes sociales ya existentes, como puede ser Facebook.

Al no disponer de dispositivos físicos, también hemos creado un software emulador de dispositivos del Web of Things. Hemos emulado la implementación de *WT* propuesta por W3Consortium [link]. En este modelo un *WT* tiene *Properties* son una propiedad interna del *WT* y también posee *Actions* que representa una función del *WT.*

En la parte más humana hemos modelado un dueño (*Owner)* como la persona que conoce las credenciales de los *WTs* y tiene una red de personas de su confianza (*Friends)* en Facebook. El *Owner* puede decidir, granularmente cuales *Actions* muestra a cada *Friend*. SAC. Al decidir a quien comparte se genera una URL única que será la que el *Owner* comparta con el *Friend*.

SAC almacena el mínimo posible de información. La información se encuentra en Facebook y en los propios *WTs*. Cuando un *Friend* con permisos consulta el resultado de una *Action*  SAC actúa de proxy entre el *Friend* y el *WT*. SAC usa las credenciales del *Owner* para acceder al *WT* y traer la respuesta.

## Motivación

Actualmente los *Owners* de *WT* no poseen una manera segura y homogénea de compartir *Actions* con sus *Friends*. El objetivo es proporcionar un sistema seguro y eficiente donde compartir *Actions* de *WT* con sus *Friends*.

Para poder compartir *WT* con *Friend* el *Owner* tiene las siguientes soluciones disponer de un *WT* multiusuarios o compartir las credenciales con *Friends*.Ambas soluciones carecen de la posibilidad de compartir granularmente los permisos del *WT* ya que el *Owner* se ve obligado a compartir todo el *WT* en su conjunto. A continuación explicamos de qué tratan ambas soluciones.

### WT multiusuario

Un *WT* multiusuario es aquel que puede tener varios usuarios distintos con permisos de acceso. Es una manera para que un *Owner* pueda dar acceso a un *Friend* al *WT.* Esta solucion tiene varios problemas. Primero el *Owner* debe conocer la manera de dar de alta un usuario nuevo a su *WT*. Todos los *WT* deben ser multiusuario.

### Compartir Credenciales

Si un *Owner* cede sus credenciales con *Friend* este será capaz de ver *Wt.* En este caso los problemas que *Owner*  se encuentra son que el debe confiar en la buena fe del *Friend* ya que pierde el control de la credencial. *Owner* debe llevar de alguna manera el control de a quién dejó cual WT

## Requisitos

Para resolver las anteriores problemáticas proponemos diseñar una aplicación. Que permita compartir granularmente los distintos *WTs* a distintos *Friends* de una manera sencilla y segura.

1. Un *Owner* tiene un lugar único donde administrar la compartición de *WTs* poseídas.
2. La aplicación debe impersonar al *Owner* en el *WT*, esto es logarse en *WT* con las credenciales previamente almacenadas del *Owner*.
3. Compartición granular. Un *Owner* puede compartir indistintamente distintos *Actions* de un mismo *WT*.
4. Las *Actions* compartidas puedes serlo a número indeterminado de *Friends*.
5. La aplicación almacena la cantidad mínima e imprescindible de información. La estructura de red de amigos se almacena externamente a la aplicación. La información de *Actions* y *Properties* se consultan al propio *WT*.

## Objetivos

El objetivo primario del Trabajo Fin de Máster es crear una aplicación que cumpla los requisitos anteriormente mencionados. Y así poder evaluar en un sistema activo la compartición granular, eficiente, segura y fácil *WTs* de un *Owner* a sus *Friends*.

### Compartición mediante Social Access Controller (SAC)

Siguiendo la idea de un SAC propuesta por Dominique Guirnard *et al* en [LINK1] básandonos en su idea de “Compartición basada en Redes Sociales” nos fijamos los siguientes objetivos.

Autenticacion delegada (TODO ¿explicar autenticación delegada? En cualquier caso cambiar de sitio esto)

La autenticación y el reconocimiento de usuarios se hace mediante “autenticación delegada” de Facebook [link]. Este mecanismo nos permite identificar inequívocamente a cada usuario. Este token también se usa para reconocer a cada *Friend*.Es asi como SAC distingue cada tipo de usuario. almacena un token único y persistente provisto por Facebook exclusivo del *Owner*

El *Owner* accederá a SAC usando autenticación delegada de Facebook. Posteriormente el *Owner* da de alta los *WTs* en SAC en esta alta debe proporcionar las credenciales para acceder al *WT*. Cuando SAC accede a la info de un *WT* muestra degranadamente cada *Action* del *WT*. SAC permite compartir cada una de las *Actions* individualmente. Una vez que un *Action* ha sido compartida el *Friend* es capaz de ver el resultado del *Action*.

### Cumplimiento de requistios

Me parece demasiao redundante todo esto

Vamos a analizar como SAC cumple con los requisitos expuestos en pág 12

Cumplimos el requisito 1 al administrar la compartición desde la aplicación única llamada SAC. Al guardar las credenciales al dar de alta el *WT* en SAC cumplimos el requisito 2. Como SAC usa las credenciales del *Owner* puede ver todos los *Actions* y mostrar solo aquellos recursos a los que el *Owner* haya dado el permiso, cumpliendo asi el requistio 3. La explicación de cómo cumplir el requisito 4 queda más clara al explicar la estructura de base de datos en la pág 44 al ver como existe una relación N:M entre action y friend. El requisito 5 de almacenar la mínima información posible se cumple en la parte del *WT* al consultar sus *Actions* en caliente y la parte de Facebook porque es Facebook quien nos provee de la estructura de red social, para más detalle se pude consultar en pág 45 al ver los datos almacenados en SAC.

### Capa de accesibilidad mediante Iot Emuator

Para poder desarrollar SAC vimos la necesidad de tener tanto *WTs* como una capa que los conectase entre ellos. Para ello hemos creado una aplicación que funciona a modo de hub de *WTs*. Esta aplicación la hemos llamado I*ot\_emulator*. Se apoya en el modelo de *WT* propuesto por W3Consortium [LINK2]. Emula la capa web por encima de los *WTs* y hace que estos sean homogéneos. En los siguientes apartado explicamos el nivel de acople llegado en comparación con el propuesto por W3Consortium.

# Diseño General

En esta sección englobamos, a modo de introducción aquel conocimiento que es compartido por Iot\_emulator como por SAC de esta manera evitamos repetirnos. Para facilitar la lectura en las secciones específicas referenciamos hacia esta sección general.

En la siguientes secciónes comenzamos hablando del nivel de acople o ajuste de las referencias bibliográficas Para llevar a cabo el Trabajo Fin de Máster hemos simplificado los modelos propuestos tanto por [LINK1] y [LINK2]. Estas simplificaciones las hemos hecho por restricciones de tiempo y por no añadir complejidad innecesaria. Explicamos qué partes sí hemos construido y explicar cuáles no. Explicamos los dos diseños de software usados en ambos proyectos. Primero hablamos de la arquitectura hexagonal; sus capas y centrándonos en capa Infrastructura para mirar qué componentes se han creado. Segundo el servicio REST donde explicaremos tanto los endpoints generados como las herramientas usadas para sus respectivas APIs. En la parte de Diseño de Base de datos nos fijaremos en las similitudes y diferencias de ambos proyectos.

## Acople al modelo *WT* del W3Consortium

Explicamos W3 Consortium proponer una serie de requrimientos y modela qué y cómo debe ser una *WT*. En el documento que nos ha servido de referencia determina en punto 4. La integración Fija tanto los requests que deben comunicarse hacia ellos como los responses. Debido a la naturaleza de este trabajo fin de máster no hemos creado un Iot\_emulator completo. A continuación explicamos en qué medida nos hemos acoplado.

Nosotros hemos adaptado el model de 2015. <https://www.w3.org/Submission/2015/SUBM-wot-model-20150824/#web-things-requirements>

### Patrón de Integración de los *WT* en W3Consortium

Mientras que W3Consortium habla de 3 maneras de conectividad; Directa, *Gateway*, o *Cloud*. En siguiente Trabajo Fin de Máster. Al crear una capa de accesibilidad con un Iot\_emulator no podemos acoplarnos específicamente a ninguna de ellas. Explicamos esto por que como veremos (TODO link a REST) veremos en la pág pág 29 el impacto que tiene esto a la hora de definir las rutas REST de la la.

### Requerimientos para un *WT* en w3Consortium

A continuación, explicamos de qué manera nos hemos adaptado los requistios reflejados en [LINK2] en apartado *5. Web Thing requirements*

Para las tabla

* La columna “Nivel de cumplimiento” describe en el grado que hemos cumplido (…) en caso de no cumplirse explicamos brevemente la razón de dicha carencia.
* La columna “Definición de requisito” () TODO

#### Nivel 0 – Un WT DEBE

|  |  |
| --- | --- |
| Definción de requisito | Nivel de cumplimiento |
| A Web Thing MUST at least be an HTTP/1.1 server | No. Usamos un único servidor con iot\_emulator para simular todos los Web Things |
| A Web Thing MUST have a root resource accessible via an HTTP URL | Sí |
| A Web Thing MUST support GET, POST, PUT and DELETE HTTP verbs | Parcialmente, todos menos DELETE |
| A Web Thing MUST implement HTTP status codes 200, 400, 500 | Sí |
| A Web Thing MUST support JSON as default representation | Sí |
| A Web Thing MUST support GET on its root URL | Sí |

#### Nivel 1 – WT DEBERÍA

|  |  |
| --- | --- |
| Definción de requisito | Nivel de cumplimiento |
| A Web Thing SHOULD use secure HTTP connections (HTTPS) | Sí |
| A Web Thing SHOULD implement the WebSocket Protocol | No |
| A Web Thing SHOULD support the Web Things model | Sí |
| A Web Thing SHOULD return a 204 for all write operations | Sí |
| A Web Thing SHOULD provide a default human-readable documentation | No |

#### Nivel 2 – WT PODRÍA

|  |  |
| --- | --- |
| Definción de requisito | Nivel de cumplimiento |
| A Web Thing MAY support the HTTP OPTIONS verb for each of its resources | Sí |
| A Web Thing MAY provide additional representation mechanisms (RDF, XML, JSON-LD) | No |
| A Web Thing MAY offer a HTML-based user interface | No |
| A Web Thing MAY provide precise information about the intended meaning of individual parts of the model | No |

Tenemos un alto grado de adaptación con los requisitos de Nivel 0 – DEBE. Los 2 puntos que no adaptamos son debido a:

* Al emular n *WT* dentro de *iot\_emulator* fno son HTTP/1.1 servers
* No soportamos Verbo HTTP DELETE ya que esta funcionalidad la asume SAC

## Acople al SAC de Dominique Guinard

Realmente el SAC creado en este Trabajo Fin de Máster cumple es similiar al propuesto por Dominique Guinard. Por aún quedan estas partes no hechas. Nuestro SAC está muy acoplado a Facebook lo ideal sería que pudiera usarse otra red social para obtener las relaciones de confianza. Otra mejora es la hora de descubrir *WTs* Guinard propone que el SAC lo haga de manera automática rastreando la red. Otras limitaciones, como compartir *Actions* por verbo HTTP, son debidas al acople al modelo *WT* del W3Consortium.

## Diseño de Software. Arquitectura Hexagonal

El código de SAC y de iot\_emulator se ha hecho siguiendo una arquitectura hexagonal. Construyendo las siguientes capas **Dominio, Aplicación e Infraestructura**. Permitiendo desacoplar la lógica de cada capa. Cada capa corresponde con una carpeta en la raíz del proyecto. Cada capa depende únicamente de la capa que tiene por debajo.

Las capas se comunican entre ellas de fuera hacia adentro con adaptadores. Siguiendo un patrón de diseño llamado “Port and Adapters”. Donde las interfaces son los “ports” y las implementaciones los “Adapters”. La capa interna ofrece “ports” que son implementados por la capa más externa en sus “Adapters”. Las capas internas no conocen las capas externas.

### Dominio

Es la capa más interna. Las entidades son objetos php que están libres de logicas como por ejemplo saber como deben representase o cómo guardarse. Para poder interactuar con las Entidades tenemos los repositories que derminan la manera de usarse, son Interfaces que sólo incluyen lógica semántica. Es la capa que

Hemos definido dos namespaces; Entity y Repository. En Entity están los objetos puros y en Repository los interfaces.

### Aplicación

Es la capa por encima del Dominio. Hacia dentro implementa los adaptadores para el Dominio. Se comunica hacia la capa de Infrastructura definiendo cómo debe ser esa comunicación. Adapta las peticiones que vienen de infrastructura.

Hemos definido dos namespaces Command y CommandHandlers. Los Commands son DTOs que transportan los datos los CommandHandlers son orquestadores, todos tienen un método llamado *handle* que recibe el Command con los datos. Además los CommandHandlers reciben por constructuror las dependencias evitando asi el acoplamiento.

Cualquier tipo de input que reciba la aplicación va a acabar en los CommandHandlers.

### Infrastructura

Es la capa más externa, es donde vive el framework usado o el Sistema de base de datos. Por lo que suele tener muchas librerias de terceros. No hay lógica propia de tu aplicación.

#### Controladores

Procesan la Request, llaman al CommandHandler (capa de Aplicación) y construyen una Response. Los Controladores nunca se encargan de la lógica de negocio.

#### Respositorios

Las implementaciones de los repositorios responden a sus correspondientes contratos de la capa de Dominio. Esto permite que sea fácil cambiar implimetnaciones de reposiotrios usando diferentes tecnologías (XX) sin tener que cambiar el código de dominio.

#### Serializadores

Para desacoplar la serialización y deserialización de los datos de otros componentes de la capa de infraestructura como los Controladores, se han creado diferentes clases dedicadas a este propósito. Las clases dentro del namespace *Serializer* definen la lógica para serializar y deserializar una entidad específica

#### Comandos Symfony

En capa de Infrastructura hemos hecho uso de Comandos Symfony que usan Commandos y CommandHandlers de la capa de aplicacion. Teniendo en cuenta que, como veremos más adelante, en SAC existen zonas restringidas a tener una Autenticación Delegada de Facebook, la manera más cómoda de desarrollar y probar ciertos Commands y CommandHandlers han sido los Comandos Symfony.

Se ejecutan desde la raíz del proyecto desde la terminal. Todos tiene el prefijo **app** (dos puntos) seguidos del nombre del **dominio al que apliquen** (dos puntos) seguidos del **Commando de la aplicación** que ejectuan

php bin/console app:Dominio:Commando

#### Ejemplo de arquitectura hexagonal: ORM Doctrine

Un ejemplo es la manera en que desacacoplamos las capas es el uso de Mysql y más en concreto con el ORM Docnçtrine. En Dominio Las entidades se definene como entidades puras y ofrecen interfaces en los Repsitorios para interactuar sobre ellos. La capa de aplicación implementa estos interfaces y además usa el ORM Doctinre. A los Commands y CommandHandlers de la capa de Aplicación se les inyecta los repositorios con la implementación usando Mysql. Es decir, se podría cambiar el tipo de capa de infraestructura por otro ORM incluso otro framework de manera que las capa de Dominio y Aplicación se mantuviesen intactas.

## Diseño de Software. API REST

API REST es un interfaz entre sistemas que usa HTTP para obtener datos. Es un protocolo clinte/servidor sin estado. Usa los verbos HTTP y URI para obtener o manipular datos. Como vermos más adelante hemos construido 2 API REST una para Iot\_emulator y otra para SAC. Se puede encontrar una explicación detallada en cada secciones de arquitectura de Iot\_emulator (Arquitectura REST Iot\_emulator0 más adelante) como del SAC(Arquitectura REST SAC). Ahora vamos a mostrar un listado de los endpoint generados.

### Endpoints

Los endpoints son el final del canal de comunicación entre dos sistemas. En las siguientes tablas cada columna tiene este significado: Verbo HTTP: método de petición para iniciar la acción indicada. POST, GET, PUT, DELETE. Endpoint: interfaz expuesta vía URL

#### Iot\_emulator

Se puede consultar el detalle en esta sección (Arquitectura REST Iot\_emulator0 más adelante)

|  |  |
| --- | --- |
| Verbo HTTP | Endpoint |
| GET | / |
| GET | /{id} (sin credenciales) |
| GET | /{id} (con credenciales) |
| POST | /create |
| GET | /{id}/actions/{action\_name} |
| GET | /{id}/properties/{property\_name} |
| GET | /{id}/actions |
| GET | /{url} |

#### SAC

Se puede consultar el detalle en esta sección (Arquitectura REST SAC)

|  |  |
| --- | --- |
| Verbo HTTP | Endpoint |
| GET | / |
| GET | /loginOk |
| GET | /api/owner |
| GET | /api/thing/{thingId} |
| POST | /owner/share/action/{actionId}/friend/{friendId} |
| GET | /api/url/provider/thing |
| GET | /api/url/provider/api/thing |
| GET | /api/url/provider/api/share/action |
| GET | /privacy |
| GET | /conditions |
| GET | /error |
| GET | /friend/thing/{thingId}/action/{actionId} |
| GET | /friend |
| GET | /owner |
| GET | /owner/things |
| POST | /owner/create |
| GET | /success |
| POST | /thing/create |
| GET | /thing/{thingd} |

### Lanzamiento Peticiones HTTP

A la hora de desarrollar una API vimos la necesidad de documentar las peticiones. Asi que en cada proyecto creamos la carpeta.

docs/request

Y para lanzar las peticiones que hay dentro usamos tres herramientas distintas; Curl, Cliente HTTP de phpstorm y Httpie.

El inconveniente del cliente HTTP de phpstorm es la necesidad de phpstorm para lanzar las peticiones algo que en la máquina de producción no disponíamos. El inconveniente de Curl es que su sintaxis no es tan limpia como httpie. Por esto acabamos usando y reescribiendo las peticiones de docs/request en formato httpie.

## Diseño de Base de Datos

### Similitudes entre Iot\_emulator y SAC

En ambos proyectos usan el ORM Doctrine para relacionar entidades de base de datos con obejtos PHP. Y lo hacemos con arquitectura hexagonal (TODO ejemplo arquitetura hexagonal: Orm doctrine)

### Diferencias entre Iot\_emulator y SAC

Mientras que en Iot\_emulator hemos usado los @annotations como manera de definer las relaciones entre entidades. En SAC hemos usado ficheros .yaml.

#### Uso de @annotations en Iot\_emulator

Ejemplo de @annotation sacado de src/Domain/Entity/Property.php de Property del Iot\_emulator. Podemos ver las anotaciones que definen la Clase repositorio o la realción 1:1 existente entre esta entidad y App\Domain\Entity\Action.

<?php  
  
namespace App\Domain\Entity;  
  
use Doctrine\ORM\Mapping as ORM;  
  
*/\*\*  
 \** ***@ORM\Entity****(repositoryClass="App\Repository\PropertyRepository")  
 \*/*class Property  
{  
 */\*\*  
 \** ***@ORM\Id****()  
 \** ***@ORM\GeneratedValue****()  
 \** ***@ORM\Column****(type="integer")  
 \*/* private $id;  
  
 */\*\*  
 \** ***@ORM\Column****(type="string", length=255)  
 \*/* private $value;  
  
 */\*\*  
 \** ***@ORM\OneToOne****(targetEntity="App\Domain\Entity\Action", inversedBy="property", cascade={"persist", "remove"})  
 \** ***@ORM\JoinColumn****(nullable=false)  
 \*/* private $idAction;

#### Uso de ficheros .yaml en SAC

Ejemplo de uso de fichero .yaml sacdo de src/Infrastructure/Resources/mappings/Thing.orm.yml del SAC. Ofrece la misma información que el fichero del Iot. Pero al estar metido en capa Infrastructura deja la Entidad más limpia e independiente.

App\Domain\Entity\Thing:  
 type: entity  
 table: thing  
 id:  
 id:  
 type: integer  
 scale: 0  
 length: null  
 unique: false  
 nullable: false  
 precision: 0  
 id: true  
 generator:  
 strategy: IDENTITY  
 fields:  
 root:  
 type: string  
 scale: 0  
 length: 255  
 unique: false  
 nullable: false  
 precision: 0  
 user:  
 type: string  
 scale: 0  
 length: 255  
 unique: false  
 nullable: false  
 precision: 0  
 password:  
 type: string  
 scale: 0  
 length: 255  
 unique: false  
 nullable: false  
 precision: 0  
 oneToMany:  
 actions:  
 targetEntity: App\Domain\Entity\Action  
 cascade:  
 - remove  
 fetch: LAZY  
 mappedBy: thing  
 inversedBy: null  
 orphanRemoval: true  
 orderBy: null  
 manyToMany:  
 owners:  
 targetEntity: App\Domain\Entity\Owner  
 cascade: { }  
 fetch: LAZY  
 mappedBy: things  
 inversedBy: null  
 joinTable: { }  
 orderBy: null  
 lifecycleCallbacks: { }

# Datos de prueba (Fixtures)

Para facilitar la hora de desarrollar y probar hemos estandarizado unos datos de pruebas. Hemos diseñado estos datos a los tres niveles involucrados en proyecto

* Facebook
* SAC
* Iot\_emulator

## Usuarios de prueba en Facebook

Usando developers.facebook.com hemos creado una red de amigos con perfiles ficticios. Hemos decidido que “Eizabeth San Segundo de la Torre” será *Owner* en SAC.

Como se ve en parte inferior derecha en [Ilustración 1 Usuarios de prueba definidos en FacebookIlustración 1]. Hemos creado tres perfiles ficticion que funcionan como *Friends* de nuestra *Owner* Elisabeth.

* Linda De las Mareas
* Susan
* Mary



Ilustración 1 Usuarios de prueba definidos en Facebook

Tal como muestra en [Ilustración 2]. Elisabeth tiene dos *Friends* una llamada Linda con la que **sí** **comparte** *WT*s y otra llamada Mary con la que **no comparte** *WTs*.



Ilustración 2 Compartición entre amigas de Owner

Las comparticiones de *WT* entre Elizabeth y Linda las hemos creado usando interfaz web directamente en SAC. El detalle de cómo crearlas está explicado en [TODO LINK a creación de comparticiones]

## WT de pruebas Iot\_emulator:

Para todos los *WT* usados en pruebas el usuario esperado en tests es “user” y la contraseña es “password”.

Hemos creado script fixture/create\_things.php que mediante peticiones POST puebla la base de datos de iot\_emulator con things.

Contruyendo esta estructura incremental. Nótese que cada nuevo id incrementa el número de action y properties. Asi el thing n tiene n actions y n properties

{"name":"thing\_name1","brand":"thing\_brand1","links":{"actions":["action\_name1"],"properties":[{"action\_name1":"property\_value1"}]}}

{"name":"thing\_name2","brand":"thing\_brand2","links":{"actions":["action\_name1","action\_name2"],"properties":[{"action\_name1":"property\_value1"},{"action\_name2":"property\_value2"}]}}

{"name":"thing\_name3","brand":"thing\_brand3","links":{"actions":["action\_name1","action\_name2","action\_name3"],"properties":[{"action\_name1":"property\_value1"},{"action\_name2":"property\_value2"},{"action\_name3":"property\_value3"}]}}

## Datos pruebas SAC

Hemos populado la base de datos de SAC con herramienta propia de doctrine para popular sus bases de datos que son las Fistures. Se lanza con este Comando de Symfony

php bin/console doctrine:fixture:load

El código se puede encontrar en

src/DataFixtures/Sac.php

Queremos destacar un grave inconveniente que tiene el uso de Fixtures. Puede ocurrir que haya datos que una vez rellenos por estos Fixtures den falsos positivos y la sensación de que el desarrollo esté mejor de lo que realmente está. Nos pasó que durante el desarrollo tanto de Iot\_emulator como del SAC al tener cargados los datos datos falsos tal como estaban diseñados los *Actions* dentro del SAC correspondían con loso *Actions*  del Iot\_emulator. No fue hasta hacer pruebas sin esto datos que nos nos dimos cuenta de la incapacidad de identificar inequívocamente el *Action* compartidos.

# Tests

## Iot\_emulator

Hemos hecho 2 tipos de tests.

* Basados en phpunit
* Hecho en PHP

en php para determinar que exista la estructura recursiva explicada en anterior sección.

### no usan phpunit

php tests/notPHPUnit/get\_actions/get\_actions.php

php tests/notPHPUnit/get\_thing/get\_thing.php.php

php tests/notPHPUnit/isIntegrityValidOnCreate.php

### usan phpunit

TODO configurar para ejecutar phpunit

## SAC

# Backend Iot\_emulator

Se puede acceder a la API con desde esta URL: <http://iot.socialaccesscontroller.tk/>. Hemos usado symfony4 para crear un API con arquitectura REST y estructura de datos JSON. Esta API emula la capa de conexión entre los *WT* emulados nos permite disponer de *WT* para compartirlos con el SAC

## Estructura básica *WT*

Nuestro *WT* tiene características propias no expuestas en [LINK2] W3Consortium. Aquí explicamos la diferencia.

### Zona pública y Zona privada

Para diferenciar peticiones hechas por parte del SAC impersonando al *Owner* y para mostrar un listado de todos los *WTs* emulados en iot\_emulator [LINK-externo-al-punto-5.1.6 R0.6 – A Web-thingMUST support GET on its root URL]

. Hemos visto la necesidad de crear dos zonas diferentes en los *WT*.

* Zona pública. Se accede a ella sin credenciales, directamente con un GET al endpoint. Muestra *WT-*brand y *WT-*name
* Zona privada. Resto de endpoints del *WT.* Se necesita enviar *user* y *password* correctos del *WT*.

### Respuesta JSON

Una vez cargados los [LINK-interno al Datos de prueba] dadas las credenciales correctas. Esta es petición GET con credenciales correctas.

http://iot.socialaccesscontroller.tk/1

Esta es la respuesta. Como vemos es formato JSON y se pueden ver la Zona Pública como la Zona Privada del *WT*.

{

"id": 1,

"name": "thing\_name1",

"brand": "thing\_brand1",

"links": {

"actions": {

"link": "\actions",

"resources": {

"action\_name1": {

"values": "property\_value1"

}

}

}

}

}

### Relación entre Actions y Properties

Mientras que en un *WT* de W3Consortium *Property* es un estado del *WT* y *Action* desencadena una función, es decir son independientes, en nuestro modelo están fuertemente acopladas. Para simplificar el desarrollo hemos hecho que el nombre de la *Property* coincida con el valor de *Action*. Siguiendo el ejemplo anterior [LINK al código anterior]

url/actions/action\_name1

Devolvería

property\_value1

Es decir, nuestros *Actions* son punteros a los valores de *Properties*.

### *WT* Endpoints

Los endpoints de los *WT* equivalen a las claves primarias de la tabla thing de la base de datos. Los *WTs* se identifican con ids numéricos que coinciden con el id interno de la base de datos. Así thing\_1 sera /1 y será el id con pk=1 en tabla thing.

## Esquema Base de datos Iot\_emulator

En la X se muestra el esquema de base de datos que da soporte a la aplicación iot\_emulator. El usuario para acceder a esta base de datos debe tener estos permisos: DDL ALTER, AND DML SELECT, INSERT, UPDATE, DELETE .

Sin dudas la tabla más importante es Things, es la entidad que engloba a las demás. Relacionado con



Ilustración 3. Esquema Iot\_emulator.

## Arquitectura REST Iot\_emulator

TODO explicar mejor cada endpoint

La url /create se usa por emuldor (TODO) y rompe REST-

La url / que no está c

|  |  |  |
| --- | --- | --- |
| Verbo HTTP | Endpoint | Descripción |
| GET | / | Index de las zonas públicas de los *WTs* |
| GET | /{id} (Sin credenciales) | Acceder a zona Pública de *WT* |
| GET | /{id} (Con credenciales) | Acceder a zona Privada de WT |
| POST | /create | Crear un *WT* |
| POST | /{id}/actions/{action\_name} | Ejecutar un *Action* |
| GET | /{id}/properties/{property\_name} | Acceder a una *Property* |
| GET | /{id}/actions | Listado de *Actions* |

### Endpoint

#### GET /

Este endpoint es propio de nuestro SAC. No está definido ni en [LINK1] ni [LINK2] y rompe la naturaleza REST del API. Existe por la naturaleza de Hub que posee el Iot\_emulator.

#### GET /{id} Sin credenciales

Accedes a la zona pública del *WT*.

### ¿COMO ACCEDER A LISTADO IOTS?

La raiz de '/' iot\_emulator muestra una lista de la parte pública de todos los iots almacenados

## Arquitectura hexagonal de Iot\_emulator

### Dominio Iot\_emulator

Explicación de esta capa en pág 17.

Poner que entidades hay: en relación a la base de datos y Doctirne se han usado annotacione, se ve q esta acoplado pero que aunque no sigue la teoria se usa en muc

src/Domain/Entity/Action.php

src/Domain/Entity/Property.php

src/Domain/Entity/Thing.php

src/Domain/Entity/User.php

src/Domain/Repository/ActionRepository.php

src/Domain/Repository/PropertyRepository.php

src/Domain/Repository/ThingRepository.php

src/Domain/Repository/UserRepository.php

### Apliación Iot\_emulator

Explicación de esta capa en (17).

Y además usamos este patrón de diseño para transmitir la información de las credenciales recibidas, es una estructura de datos independiente a nuestro modelo de datos y solo contiene datos y ninguna lógica. Además hemos usado un DTO específico para encapsular las credenciales de *WT*.

src/Application/Dto/UserCredentialsDto.php

#### Estructura de Command y CommandHandlers Iot\_emulator

Existe una relación 1 a 1 entre todos los Commands y sus CommandHandlers. La [Lista de Código X] muestra un listado de Comandos y CommandHandlers respectivamente.

// Commands

src/Application/Command/Thing/CreateThingCommand.php

src/Application/Command/Thing/ExecuteActionCommand.php

src/Application/Command/Thing/SearchThingByIdCommmand.php

// CommandHandlers

src/Application/CommandHandler/Thing/CreateThingHandler.php

src/Application/CommandHandler/Thing/ExecuteActionHandler.php

src/Application/CommandHandler/Thing/SearchThingByIdHandler.php

### Infrastructura Iot\_emulator

Tenemos una explicación de la capa de Infrastructura en la (pág 17).

#### Controladores Iot\_emulator

Tenemos una explicación de los controladores en la (pág 17).

src/Infrastructure/FallbackController.php

src/Infrastructure/ThingController.php

#### Repositorios Iot\_emulator

src/Infrastructure/MySQLActionRepository.php

src/Infrastructure/MySQLPropertyRepository.php

src/Infrastructure/MySQLThingRepository.php

#### Comandos Symfony Iot\_emulator

A continuación compartimos a modo de ejemplo la ejecución del Comando Symfony src/Infrastructure/Thing/Command/SearchThingByIdThingIdCommand desde la terminal. Queremos hacer notar que estamos proporcionando tanto el usuario “user” como la contraseña “password” y que al final procesamos la salida con “jq”.

php bin/console app:Thing:SearchThingByIdThingIdCommand 1 user password | jq

Resultado

{

"id": 1,

"name": "thing\_name1",

"brand": "thing\_brand1",

"links": {

"actions": {

"link": "/actions",

"resources": {

"action\_name1": {

"values": "property\_value1"

}

}

}

}

}

#### Serializadores Iot\_emulator

Para serializar; Thing con Credenciales, sin credenciales y *Action*

src/Infrastructure/Thing/Command/Serializer/ThingActions

src/Infrastructure/Thing/Command/Serializer/ThingWithCredetials

src/Infrastructura/Thing/Command/Serializer/ThingWithoutCredentials

## Seguridad Iot\_emulator

Las peticiones con credenciales incorrectas

* No pueden acceder a la Zona Privada de un *WT*.
* No pueden dar de alta nuevos

Las peticiones sin credenciales

* No pueden acceder a la Zona Privada de un *WT*
* No pueden ejectuar *Actions*

En caso contrario y para no dar información sensible a un posible atacante devolvemos HTP 400 “Resource not found”

# Frontend SAC

URL: <https://socialaccesscontroller.tk>

Desde un punto de vista funcional y visual el SAC tiene tres varias partes diferenciadas. La primera es la “Raíz del proyecto” donde se desencadena el proceso de Login [37]. Las segunda es para usuario con rol *Owner* y la tercera para usuario con rol *Friend*.

En apartado “Raíz del proyecto” [32] mostramos el frontend de la entrada para cualquier rol. En apartado “Mapa web para *Owner*” [32] mostramos y explicamos las funcionalidades disponibles para *Owner.* En apartado “Mapa web parar *Friend* “[] mostramos y explicamos las funcionalidades disponibles para *Friend.*

Tal como explicamos en backend [37] solo el *Owner* y sus *Friends* podrán acceder más allá de “Raíz del proyecto”

Con la idea de hacer un proyecto escalable las vistas de SAC cargan muy poca información que van rellenando posteriormente con peticiones Ajax. Veáse por ejemplo los listado de *WTs* o listado de *Friends*.

## Raíz del proyecto

La ilustración [32] es una captura de pantalla donde se muestra en la raíz del proyecto, pide al usuario logarse vía Facebook.

El detalle de como gestiona SAC la Autenticación delegada se puede ver en la (pág 37) para el detalle sobre los datos guardados en base de datos de SAC en la (pág 44).



Ilustración 4. Captura de pantalla de la Raíz del proyecto.

## Mapa web para *Owner*

### Index del *Owner*

Endpoint: https://socialaccesscontroller.tk/owner

La (Ilustración 5) es una captura de pantalla cuando entra el *Owner*. Desde aquí puede ver información general del suya (33). Listado de todos los *WTs* dados de alta en SAC (33) con información sobre el *WT* y el estado de conexión. Un formulario para dar de alta nuevo *WTs* (34).



Ilustración 5. Visión general del Index del Owner.

#### Información general del Owner

La información general del *Owner*  se ve en ilustración[] es obtenida de Facebook durante la creación de nuevo *Owner* [39] y almacenada en SAC.



Ilustración 6. Información general del Owner vista en Index del Owner.

#### Listado de WTs dados de alta en SAC

La ilustración[] es una captura de pantalla donde se ve el listado de todos los *WT* dados de alta por *Owner* en SAC. En este ejemplo existen tres *WT* dados de alta cuyos nombres son *thing\_name1, thing\_name2 y thing\_name3*. Corresponden con los datos de prueba de Iot\_emulator usados durante el desarrollo y explicado en este apartado [23].

Cada *WT* tiene la información “nombre del *WT*” “*Brand* del *WT*”. El color de las letras determina si SAC ha conectado correctamente con el *WT.* Siendo la letra negra si SAC ha podido conectarse al *WT* (conexión exitosa). El texto en rojo muestra el error encontrado. Cada *WT* muestra un botón “Admin” para navegar a la página donde compartir ese *WT*[].



Ilustración 7. Listado de todos los WT que se muestran en Index del Owner.

#### Formulario para dar de alta nuevo WT

La (Ilustración 8) es una captura de pantalla donde se muestra el formulario para dar de alta nuevo WT. El *Owner* debe conocer el endpoint y credenciales de cada *WT* para introducirlas en el formulario mostrado. En caso de éxito SAC mostrará la página de “Éxito (pág 36)” en caso de error mostrará la página de “Error (pág 36)” informando del mismo.

Una de las mejoras propuestas [] sería la posibilidad de que SAC pregunte a la raíz de Iot\_emulator y construya una lista con los endpoints descubiertos. Haciendo más cómodo este proceso de dar de alta.



Ilustración 8. Formulario para dar de alta nuevo WT que se ve en Index del Owner.

## Admin de un *WT*

Endpoint: [https://socialaccesscontroller.tk/thing/{id}](https://socialaccesscontroller.tk/thing/%7bid%7dthing)

La ilustración () muestra la página a la que se accede a esta página dando al botón “Admin” para un *WT* concreto. Se puede ver el listado de *Actions* y sus *Properties* incluidos es *WT*.



Ilustración 9. Index de un Thing

## Ver listado de *Friends*

Al presionar botón de “Admin” en el “Index de un Thing” como muestra la (Ilustración-X) aparece un popup con el listado de *Friends*. Si esta *Action* puede ser compartida aparece un botón con “Share” en caso contrario aparece “(already shared)”.

Ilustración 10. Listado de amigos.

## Resultado de compartición

La (Ilustración 11) es una captura de pantalla que muestra el resultado de compartir un *Action* con un *Friend* tras pulsar el botón “Share” en el listado de *Friends*. Se puede ver el enlace generado que será usado por el *Friend* para ver el resutlado del *Action*.



Ilustración 11Resultado de la Ccompartición

## Mapa web para *Friend*

El punto de entrada del amigo hay:

- información general sobre el amigo

- listado de acciones compartidas por owner

VER UNA PROPIEDAD COMPARTIDA

Al dar en botón "Mostrar" se puede ver la propiedad de la acción compartida. Es un dato actualizado ya que en este momento preguntamos por dicha propiedad al WT.

## Página de Error y de Éxito

Existen eventos que acaban en éxito o en error. A continuación ilustraciones con capturas de pantallas de ambos.

### Página de Éxito

En la (ilustración TODO) se muestra una caputra de pantalla que aparece cuando un evento acaba satisfactoriamente, por ejemplo cuando un *Action* se comparte o un *Thing* se añade.

### Página de Error

En la (ilustración TODO) se muestra una captura de pantalla que aparece cuando ocurre un error. Por ejemplo cuando un *Friend* intenta ver un *Action* que no ha sido compartido.

# Backend SAC

Aparte del backend propiamente dicho que responde a peticiones existe otro componente que es el **Api SAC**: API usada por el Frontend SAC para obtener datos, se explica en este apartado [TODO lin api SAC].

Sac almacena el mínimo posible sobre el iot esto es:

Endpoint

Usuario

Contraseña

## Autenticación Delegada en Facebook

Cumpliendo con requistios de almacenar el mínimo posible de información [TODO link a Requistios] y de usar red de contactos de terceros [TODO link a requisito. Hemos implementado una autenticacón delegada en Facebook.

Cuando un usuario se loga a nuestra página lo hace a través de Facebook que nos devuelve un accessToken y un token único e invariable para cada usuario. SAC persiste ese token para recordar e identificar al usuario en futuras sesiones. El accessToken es válido para la sesión actual.

Es Facebook, no SAC, quien determina si un usuario es válido o no, quien sabe que *Friends* lo son del *Owner* y proporciona un token para identificar a los usuarios en distintas sesiones.

Lo que en Facebook se llama token en SAC es almacenado con el nombre de “fb\_delegated”

## Proceso de Login

En la siguiente ilustración [Ilustración 11. Caso de uso durante LoginIlustración 11] mostramos el árbol de decisiones que ocurre en SAC cuando un usuario intenta logarse en la raíz del proyecto.

Seguidamente explicamos las cuatro posibles finales para este árbol de decisiones: “Raíz del proyecto”,”Crear *Owner* “, “Index de *Owner*”. Index de *Friend*”o “Página de error”. A continuación explicamos los cuatro caminos:

A close up of a logo

Description automatically generated

Ilustración 12. Caso de uso durante Login.

### Camino “Raíz del proyecto”

SAC no dejará pasar ningún usuario de “Raíz de proyecto” si Facebook no devuelve un accessToken. Es este punto donde se pone de manifiesto el acceso delegado. Tal como comentábamos en [11] la autenticación se delega en Facebook. Es Facebook quien determina si un usuario es válido o no si Facebook.

En resto de caminos el usuario ya está logado correctamente en Facebook.

### Camino “Crear Owner”

Este camino acaba de la misma manera que “Index *Owner*“ pero con el paso extra de “Crear Owner”. La diferencia con el camino “Index *Owner*“ es que SAC no tiene *Owner* definido. SAC está preparado para que solo exista un único *Owner* de *WTs*. Es el primer usuario de Facebook logado quien asume el papel de *Owner*. El detalle de lo que ocurre se puede encontrar aquí []

### Camino “Index de *Owner*”

SAC reconoce al usuario logado como *Owner* y muestra “Index de *Owne*r” [LINK hacia vista de Owner]

### Camino “Index de *Friend*”

SAC reconoce al usuario logado como *Friend* de *Owner* y muestra “Index de *Friend*” [LINK hacia vista de Friend]

### Camino “Página de error”

SAC no reconoce al usuario logado ni como *Owner* ni como *Friend.* Entonces muestra página de error

## Crear Owner en SAC

Cuando SAC crea al *Owner* ocurren varias cosas. Primero se guarda en tabla owner el fb\_delegated (token de Facebook) y nombre del *Owner*. Segundo consultamos a Facebook el listado de *Friends* y lo guardamos en SAC guardando todos los fb\_delegated (tokens de Fadebook) de cada *Friend*.Una mejora propuesta es poder actualizar la lista de *Friends* [59]

## Crear Actions en SAC

Persistimos los datos de las Actions la primera vez que consultamos por ellas, esto es a la hora de mostrar [http://socialaccesscontroler/thing/{id}](http://socialaccesscontroler/thing/%7bid%7d). Link a donde explicamos relacióne entre Actions de los 2 proyectos.

Las *Actions* difieren en el Iot\_emulator. Al igual que pasa con

## Acciones sobre los WTs

### Dar de alta nuevo *WT*

Cuando SAC recibe una petición de alta de nuevo *WT*. Persiste llama a a Iot\_emulator y persiste en tabla thing. Root, user y password. Y en tabla action el nombre de las actions. Los detalles a más bajo nivel están en la ().

Las *Things* difieren en el Iot\_emulator. Al igual que pasa con

### Obtener información de un *WT*

Cuando SAC necesita complementar la información de un *WT* necesita consultarla al *WT* a través del Iot\_emulator. Existe información en la propia base de datos de SAC como son endpoint (root) su usuario y contraseña. Usando estas credenciales y enpoint consulta a Iot\_emulator () se obtiene de la tabla thing a través de la Entidad de Dominio Owner. Los detalles a más bajo nivel están en (46).

Cuando se necesita un listado de *WTs* por ejemplo en (Frontend listado WTs) SAC realiza esta lógica para cada *WT* del listado.

### Mostrar *Actions* de un *WT*

Para construir la información del listado de iots sac consulta a uno a uno a cada iot. Para hacerlo hemos creado en el Dominio la entidad ThingConnected. Y proporcionamos una api interna de sac para estas consultas.

En este momento sac consulta al iot por sus acciones y propiedades y los alamacena en base de datos.!!!!!

### Compartir un *Action* con un friend

Crear una relación entre friend y action

Al hacerlo se genera una relación entre una acción y un amigo. Se genera una URL vía API interna y se muestra al Owner que la compartirá con el friend.

### Crear URL de Compartición

Para crear la URL que el *Owner* compartirá con el *Friend* se usa SAC API tal como lo explicamos en la (40).

## API SAC

Hemos visto la necesidad de crear un API SAC por estas razones. Primero de conseguir una respuesta rápida del backend. Tener un frontend liviano que haga consultas Ajax por datos que necesite y no cargar el Response con datos que quizás no se usen. Segundo por seguridad, para evitar enviar información sensible al frontend como son el user y password de cada *WT.*

### DIAGRAMA de flujo entre Navegador SAC y Iot\_emulator

En la (Ilustración 12) vemos los pasos para btención de datos vía Ajax de Iot\_emulator desde un navegador. Como un *Friend* obtiene un dato de un *WT*.

1. HACER LEYENDA
2. Request inicial
3. Response incial.
4. Frontend llama a SAC API vía Ajax
5. Request al Iot\_emulator
6. Response del Iot\_emulator
7. Respuesta Ajax para rellenar el listado



Ilustración 13 Obtención de datos via Ajax del Iot\_emulator

### Usar el Sistema de enrutamiento de symfony desde frontend

Symfony ofrece una manera robusta de definir los paths de enrutamiento. La ruta concreta se define en ficheros yaml y se definen alias para referenciar las rutas en el código. De esta manera la ruta se puede modificar sencillamente en un único punto, el fichero.

Por ejemplo, si definimos una ruta con alias “nombre\_ruta” y con un parámetro tal que “/ruta/{key\_param}”. El ejemplo de uso de la anterior ruta tanto para backend como frontend podrían ser los siguientes:

Desde el backend en un AbstractController se usaría el método de Symfony *genrateUrl*:

// dentro de AbstractController

$url = $this->generateUrl(‘nombre\_alias’, [‘key\_param’ => ‘value\_param’)

print $url // /ruta/value\_param;

Desde el frontend con función de Twig *path*:

<!-- dentro de un template -->

<a href = “{{ path(‘nombre\_alias’,{‘key\_param’:value\_param}) }}”>Link</a>

<-- <a href=“/ruta/value\_param”>Link</a>

#### Problema en twig al usar Ajax

Nos hemos visto en la incapcidad de poder usar la función específica en Twig *path(*)*.* Ya que cuando Twig resuelve la función *path()* necesita el valor concreto del parámetro. Ocurre en varios puntos de nuestra arquitectura que el parámetro concreto para la ruta traemos a posteriori vía Ajax. Por eso no podemos usar *path* de Twig en frontend. LEER EL TEXTO, ENTENDIOD PERO HAY Q MEJORAR

Decidimos hacer que la Api funcionase como un generador de URLs. Desde el frontend con los valores de los parámetros ya disponible que se consultase a los enpoint de API SAC. Entonces desde el AbstractController sí se puede usar el método *generateURL*.

El esquema en los pasos sería muy parecido a lo explicado en la (Ilustración 12 Obtención de datos via Ajax del Iot\_emulator) pero sin los pasos 4 y 5 ya que la definición de la ruta está en las rutas explicdas en (pág 48) que forman parte de SAC por lo que no hace falta llamar al Iot\_emulator.

### Ejemplo de uso de API SAC

A modo de ejemplo del uso de API SAC vamos a explicar los pasos que realiza SAC para obtener información para construir el listado de *WT* (33) este listado se construye en “Index de *Owner*” (33). Es un buen ejemplo porque usamos los dos tipos de uso explicados anteriormente, estos son; la obtención de datos vía Iot\_emulator () como la generación de URLs ().La (Ilustración 13) es una captura de pantalla tras la carga completa de la página.



Ilustración 14. Ejemplo de uso de API SAC al generar el Listado de WT.

Explicamos cada paso siguiendo lo expuestos en “obtención de datos vía Ajax de Iot\_emulator” (X). Nos vamos a centrar en el *WT* con id = 1. Las llamadas para id 2 y 3 son muy similares.

#### El requets inicial

El resquest inicial es una petición GET a <https://socialaccesscontroller.tk/owner> hecha por el *Owner*.

#### Response inicial

En Symfony en App\Infrastructure\Controllers\OwnerController devuelve un esqueleto de la página pero sin apenas información. Inmediatamente empieza a pedir información.

#### Frontend llama a SAC API vía Ajax

Frontend lanza vía Ajax estas peticiones en busca de estos datos: información del *Owner*, la URL para administrar cada *WT*, la URL para preguntar por cada *WT*,request a la URL anterior y obtener los detalles del *WT*.

##### Request para obtener información del Owner - api/owner

La (lustracionX) es una captura de pantalla donde podemos ver la llamada a el endpoint api/owner de API SAC.



Ilustración 15XXX

La (ilustraciónX) es una captrua de pantalla donde podemos ver la response que devuelve como son el nombre del *Owner*, y un array llamado things con información de los *WTs*. Toda esta la información contenida en esta respuesta se encuentra almacenada en base de datos de SAC.



Ilustración 16xxx

##### Request para obterner URL para administrar cada WT -api/url/provider/thing

La (lustracionX) es una captura de pantalla donde podemos ver la request ‘api/url/provider/thing’ para obtener una URL.



Ilustración 17xx

La (ilustraciónX) es una captrua de pantalla donde podemos ver la response: “thing/1”.



Ilustración 18. Xxxx.

Esta información será usada como action del botón “Admin”. Como puede verse esta ruta devuelta en la response es la que está definida en fichero de enrutado src/Infrastructure/Resources/config/routes/thing.yaml. De esta manera hemos conseguido usar sistema de enrutado de Symfony desde el frontend que era la solución buscada en (Solucion generar un URL GENERADOR)

thing\_info:

path: /thing/{thingId}

##### Request para obtener URL del SAC API - api/url/provider/api/thing

El propósito de esta sección es muy similar al anterior(Request para (…) api/thing), esto es obterner una URL. Como funcionan igual y son muy parecidos no vamos a pararnos a explicar con mismo nivel de detalle.

La (lustracionX) es una captura de pantalla donde podemos ver la request “api/url/provider/api/thing” para obtener una URL.



Ilustración 19xxxx

La (ilustraciónX) es una captura de pantalla donde podemos ver la response “api/thing/1”



Ilustración 20xxxx

##### Request para obtener del SAC API información de un WT - api/thing/1

La (lustracionX) es una captura de pantalla donde podemos ver la request “api/thing/1”. Nótese que la URL de este request corresponde con la response de la request (TODO Request para obterner URL del SAC API). Este request es quien desencadena la llamada al Iot\_emulator.



Ilustración 21xxxx

#### Request al Iot\_emulator

En el siguiente código podemos ver estos request desde el access.log del nginx del servidor.

# cat /var/log/nginx/iot.socalaccesscontroller.access.log

35.180.227.177 - - [30/Aug/2019:09:47:47 +0000] “GET /1 HTTP/1.1” 2001 162 “-“ “-“

#### Response del Iot\_emulator

Lo que aquí ocurre recae totalmente en el Ioe\_emulator y enconcreto en endpoing GET /thing/{id} podemos ver la explicación en (TODO link a Aquitecturar REST iot\_emulator a este endpoint en concreto).

#### Respuesta Ajax para rellenar frontend

La respuesta del Iot\_emulator la serializamos en App\Infrastructura\Controllers\Api\ThingApiControler y devuelta con un Symfony\Component\HttpFoundation\JsonResponse. En la (ilustración XX) se ve la respuesta que llega al frontend. De aquí sacamos elNombre y *Brand*.



Ilustración 22XXXX

## Esquema Base de datos SAC

En la (Ilustración 23) mostramos el esquema usado en SAC. En ella se ven las tablas y campos usados. Pensamos que el esquema es auto explicativos. Vamos a hacer un repaso de los datos que almacenamos (). También expliaremos varios los campos fb\_delegated de tabla owner y tabla friend. El campo root de tabla thing. El campos name de la tabla Action.También la existencia de relación n:m entre owners y things. El usuario para acceder a esta base de datos debe tener esto permisos: DDL ALTER, DML SELECT, INSERT, UPDATE, DELETE.



Ilustración 23. Esquema de SAC.

### Datos almacenados en Base de Datos de SAC

Los campos almacenados en SAC obedecen al cumpliendo del requisito de almacenar la mínima cantidad de información por parte de SAC [5]. Como puede verse en la (Ilustración 23). Para *Owner* almacenamos el token de Facebook y su nombre. Para *Friend* lo mismo el token de Facebook y su nombre. Para *Thing* el root que lo explicamos en (pág 39) y sus credenciales; usuario y contraseña. Guardamos en *Action*el nombre ya que es nuestro nexo de unión con las *Actions* traidas del *WT*.

### Explicación Tabla Action: campo Name

Usamos el valor de la tabla “action” y campo “name” como nexo de union para *Action* entre la información del SAC e Iot\_emulator. Resolvemos esta union tal que los valores almacenados en table “action” campo “name” son matcheados con la respuesta del Iot\_emulator a nivel de “links->actions->link->resources”. A modo de aclaración mostramos una estructura básica de un *WT* vista en () con el path mencionado anteriormente resaltado.

{

"id": 1,

"name": "thing\_name1",

"brand": "thing\_brand1",

"**links**": {

"**actions**": {

"**link**": "\actions",

"**resources**": {

"action\_name1": {

"values": "property\_value1"

}

}

}

}

}

### Explicación fb\_delegated en tablas owner y friend

Estos campos tienen su utilidad durante el proceso de logeo del usuario y en concreto con la Autenticación Delegada [37].

El token proporcionado por Facebook lo persisitimos en base de datos y se usa para identificar al usuario logado en los campos explicado haciendo posible reconcocer al usuario en futuras sesiones. Tanto el token del *Owner* como los tokens de sus *Friends* los almacenamos en sus tablas; tabla owner para *Owner* y tabla friend para *Friends* durante proceso de creaciónde *Owner* [**¡Error! Marcador no definido.**].

### Explicación Tabla thing: campo root

Este campo es usado por API SAC para traer los datos actualizados de los *WTs*. El campo root es el endpoint al que dispara API SAC para obtener los datos actulizados. Es donde interviene la Entidad de Dominio ThingConnected (pág 40).

### EXplicación Relación n:m entre owners y things

Mientras que la base de datos está preparada para que SAC pueda soportar múltiples *Owners*, relación N:M entre *Owner* y *Thing* El caso de uso diseñado para la creación del *Owner* provoca hace inviable múltiples *Owners* por lo que una relación 1:N entre *Owner* y *Thing* hubiera sido suficiente. Hemos preferido estabilizar las diferentes funcionalidades ya implementadas y de cara a una facilitar una ampliación del proyecto dejar los esquemas con estas relaciones “mas grandes”. En el apartado de mejoras aparece esta (59).

## Arquitectura REST SAC

TODO explicar mejor cada endpoint

|  |  |  |
| --- | --- | --- |
| Verbo HTTP | Endpoint | Descripción |
|  | / |  |
|  | /loginOk |  |
|  | /api/owner |  |
|  | /api/thing/{thingId} |  |
|  | /owner/share/action/{actionId}/friend/{friendId} |  |
|  | /api/url/provider/thing |  |
|  | /api/url/provider/api/thing |  |
|  | /api/url/provider/api/share/action |  |
|  | /privacy | Requisito de Facebook |
|  | /conditions | Requisito de Facebook |
|  | /error |  |
|  | /friend/thing/{thingId}/action/{actionId} |  |
|  | /friend |  |
|  | /owner |  |
|  | /owner/Friends | SE USA? |
|  | /owner/things |  |
|  | /owner/create | SE USA? |
|  | /success |  |
|  | /thing/create |  |
|  | /thing/{thingd} |  |

## Arquitectura Hexagonal SAC

### Domnio SAC

Explicación de esta capa en (17).

// Entities

src/Domain/Entity/Action.php

src/Domain/Entity/Friend.php

src/Domain/Entity/Owner.php

src/Domain/Entity/Thing.php

// Repositories

src/Domain/Repository/OwnerRepository.php

src/Domain/Repository/ThingConnectedRepository.php

src/Domain/Repository/ThingRepository.php

src/Domain/Repository/Action:

src/Domain/Repository/Action/ActionRepository.php

src/Domain/Repository/Friend:

src/Domain/Repository/Friend/FriendRepository.php

#### ThingConnected

Es la entidad encargada de almacenar la infromación de un *WT* traída del Iot\_emulator. Tiene la particularidad que es un Respoitory que en su implementación src/Infrastructure/ThingConnected/CurlThingConnectedRepository.php usa Curl como fuente de sus datos y no usa Mysql como hacen el resto de Repositorios.

ThingConnected es una propiedad privada de Thing y tiene su getter y setter. Existe un Command y CommandHandlers encargados de traer información del ThingConnected.

### Aplicación SAC

Explicación de esta capa en (17).

#### Commands SAC

Explicar: GetFbSharingStatusByOwnerCommand.php, IsActualUserAnOwnerCommand.php, GetThingConnectedInfoCommand.php, MergeThingWithThingConnectedByIdCommand.php

// Commandos para Action

src/Application/Command/Action/CreateActionCommand.php

src/Application/Command/Action/SearchActionByIdCommand.php

// Comandos para Friend

src/Application/Command/Friend/CreateFriendCommand.php

src/Application/Command/Friend/SearchFriendByFbDelegatedCommand.php

src/Application/Command/Friend/SearchFriendByIdCommand.php

// Comandos para Owner

src/Application/Command/Owner/AddFriendToOwnerCommand.php

src/Application/Command/Owner/AddThingToOwnerCommand.php

src/Application/Command/Owner/CreateOwnerCommand.php

src/Application/Command/Owner/GetFbSharingStatusByOwnerCommand.php

src/Application/Command/Owner/GetListThingsByOwnerCommand.php

src/Application/Command/Owner/IsActualUserAnOwnerCommand.php

src/Application/Command/Owner/SearchOwnerByFbDelegatedCommand.php

src/Application/Command/Owner/ShareActionWithFriendCommand.php

// Comandos para Thing

src/Application/Command/Thing/CreateThingCommand.php

src/Application/Command/Thing/GetActionsByThingIdCommand.php

src/Application/Command/Thing/GetThingConnectedInfoCommand.php

src/Application/Command/Thing/MergeThingWithThingConnectedByIdCommand.php

src/Application/Command/Thing/SearchThingByIdCommand.php

src/Application/Command/Thing/ThingConnected

src/Application/Command/Thing/ThingConnected/GetThingConnectedCompleteByIdCommand.php

#### CommandHandlers SAC

// CommandHandler para Action

src/Application/CommandHandler/Action/CreateActionHandler.php

src/Application/CommandHandler/Action/SearchActionByIdHandler.php

// CommandHandler para Friend

src/Application/CommandHandler/Friend/CreateFriendHandler.php

src/Application/CommandHandler/Friend/SearchFriendByFbDelegatedHandler.php

src/Application/CommandHandler/Friend/SearchFriendByIdHandler.php

// CommandHandler para Owner

src/Application/CommandHandler/Owner/AddFriendToOwnerHandler.php

src/Application/CommandHandler/Owner/AddThingToOwnerHandler.php

src/Application/CommandHandler/Owner/CreateOwnerHandler.php

src/Application/CommandHandler/Owner/GetFbSharingStatusByOwnerHandler.php

src/Application/CommandHandler/Owner/GetListThingsByOwnerHandler.php

src/Application/CommandHandler/Owner/IsActualUserAnOwnerHandler.php

src/Application/CommandHandler/Owner/SearchOwnerByFbDelegatedHandler.php

src/Application/CommandHandler/Owner/ShareActionWithFriendHandler.php

// CommandHandler para Thing

src/Application/CommandHandler/Thing/CreateThingHandler.php

src/Application/CommandHandler/Thing/GetActionsByThingIdHandler.php

src/Application/CommandHandler/Thing/MergeThingWithThingConnectedByIdHandler.php

src/Application/CommandHandler/Thing/SearchThingByIdHandler.php

// CommandHandler para ThingConnected

src/Application/CommandHandler/Thing/ThingConnected/GetThingConnectedCompleteHandler.php

src/Application/CommandHandler/Thing/ThingConnected/SearchThingConnectedActionsHandler.php

src/Application/CommandHandler/Thing/ThingConnected/SearchThingConnectedBrandHandler.php

src/Application/CommandHandler/Thing/ThingConnected/SearchThingConnectedNameHandler.php

### Infrastructura SAC

Explicación de esta capa en (17).

#### Resources SAC

Ahondando en arquitectura desacoplada SAC posee la carpeta “Resources” donde hemos puesto las rutas usadas por backend o frontend. Y el mapeo de entidades del ORM. Dejando la entidades mas “puras”.

##### Archivos .yaml con definiciones de rutas

Explicar que esta dividdio en distinto archivos. La manera que frontend consulta al SAC API para generar URLS para backend esta explicada en (20).

// Configurciones de rutas

src/Infrastructure/Resources/config/routes/api.yaml

src/Infrastructure/Resources/config/routes/credentials.yaml

src/Infrastructure/Resources/config/routes/error.yaml

src/Infrastructure/Resources/config/routes/friend.yaml

src/Infrastructure/Resources/config/routes/owner.yaml

src/Infrastructure/Resources/config/routes/success.yaml

##### Mapeo de ORM

Tal como comentamos en (20) en SAC existen fichero .yaml donde cada entidad define tanto las relaciones entre ellas como sus propiedades. En es estos directorios donde quedan definidas

// Mapeo de ORM

src/Infrastructure/Resources/mappings/Action.orm.yml

src/Infrastructure/Resources/mappings/Friend.orm.yml

src/Infrastructure/Resources/mappings/Owner.orm.yml

src/Infrastructure/Resources/mappings/Thing.orm.yml

#### Controladores SAC

SAC posee varias zonas diferenciadas, zona *Owner,* zona *Friend,* y SAC API. Esta heterogenedidad se ve reflejada en los controladores que posee.

##### Controladores de API

ApiUrlGeneratorController es el generador de Urls explicado en (40). OwnerApiController es donde SAC consulta información del *Owner* se puede ver un request y una response a este endpoint en (42). Un ejemplo de unos de ThingApiController también se puede ver en (43) y está relacionado con la entidad ThingConnected.

// Controladores del API

src/Infrastructure/Controllers/Api/ApiUrlGeneratorController.php

src/Infrastructure/Controllers/Api/OwnerApiController.php

src/Infrastructure/Controllers/Api/ThingApiController.php

##### Controladores para resultados tanto de Éxito como de Error

Se puede ver ilustraciones de ambas en (36).

src/Infrastructure/Controllers/ErrorController.php

src/Infrastructure/Controllers/SuccessController.php

##### Controladores de Owner, Friend y Credentials

CredentialsController tiene función de ofrecer la página “Raíz de Proyecto” vista en (32) asi como endpoints pedidos por Facebook como son /privacy y /conditions. OwnerController se encarga de la parte vista en “Mapa web para *Owner*” (33) del enpoint /owner. FriendController la parte vista en “Mapa web para *Friend*” (36) del enpoint /friend. Por ultimo ThingController muestra “Admin de un Wt” (34) en endpoint thing/info también procesa thing/create.

src/Infrastructure/Controllers/CredentialsController.php

src/Infrastructure/Controllers/FriendController.php

src/Infrastructure/Controllers/HasFbSessionController.php <<<<< DUDA

src/Infrastructure/Controllers/OwnerController.php

src/Infrastructure/Controllers/ThingController.php

#### Respositorios SAC

src/Infrastructure/Action/MySQLActionRepository.php

src/Infrastructure/Friend/MySQLFriendRepository.php

src/Infrastructure/Owner/MySQLOwnerRepository.php

src/Infrastructure/Thing/MySQLThingRepository.php

src/Infrastructure/ThingConnected/CurlThingConnectedRepository.php

#### Serializadores SAC

// Serializador de Action

src/Infrastructure/Action/Serializer/ActionArraySerializer.php

// Serializador de Owner

src/Infrastructure/Owner/Serializer/OwnerArraySeralizer.php

// Serializador de Thing

src/Infrastructure/Thing/Serializer/ThingArraySerializer.php

src/Infrastructure/Thing/Serializer/ThingWithThingConnectedArraySerializer.php

// Serializador de ThingConnected

src/Infrastructure/ThingConnected/Serializer/ThingConnectedSerializer.php

#### Comandos Symfony SAC

##### Namespace

En siguiente código mostramos el namespace de los comandos Symfony para mostrar que está ordenado dentro de una carpeta con el nombre de la Entidad del Dominio seguida de la palabra “Command”.

src/Infrastructure/Action/Command

src/Infrastructure/Friend/Command

src/Infrastructure/Owner/Command

src/Infrastructure/Thing/Command

src/Infrastructure/ThingConnected/Command

##### Listado completo

Para mostrar el listado completo compartimos del listado generado por bin/console la zona perteneciente a “app”.

$ php bin/console

app

app:Action:Create Creates an Action

app:Action:SearchActionById Searches an action By id

app:Friend:Create Test

app:Friend:SearchFriendByFbDelegated given fbDelegated returns friend

app:Friend:SearchFriendById given id returns friend

app:Owner:AddFriendToOwner Given fbDelegated of Owner and fbDelegated of Friend will create a relationshing in owner\_friend table

app:Owner:AddThingToOwner Given fbDelegated of Owner and Thing id will create a relationship in owner\_thing table

app:Owner:Create Add a short description for your command

app:Owner:GetFbSharingStatus gets All relationships between Owner-Friends-Actions

app:Owner:GetListThingsByOwner Given an fb\_delegated returns list of things

app:Owner:SearchByfbDelegated Given a fbDelegated gets Owner

app:Owner:ShareActionToFriend Given an owner fbDelegated, a Friend ID and ActionId. Shares Given Action to Friend

app:Thing:Create Add a short description for your command

app:Thing:GetActionsByThingId Given an thing.id returns Actions

app:Thing:MergeThingWithThingConnectedById Given and (int) id merges sac Thing with ThingConnected

app:Thing:SearchByThingId Given a (int) id searches Thing

app:ThingConnected:GetThingActionsByThingId Connects to thing and retrieves name

app:ThingConnected:GetThingBrandyThingId Connects to thing and retrieves name

app:ThingConnected:GetThingConnectedCompleteById Connects to thing and retrieves All ThingConnected Info

app:ThingConnected:GetThingNameByThingId Connects to thing and retrieves name

#### Event Susbscribers SAC

Para poder proteger zonas que necesiten de autenticación Facebook hemos usado los Eventos de Kernel de Syfmony. Esta sección es única para SAC.

Explicar MAS el controlador extiende de aquí.

// EventSubscriber

src/Infrastructure/EventSubscriber/HasFbSessionSubscriber.php

## SEGURIDAD

### HTTPs

Certificado creado con XXXX (TODO buscar)

No podrá acceder a zona Friend:

- Ningúna persona que no sea amigo

- Friends sin acciones compartidas

No podrá ver Actions aquellos Friends que no tengan esa Action compartida por Owner

# Tecnologías usadas

## PHPStorm:

IDE Comercial multiplataforma. Hemos elegido este IDE frente a otros por su manera amigable de funcionar con muchas tecnologías del trabajo fin de máster como son:

* PHP
* HTML
* MySQL
* Doctrine
* Javasript
* Twig
* Symfony
* Cliente HTTP

Asi como ayudas que ofrece mojerar el código:

- PSR

- Creación de servicios symfony

- Búsqueda inteligente de

- definiciones de métodos

- implementaciones de interfaces

## Facebook

### Facebook API

Autenticación delegada

Consulta de lista de amigos

### Facebook.developers

Creación de usuarios de prueba

Login

Creación de aplicación web

## Sistema operativo

Hemos usado Ubuntu tanto en el desarrollo como la puesta en producción. Es un sistema operativo open source basado en Debian con mucho TODO "bagaje" y centrado en robustez. Se han usado las siguientes capacidades de Ubuntu

### variable de entorno

${USER} para poder desarrollar en distintas máquinas y poder compartir comandos

### sshfs:

Permite para montar en local vía ssh sistema de ficheros de AWS, asi poder trabajar con phpstorm:

sudo sshfs ubuntu@35.180.227.177:/var/www/iot\_emulator /mnt/iot\_emulator -o IdentityFile=/home/${USER}/dev/sac\_sandbox/docs/socialaccesscontroller-paris.pem -o allow\_other

sudo sshfs ubuntu@35.180.227.177:/var/www/sac /mnt/sac -o IdentityFile=/home/${USER}/dev/sac\_sandbox/docs/socialaccesscontroller-paris.pem -o allow\_other

### alias

Alias definidos durante el desarrollo. De esta manera se agiliza el reuso de conjuntos de comandos usados reitiradamente

alias iot\_emulator='cd ~/dev/iot\_emulator'

alias iot\_emulator\_clean\_http\_requests='rm /home/${USER}/dev/iot\_emulator/.idea/httpRequests/\*'

alias iot\_emulator\_php\_server\_run='iot\_emulator && php bin/console server:run'

alias iot\_emulator\_shcema\_drop\_and\_create\_fixtures\_load\_NOT\_symfonys='iot\_emulator && php bin/console doctrine:schema:drop --force && php bin/console doctrine:schema:update --force && php fixture/create\_things.php && cd -'

alias sac\_clean\_http\_requests='rm /home/${USER}/dev/sac/.idea/httpRequests/\*'

alias sac\_fixtures\_load='sac && php bin/console doctrine:fixture:load -n && cd -'

alias sac\_fixtures\_load\_append='sac && php bin/console doctrine:fixture:load -n --append && cd -'

alias sac\_php\_server\_run='sac && php bin/console server:run'

alias sac\_sandbox='cd /home/${USER}/dev/sac\_sandbox/sac\_sandbox'

alias sac\_sandbox\_fixtures\_load='sac\_sandbox && php bin/console doctrine:fixture:load -n && cd -'

alias sac\_sandbox\_fixtures\_load\_append='sac\_sandbox && php bin/console doctrine:fixture:load -n --append && cd -'

alias sac\_schema\_drop\_and\_create='sac && php bin/console doctrine:schema:drop --force && php bin/console doctrine:schema:update --force && cd -'

alias sac\_schema\_drop\_and\_create\_and\_fixtures\_load='sac\_schema\_drop\_and\_create && sac\_fixtures\_load && cd -'

### shell-script para provisionamiento

TOD copiar Shell script

### httpie

[LINK-INTERNO] DISeNYO-FIXTURES

Cliente http de terminal usado junto con cliente de phpstorm a la hora de probar y desarrollar las distintas apis de sac e iot\_emulator. En ambos proyectos se encuentra en docs/requests

### jq

Procesador json por terminal, usado para mostrar respuestas curl o buscar ciertos claves o valores en respuestas.

### git

Sistema de control de versiones que nos ha permitido trabajar en distintas necesidades de los proyectos, pudiendo dividir el trabajo en ramas.

## github

Lugar donde almacenar los proyectos de manera privada y poder acceder a ellos en etapa de provisionamiento. Estos son los repositorios creados:

* https://github.com/danielsalgadop/sac
* https://github.com/danielsalgadop/iot\_emulator

## aws ec1

Hemos elegido este servicio de computación por su buena relación precio/calidad, por su fácil configuración y alta disponibilidad. Aqui hemos configurado una máquina ubuntu con ambos proyectos desplegados

### Características de máquina desplegada en producción

TODO

## nginx

Hemos usado nginx por su facilidad a la hora configurar subdominios y https

## html5

## javascript

## jquery

## Jquery-UI

## moustache

## symfony 4

## npm

## mysql

## Doctrine

# Análisis resultados

Hemos creado un entorno seguro donde las relaciones en facebook se mantienen dentro de SAC. Un dueño puede elegir qué acciones compartir y a quien compartirlas.

Es posible almacenar las credenciales y los permisos de los *WTs* de manera segura en SAC y dejar la red de contactos a a un tercero, en este caso a Facebook.

Se consigue simplificar la manera de compartir *WTs*

Esta aplicación es un intermediario que unirá y dará seguridad a la información que existe en Facebook (red de amigos) con la información que el *Owner* quiere compartir con *Friend* y que existe en los *WTs* (*Actions*)*.*

# Relación con Asignaturas del máster

--------- ENTORNO WEB

uso alias para llamar comandos symfony (doctrine)

git

AWS

variable entorno

httpie request

nginx

--------- FRONT

html5

javascript

jquery

moustache

--------- FRAMEWORKS

symfony 4

--------- DESARROLLO EFICIENTE

arquitectura hexagonal

PATRONES: DTO, Inyección de dependencias (esto se ve en los constructs(cuando metemos un Handlers aquí)

Nombres semánticos

--------- SQL

mysql

Doctrine

--------- SEO

routas SEO

--------- RENDIMIENTO

--------- EMPRENDEDURIA

## Frameworks

## Emprendiduría

## Entorno Web

## Seguridad

## Frontend

## PHP

## Bases de Datos

# Conclusiones

"Lo que acabas de explicar"

El proyecto esta hecho en el marco de IOT.

Se ha hecho iot\_emulator en capa de accesibilidad y sac en capa de compartición

Hemos podido usar facebook para manejar el acceso a iot basado en sus propiar redes sociales. Dando links personalizados

Hemos emulado iot según las reglas adapatadas por nosotros establecidas por w3 web thing model

# Instalación

## ---- INSTALATION SYSTEM REQUIREMENTS

Hemos usado EC2 con conexiones ssh (amdinistracion del sistema), http (iot\_emulator) https (sac).

como instalar phpunit, hasta que no lanzo el primer phpunit no lo instala

System requiremetns, PHP (y extensiones), nginx, mysql and npm:

- sudo apt-get install mysql-server nginx php php-zip php-mysql php7.2-xml npm php-curl php-fpm composer php-fpm -y

- sudo apt-get install php-curl (actualy in iot\_emulator, fixtures are done in a php script via curl)

- sudo a2enmod rewrite

Development requirements

- sudo apt-get install httpie jq

## ------- INSTALACION SAC

(sac) App scaffold:

git clone https://github.com/danielsalgadop/sac

composer install

Facebook, create proyect and get FACEBOOK\_APP\_ID and FACEBOOK\_SECRET

(sac) Create .env.local

- fill FACEBOOK\_APP\_ID and FACEBOOK\_SECRET

- fill DATABASE\_URL

Create .env.local with FB and Mysql credentials

php bin/console doctrine:database:create

php bin/console doctrine:schema:create

## ------------ INSTALACION IOT\_EMULATOR

(iot\_emulator) App scaffold:

git clone https://github.com/danielsalgadop/iot\_emulator

composer install

npm install

(iot\_emulator)

- For simplicity database user, database name and table all are 'iot'

php bin/console doctrine:database:create

php bin/console doctrine:schema:create

# Mejoras

### Multi-owner

XXX

### Cacheado inteligente de datos de cada *WT*

Existen datos más estables en el tiempo, como puede ser el *Brand* o el nombre de un *WT.* Frente a otros como el dato de la temperatura registrada por un termómetro que tienen utilidad por la actualización constante que el *WT* ofrece.

Proponemos que aquellos que nonecesitan ser actualizados puedan ser almacenados en sistema de chacheo estilo Redis. Mientras que los otros sí deban ser consultados en cada momento.

### Actualización de Friends Facebook

Como los *Friends* pueden cambiar debería existir una manera o repetida automáticamente en el tiempo o lanzada por el usuario para poder actualizar la lista de amigos.

### Descubrimiento de *WTs*

En proceso de alta de un *WT* incluir un botón “Descubrimiento” que muestre un popup con los endpoint descubiertos, que permita al usuario de manera cómoda introducir “usuario” y “contraseña” para dar de alta masivamente *WTs* en SAC.

Cabe recordar que Iot\_emulator ofrece los endpoints públicos de todos los *WTs* emulados en un JSON haciendo una petición GET a su raíz.

# Glosario

* **Web Thing** (*WT)*. Objeto con conexión a internet que ofrece información interna vía http(s) y arquitectura REST. En el presente proyecto *WT* devuelven formato de datos JSON. Posee dos zonas:
  1. Zona Pública
     + Nombre
     + Brand
  2. Zona Privada
     + Action
     + Property
* **Property**: Es una propiedad de un *WT* representa un estado interno.
* **Action**: Es una iteracción con un *WT* que permite invocar una función en un *WT*. Una acción permite ver el estado de una *Property*.
* **Iot\_emulator**: recurso web donde configurar emulaciones de conjuntos de *Iots*.
* **Owner**: persona que posee *Iots*, conoce las credenciales para acceder a zona privada de *Iot*. Ademas posee cuenta de facebook y red de amigos dentro de esta red social
* **Friend**: persona conectada como amigo en facebook del owner.
* **Social Access Controller** (*SAC)*. Aplicación acoplada a facebook donde un *Owner* puede compartir cierta *Action* con un *Friend*.

# Bibliografía

1-"Sharing Using Social Networks in a Composable Web of Things"

2- <https://www.w3.org/Submission/2015/SUBM-wot-model-20150824/> El de w3Consorcium

3- “Tokens de Acceso” https://developers.facebook.com/docs/facebook-login/access-tokens/?locale=es\_LA