

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 y disponibles vía web 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

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 WTs. El modelo emulado es 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.

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 Iot\_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. A continuación vermos qué partes sí hemos construido y explicaremos cuáles no. n ambos proyectos.hemos usado diseño de software siguiendo 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 hemos creado un Iot\_emulator. A continuación, explicamos en qué medida nos hemos acoplado.

### Patrón de Integración de los WT en W3Consortium

W3Consortium reflejade 3 maneras de conectividad; Directa, Gateway, o Cloud. Nosotros no podemos acoplarnos estrictamente a ninguna de ellas ya que emulamos la capa de accesibilidad con el Iot\_emulator. Como veremos en la pág 29 esta falta de patrón de integración tiene impacto a la hora de definir las rutas REST.

### Requerimientos para un WT en w3Consortium

A continuación, explicamos de qué manera nos hemos adaptado los requistios del W3Consortium. Para ello tomamos como referencia las tablas del apartado 5. Web Thing requirements. Estos requisitos determinan que debe cumplir un objeto conectado a una red para poder ser considerado con un WT.

En las siguientes tablas la columna “Nivel de cumplimiento” describe en el grado en que hemos cumplido los requitistos en el Iot\_emulator y en caso de no cumplirse explicamos brevemente la razón de dicha carencia. La columna “Definición de requisito” está sacada del W3Consortium.

#### 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 | Sí |
| 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 |

Tenemos un alto grado de adaptación con estos requisitos. Cada WT no es un único servidor porque todos los WT emulados se encuentras detrás del Iot\_emulator que funciona de hub,

#### 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 |

Seguimos teniendo un alto grado de adaptación. No hemos implementado WebSocket ya que el mayor esfuerzo lo hemos hecho en SAC y no en Iot\_emultor asi que los WebSockets quedan fuera del alcance pretendido en el Trabajo Fin de Máster. La documentación legible para humanos tampoco hemos considerado que fuera necesaria para los objetivos buscados.

#### Nivel 2 – WT PODRÍA

|  |  |
| --- | --- |
| Definción de requisito | Nivel de cumplimiento |
| A Web Thing MAY support the HTTP OPTIONS verb for each of its resources | No |
| 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 |

Este nivel al ser opcional nos ha parecido poco importante y no lo hemos acometido

Como vemos el Iot\_emulator y los WT emulados han sido diseñados para cumplir con los requisitos del SAC y no tanto para ser emulaciones estrictas tal como definie el W3Consortium.

## Acople al SAC de Dominique Guinard

Realmente el SAC creado en este Trabajo Fin de Máster se basa al propuesto por Dominique Guinard. Por aún quedan partes no acopladas. La primera diferencia es el nivel que dependencia total de Facebook mientras que Guinard habla de administración basada en estructura social basada en varias redes sociales Otra mejora es la hora de descubrir WTs Guinard propone que el SAC lo haga de manera automática rastreando la red. Tampoco ofrecemos la posibilidad de discernir qué verbo HTTP quiere el Owner ofrecer al Friend.

## 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 internas no conocen las capas externas.

Las capas se comunican entre ellas de fuera hacia adentro con adaptadores. Siguiendo un patrón de diseño llamado “Port and Adapters”. La capa interna ofrece “Ports” a modo de contrato hacia la capa externa. La capa externa usa “Adapters” que son implementaciones de los “Ports”.

### Dominio

Es la capa más interna. Contiente las entidades que definen a nuestra aplicación. Las Entidades son objetos PHP que están libres de lógicas como por ejemplo saber cómo deben representase o cómo guardarse. Para interactuar con las Entidades tenemos los “Ports” en los Repositorios estos definen los contratos con Interfaces PHP que sólo incluyen lógica semántica.

Para ello, hemos definido dos namespaces; Entity y Repository. En Entity están los objetos puros y en Repository las interfaces.

### Aplicación

Es la capa por encima del Dominio. Hacia dentro implementa los adaptadores para el Dominio de los Repositorios de este y se comunica hacia la arriba (capa de Infraestructura) definiendo cómo debe ser esa comunicación. Adapta las peticiones que vienen de Infraestructura para usar las entidades de Dominio.

Hemos definido dos namespaces Command y CommandHandlers. Los Commands son DTOs que transportan los datos a los CommandHandlers. Los CommandHandlers son orquestadores que por un lado manejan las entidades Dominio mediante sus Repositorios recibidos en el constructor y por el otro poseen un método (handle) que recibo el DTO (el Commando) con los datos necesarios. Si hay algún problema suelen lanzar Excepción.

Cualquier tipo de input que reciba la aplicación va a acabar en los CommandHandlers. En nuestro caso el caminio a los CommandHandlers puede llegar de dos sitios o inputs vía HTTP gestionados por Controlers o son invocados por Comandos de Symfony desde la terminal.

### Infrastructura

La relación con la capa de Aplicación es clara en cuanto que reciben por constructor los CommandHandlers e instanciand los Commandos (DTO) con los datos recibidos del exterior. Con estas dos piezas ejecutan el método handle de los CommandHandlers.

Es la capa más externa, es donde vive el framework usado (Symfony) y el sistema de base de datos (MySQL). Por lo que suele tener incluir muchas librerías de terceros. No existe lógica propia de nuestra 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.

#### Implementación de Respositorios

Las implementaciones de los repositorios responden a sus correspondientes contratos de la capa de Dominio. Esto permite que sea fácil cambiar implementaciones de repositorios usando diferentes tecnologías, por ejemplo en cuanto al framework PHP cambiarlo por Laravel o ZendFramework o cambiar la base de datos a Postgress o mongo. Estos cambios se harían 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 lo son los Controladores, hemos 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 ha sido usando estos Comandos Symfony.

Se ejecutan desde la raíz del proyecto desde la terminal. Todos tiene el prefijo **app** (dos puntos) seguidos del nombre del **entidad de Dominio que usen** (dos puntos) seguidos del **Commando de la aplicación** que ejecutan.

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.

tests/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. Al usar Doctrine la capa de Infrastructura tiene acceso al entityManager de Doctrine que posee algunos métodos muy útiles como find o findOneBy.

### 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** WTs 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 create\_things.php. Este script lanza petidiones HTTP para crear WT de prueba. Estos datos quedan persisitidos en la base de datos.

Se lanza desde la raíz del proyecto

php fixture/create\_things.php

Los datos de prueba tienen esta estructura de naturaleza 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 los Fixtures. 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 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 PHP puro y los hechos con PHPUnit-

### No usan PHPUnit

Los dos primeros esperan la escturctura de datos vistos den la pág 13 de los datos de puebras. El último testea el método isIntegrityValidOnCreate que comprueba si los datos recibidos en la creación de un WT son correctos.

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

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

php tests/notPHPUnit/isIntegrityValidOnCreate.php

### Usan PHPUnit

El primero comprueba que el nombre de una una Property coincidad con el nombre de un Actions. El último es redundante con el test del mismo nombre que no usa PHPUnit.

tests/Domain/Entity/Thing/hasActionsAndPropertiesConcordanceTest.php:16

tests/Domain/Entity/Thing/isIntegrityValidOnCreate.php

### Peticiones HTTP

Hemos creado estas peticiones para testear que la API funcionaba como esperábamos. Son tests que actuan sobre WT están divididas en tests que deben fallar (must\_work) y tests que deben funcionar (must\_fail).

tests/requests/get\_value\_of\_property/must\_fail/get\_value\_of\_inexistent\_property.http

tests/requests/get\_value\_of\_property/must\_fail/get\_value\_of\_inexistent\_thing.http

tests/requests/get\_value\_of\_property/get\_value\_of\_property.http

tests/requests/execute\_action/must\_work/exectue\_action.http

tests/requests/execute\_action/must\_fail/incorrect\_action\_name\_in\_payload.http

tests/requests/execute\_action/must\_fail/incorrect\_action\_name\_in\_url.http

tests/requests/get\_actions/must\_work/get\_actions.http

tests/requests/get\_actions/must\_fail/get\_actions\_empty\_credentials.http

tests/requests/get\_actions/must\_fail/get\_actions\_incorrect\_password.http

tests/requests/get\_thing/must\_work/get\_thing.http

tests/requests/get\_thing/must\_fail/invalid\_credentials.http

tests/requests/get\_thing/must\_fail/no\_credentials.http

tests/requests/get\_thing/must\_fail/search\_non-existent.http

tests/requests/list\_of\_things/must\_work/list\_of\_things.http

tests/requests/delete/delete\_thing\_by\_id.http

tests/requests/update/must\_work/update\_id\_1\_action\_jsonAction1.http

tests/requests/update/must\_fail/non-existing\_property.http

tests/requests/update/must\_fail/non-existing\_action\_nor\_property.http

tests/requests/update/must\_fail/non-existing\_action.http

tests/requests/create/must\_work/create\_thing.http

tests/requests/create/must\_fail/noActionPropertyConcordance/wrong\_property\_key.http

tests/requests/create/must\_fail/noActionPropertyConcordance/missing\_property\_key.http

tests/requests/create/must\_fail/noActionPropertyConcordance/wrong\_action\_key.http

tests/requests/create/must\_fail/noActionPropertyConcordance/missing\_action\_key.http

tests/requests/create/must\_fail/wrongCredentials/no\_user\_nor\_password.http

tests/requests/create/must\_fail/wrongCredentials/no\_user.http

tests/requests/create/must\_fail/wrongCredentials/no\_password.http

tests/requests/create/must\_fail/insufficient\_data/no\_payload.http

tests/requests/create/must\_fail/insufficient\_data/no\_name.http

tests/requests/create/must\_fail/insufficient\_data/empty\_json.http

tests/requests/create/must\_fail/insufficient\_data/no\_brand.http

## SAC

### Peticiones HTTP

Estas peticiones están diseñadas para probar los endpoints del Owner del Thing y de la API SAC.

./tests/requests/thing/create/create\_thing.http

./tests/requests/owner/create/must\_work

./tests/requests/owner/create/must\_work/create.http

./tests/requests/api/thing/must\_work/GetThingById.http

./tests/requests/api/owner/must\_work/GetBasicInfo.http

# 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 definidas por nosotros que no son del W3Consortium. Aquí explicamos la diferencia.

### Zona pública y Zona privada

Hemos decidido crear dos zonas diferentes en los WT Asi somos capaces de diferenciar peticiones hechas sin credenciales y con ellas. Esta diferente respuesta queda clara al mostrar el listado de todos los WTs emulados en iot\_emulator.

A la Zona pública se llega con una requests GET sin credenciales y muestra WT-brand y WT-name. En cambio, la Zona privada se accede enviando credenciales “user” y “password” en los headers de la requesst y muestra los mismo que la Zona Pública y además losActions del WT.

#### Respuesta JSON zona privada

Este ejemplo vemos una petición GET hecha con httpie con las credenciales correctas. Y más abajo la respuesta

http <http://iot.socialaccesscontrller.tk/1> user:user password:password

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"

}

}

}

}

}

#### Respuesta JSON zona pública

Ahora lanzamos la petición sin las credenciales

http <http://iot.socialaccesscontrller.tk/1>

Vemos que la respuesta no contiene la sección de los Actions.

{

"id": 1,

"name": "thing\_name1",

"brand": "thing\_brand1"

}

### 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

url/actions/action\_name1

Devolvería

property\_value1

Es decir, nuestros Actions sirven para modificar el valor de la propety. 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 de la base de datos del Iot\_emulator.

## Arquitectura REST Iot\_emulator

Vamos a explicar cada uno de los endpoints explicando su funcionalidad.

|  |  |  |
| --- | --- | --- |
| 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. Muesta un listado con la todos los WT mostrando su parte pública.

#### GET /{id} Sin credenciales

Accedes a la zona pública del WT con id igual a {id}. Sólo se muestra el WT-brand y WT-name.

#### GET /{id} Con credenciales

Accedes a la zona privada del WT con id igual a {id}. Además de la zona pública se ve la parte de Actions del WT.

#### POST /create

Crear un nuevo WT siempre que reciba un json con datos bien necesarios, estos datos son, un json con el nombre de WT “name”, su brand “brand” y las Actions y Properties que cumplan la concordancia vista en pág 18. También espera que en las cabeceras HTTP venga el “user” y “password” que serán usados para autenticar al Owner en el WT. El siguiente código muestra la request forjada en test:

tests/requests/create/must\_work/create\_thing.http

En esta petición podemos ver varias cosas. Esta escrita para funcionar con cliente HTTP de phpstorm y además espera que en el sistema operativo exista una variable global llamada $IOT\_EMULATOR que contiene la dirección al Iot\_emulator. En producción $IOT\_EMULATOR equivaldría a http://ioit.socialaccesscontroller.tk.

POST {$IOT\_EMULATOR}/create  
*Content-Type*: application/json  
*user*: user  
*password*: password  
  
{  
 "name": "jsonName",  
 "brand":"jsonBrand",  
 "links":  
 {  
 "properties":[  
 {"property\_name1":"property\_value1"},  
 {"property\_name2":"property\_value2"}  
 ],  
 "actions":["property\_name1","property\_name2"]  
 }  
}

#### POST /{id}/actions/{action\_name}

Ejecuta la Action con nombre {action\_name} del WT con id igual a {id}. Como las Actions son punteros a propiedades con el mismo nombre, en la práctica cambia el valor de la Property. Espera que la request contenga un json con el nombre de la Action y el valor que se le añadirá a la Property.

#### GET /{id}/properties/{property\_name}

Muestra el valor de la Property con nombre {property\_name} del WT con id igual a {id}

#### GET /{id}/actions

Muestra un listado de los Actions del WT con id igual a {id}.

## Arquitectura hexagonal de Iot\_emulator

### Dominio Iot\_emulator

Explicación de capa de Dominio en pág 17. Hemos creado Entidades y Repositorios.

// Entidades

src/Domain/Entity/Action.php

src/Domain/Entity/Property.php

src/Domain/Entity/Thing.php

src/Domain/Entity/User.php

// Repositorios

src/Domain/Repository/ActionRepository.php

src/Domain/Repository/PropertyRepository.php

src/Domain/Repository/ThingRepository.php

src/Domain/Repository/UserRepository.php

Teniendo en cuenta que la funcionalidad del Iot\_emulator es la de emular WT es lógico que la entidad más importante sea la de Thing. Como se en la Ilustración 3 el resto de entidades depende de ella de alguna u otra manera.La relación entre entidades la hemos creado usando el ORM Doctine y para definir las relaciones las @annotations esscritas en las propias Entidades. Es una manera de hacerlo que no sigue la teoría hexagonal ya que está muy acoplada llevando lógica de Infrastructura al Dominio pero se usa en muchos proyectos.

Este código está sacado de src/Domain/Entity/Thing.php. Mostramos las porciones de las @annotatios donde se puede ver las relaciones que hay entre Thing y resto de entidades.

*/\*\*  
 \** ***@ORM\OneToMany****(targetEntity="App\Domain\Entity\Action", mappedBy="IdThing", orphanRemoval=true, cascade={"persist"})  
 \*/*private $actions;  
  
*/\*\*  
 \** ***@ORM\OneToOne****(targetEntity="App\Domain\Entity\User", mappedBy="thing", cascade={"persist", "remove"})  
 \*/*private $user;

### Aplicación Iot\_emulator

Explicación de capa de Aplicación en pág 17. Hemos creado Commands como CommandHandlers y Dto

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

Existe una relación 1 a 1 entre todos los Commands y sus CommandHandlers. El siguiente código 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

#### DTO específico de Iot\_emulator

Además de los DTOs creados en Application/Command hemos usado uno para transmitir 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

### 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

##### Namespace

En siguiente código mostramos el namespace del comando Symfony vemos como el Comando está en carpeta con el nombre de la Entidad Thing del Dominio seguida de la palabra “Command”.

src/Infrastructure/Thing/Command/SearchThingByThingIdCommand.php

##### Listado completo

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

$ php bin/console

app:Thing:SearchThingByThingId Given a (int) id, (int) user and (int) password searches Thing

A continuación, compartimos 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 y sin Credenciales serializar y Action.

src/Infrastructure/Thing/Command/Serializer/ThingActions

src/Infrastructure/Thing/Command/Serializer/ThingWithCredetials

src/Infrastructura/Thing/Command/Serializer/ThingWithoutCredentials

## Seguridad Iot\_emulator

A la hora de tener seguridad, desarrollar y hacer las pruebas hemos seguido las siguientes premisas.

Las peticiones con credenciales incorrectas o sin credenciales.

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

# 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]. La 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. Véase por ejemplo el listado de WTs o el 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

La ilustración 5 es una captura de pantalla al index del Owner encontrado en https://socialaccesscontroller.tk/owner. Owner accede a él después de logarse. Desde aquí puede ver su propia información general pág 33, el listado de todos los WTs dados de alta en SAC pág 33 con información sobre cada WT y el estado de conexión y finalmente un formulario para dar de alta nuevo WTs pág 34.



Ilustración 5. Visión general del Index del Owner.

#### Información general del Owner

La información general del Owner es la parte del index que mostramos en la Ilustración 6. Estos datos son obtenidos de Facebook durante la creación de nuevo Owner pág 39.



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

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

La Ilustración 7 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 vistos en pág 23.

Cada WT tiene la información nombre del WT y 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” en caso de error mostrará la página de “Error” informando del mismo.

Una de las mejoras explicada y propuesta en la pág 54 es la posibilidad de que SAC descubra WTs para hacer más cómodo el proceso de alta.



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

## Admin de un WT

La Ilustración 9 es una captura de pantalla de aministracion de un WT en [https://socialaccesscontroller.tk/thing/{id}](https://socialaccesscontroller.tk/thing/%7bid%7dthing). Se accede tras dar al clicar “Admin” para un WT concreto desde el “Index del Owner”. Mostramos el nombre del WT el Brand y un listado de Actions y sus Properties propios del 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 resultado del Action.



Ilustración 11. Resultado de la Compartición

## 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 cuando se añade un WT.

### 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

El bakend del SAC está compuesto por dos partes bien diferenciadas. El backend propiamente dicho que responde a peticiones externas y otro componente que es el Api SAC: API usada por el Frontend SAC para obtener datos.

Para explicar toda la complejidad vamos a ver en empezar viendo como logear usando la autenticación delegada en Facebook que desencadena lo que vemos después que es el proceso de login con todos los distintos caminos posibles en un árbol de decisión. Para exponer algo tan central en la lógica de negocio listamos y comentamos todos las posibles acciones que se acometen en SAC con los WT. Tras un razonamiento de porque hemos creado un API SAC y explicar el diagrama general de su comportamiento miraremos en detalle y a modo de resumen como construimos el listado de WTs. Igual que en Iot\_emulator exponemos la arquitectura de software seguida, diseño de base de datos. Por último las directrices de seguridad aplicadas al SAC.

## Autenticación Delegada en Facebook

Cumpliendo con requisitos de la pág 2 de almacenar el mínimo posible de información y de usar red de contactos de terceros. Hemos implementado una autenticación delegada en Facebook. Esta autenticación nos permite identificar inequívoca y persistentemente a cada usuario. Es Facebook, no SAC, quien determina si un usuario está autenticado también es Facebook quien nos dice que Friends tiene el Owner.

Cuando un usuario se loga al SAC lo hace a través del login de Facebook. Facebook en caso de éxito nos devuelve un accessToken temporal para la sesión actual y un token único e invariable en el tiempo para cada usuario. SAC persiste el token persistente para recordar e identificar al usuario en futuras sesiones. Cuando un Friend se loga ocurre lo mismo, es con el token persistente del Friend como SAC reconoce la relación de confianza entre Owner y Friend.

El token persistente de Facebook en SAC lo hemos llamado “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.

En el 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 en este caso no tiene un Owner definido. Como 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.

### Camino “Index de Owner”

SAC reconoce al usuario logado como Owner y muestra “Index de Owner” se puede ver en la Ilustración 5.

### 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 persistente de Facebook) y el 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 en tabla friend. Una mejora propuesta en pág 54 es el poder actualizar la lista de Friends en distintos momentos.

## Crear Actions en SAC

Se persisten las Actions en SAC al consultar por los Actions de un WT desde la página [http://socialaccesscontroler/thing/{id}](http://socialaccesscontroler/thing/%7bid%7d%20) y es la primera vez que SAC consulta al Iot\_emulator por estos datos. En caso contrario evitamos volver a guardarlos.

Las Actions en Iot\_emulator es un concepto distinto al Actions del SAC. En Iot\_emulator un Action de un WT emulado es el equivalente en un proyecto no emulado con funciones a ejecutar en un objeto físico disponible vía web. Para SAC un Action son dos cosas, primero una cadena con la que reconocer una Action del Iot\_emulator y segundo una entidad de la base de datos para compartir con Friend.

## Acciones sobre los WTs

Explicamos las acciones más importantes que se pueden acometer en un WT en SAC, aunque mencionamos la base de datos o la API SAC no entramos en detalles técnicos. Se pueden consultar sobe base de datos en pág 37 y sobre API SAC en pág 32.

### Dar de alta nuevo WT

Cuando SAC recibe una petición de alta de nuevo WT. Persiste en tabla thing de la base de datos los datos llegados desde el formulario de alta estos endpoint, user y password, nada más.

Al igual que pasa con la diferencia entre Actions para Iot\_emulator y SAC los WTs son conceptos distintos en ambos proyectos. Un WT en Iot\_emulator emula un objeto físico disponible vía web. En cambio, en SAC un WT es un puntero hacia el WT del Iot\_emulator.

### Obtener información de un WT

Cuando SAC necesita la información de un WT la obtiene a través del Iot\_emulator. Los datos propios de SAC son el endpoint (root), usuario y contraseña. Usando estas credenciales y endpoint consulta a Iot\_emulator.

Este procedimiento se hace en varios puntos. Cuando se necesita es un listado de WTs SAC realiza esta lógica para cada WT del listado. A la hora de mostrar listado de Actions de un WT.

### Compartir un Action con un friend

Al compartir un Action con un Friend se genera una relación entre tabla action y tabla friend. Además se genera una URL vía API SAC y se muestra al Owner que la compartirá con el Friend.

## API SAC

Hemos visto la necesidad de crear un API SAC por varias razones. Primero la necesidad 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. Además, como veremos más adelante para poder usar el sistema de rutas se Symfony en peticiones Ajax.

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

En la Ilustración 13 es un flujo de requests y responses donde vemos como los datos que una página necesita se consultan posteriormente a la primera respuesta del SAC (flecha 2). De esta manera la respuesta en más rápida, cada página consulta sólo los datos que necesita y las consultas con información sensible (como usuario o contraseñas de un WT) quedan relagadas a llamadas entre servidores (flecha 4) y esta información no viaja al usuario. Esta ilustración además corresponde con el esquema clásico para peticiones Ajax. SAC lo usa en muchos puntos por ejemplo en la petición central del uso del SAC que es cuando un SAC obtiene un dato de un WT usando credenciales del Owner para mostrarlo a un Friend.

HACER LEYENDA

Request inicial

Response incial.

Frontend llama a SAC API vía Ajax

Request al Iot\_emulator

Response del Iot\_emulator

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 yaml.

Por ejemplo, si definimos una ruta con alias “nombre\_ruta” y con un parámetro recibido por url tal que “/ruta/{key\_param}”. Usando las herramientas que provee Symfony pordriamos resolver esta ruta desde frontend o backend de las siguientes maneras:

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>

Vemos como en ambos caso la ruta se resulve al “/ruta/value\_param”

#### 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 el Nombre 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 Actionel 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 creación de 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} |  |

Todo al menos API endopoints

## Arquitectura Hexagonal SAC

Vamos a explicar de qué manera se ha hecho una arquitectura hexagonal. Haciendo un repaso a cada capa vista en pág 5 . Al final veremos un ejemplo en la manera que hexagonal que SAC resuelve la manera de traer datos del Iot\_emulator. Como veremos la Entidad Thing de la capa de Dominio que tiene una propiedad llamada thingConnected es donde almacena la respuesta del Iot\_emulator en esta misma capa existe el Repositorio ThingConnectedRepository. En capa de Aplicación existen varios CommandHandlers que acceden hacen las consultas al Iot\_emulator cumpliendo el contrato establecido en ThingConnectedRepository. En la capa de Infrastructura hay Controlers tanto de API SAC como de la web como Commandos Symfony que usan dichos CommandHandlers.

### Dominio SAC

Explicación de capa de Dominio en pág 17.

// Entidades

src/Domain/Entity/Action.php

src/Domain/Entity/Friend.php

src/Domain/Entity/Owner.php

src/Domain/Entity/Thing.php

// Repositorios

src/Domain/Repository/OwnerRepository.php

src/Domain/Repository/ThingConnectedRepository.php

src/Domain/Repository/ThingRepository.php

src/Domain/Repository/ActionRepository.php

src/Domain/Repository/FriendRepository.php

Las cuatro Entidades son básicas para el funcionamiento de SAC. Dado su complejidad vamos a las propiedades privadas del Thing que son

private $id;  
private $root;  
private $user;  
private $password;  
private $actions;  
private $owners;  
private $thingConnected;

. El “id” es propio de la base de datos. “root” es el endpoint del WT en Iot\_emulator, tanto “user” como “password” son cadenas de texto que se usan como credenciales al logarse en WT de Iot\_emulator. Actions y Owners están relacionados con tablas homónimas y estar relaciones están resueltas con el ORM Doctrine. Explicamos la última propiedad privada en apartado de la pág 48.

### Aplicación SAC

Explicación de esta capa en (17).

Pensamos que la mayoría de nombre de Commands y CommandHandlers son bastante descriptivos. Como los Commands son DTOs puros las explicaciones las haremos en el apartado de CommandHandles SAC.

#### Commands SAC

// 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

// Comandos para ThingConnected

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

#### CommandHandlers SAC

Vamos a centrarnos en los CommandHandlers que requieren más atención, que son: GetFbSharingStatusByOwnerCommandHandler, SearchThingConnectedActionsHandler, SearchThingConnectedNameHandler, SearchThingConnectedBrandHandler, IsActualUserAnOwnerHandler y relacionado con la explicación de ThingConnected GetThingConnectedCompleteHandler

// 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

##### GetFbSharingStatusByOwnerCommandHandler

Este CommandHandler genera un array de arrays que hemos llamado GetFbSharingStatus que la usamos a la hora de construir la lista de amigos vista en Ilustración 10 para saber que Friend tiene compartida cual Action. Esta es la idea general para un único WT con id igual a 1. El contenido del array “actions” son los fb\_delegate de cada Friend

// estructura de shareStatus

[

1 => array:2 [

"thingId" => 1

"actions" => array:2 [

1 => "103671587486416"

2 => "104003390786397"

]

]

]

##### IsActualUserAnOwnerHandler

Lo usamos durante el proceso de logeo pág 29 y determina si el usuario logado es Owner del SAC, en caso contrario lanza Excepción.

##### SearchThingConnectedActionsHandler, SearchThingConnectedBrandHandler y SearchThingConnectedNameHandler

Estos tres Handlers consultan al Iot\_emulator y devuelven el nombre de WT la brand del WT o listado de Actions. Queremos destacarlos por que son los únicos Handlers que no tienen su Command específico. Es decir, los tres comparten el mismo Command en concreto este

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

Existen por que están definidos en el Repositorio ThingConnectedRepository. Traen toda la información del WT y luego filtran. Esta implementación se podría mejorar ya que tanto para el Brand como para el Name se pueden obviar las credenciales, no así para listado de Actions.

##### GetThingConnectedInfoCommandHandler

La explicación para este CommandHandler esta explicada en el ejemplo general hexagonal en pág 48.

### Infrastructura SAC

Explicación de esta capa en (17). Esta capa tiene mucha información pero es información que si bien siendo imprescindible puede ser substituida por otra sin que eso afectase a la funcionalidad del Dominio de la aplicación.

Empezaremos viendo la carpeta Resources y las configuraciones contenidas. Explicaremos cómo hemos dividido los controladores. En la parte de Repositorios del SAC veremos como la capa de Infrastructura implementa los contratos (interfaces) ofrecidos por el Dominio. En los Respositorios también terminaremos con el ejemplo iniciado en la parte de Dominio relacionado con ThingConnected. En serializdores vemos como Infrastructura se encarga de dar forma al Dominio. Hemos generado mucha cantidad de Comandos SAC tanto para el desarrollo como para pruebas. Finalemente veremos la manera en la que hemos restringido el acceso a Controladores enteros usando Event Subscribers.

#### 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

Cada colección de rutas está dividida según su funcionalidad, api.yaml para API SAC, tanto para el generador de rutas como para consultas hacia Iot\_emulator. Fichero credentials.yaml para lo relacionado con el Login. Los dos ficheros error.yaml y success.yaml para las páginas de erro y éxito respectivamente. El fichero owner.yaml tiene las rutas de todo el mapa web del Owner. El fichero friend.yaml el mapa web del Friend

// 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. Es en 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 pág 40. OwnerApiController es donde SAC consulta información del Owner un ejemplo de uso de endpoint está en pág 42. Al igual un ejemplo para ThingApiController lo tenemos en pág 43.

// 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

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

Los Respositorios relacionados con las Entidades de Dominio Action, Friend, Owner y Thing son Repositorios sencillos que implementan los Respositorios de Dominio. Queremos destacar el método save de MySQLActionRespository es el único que posee una salvaguarda al guardar una Entidad, la explicación a esto está se puede ver en pág 31. En siguiente código se ve el método save del Repositorio mencionado. Como se puede ver si el Action ya existe no se crea de nuevo.

DUDA monunmental, esto de $this->em->getRepository(Action::class) es un burrada,no? No será mejor en contsturctor ya devinir un $actionRespositrory?

public function save(string $name, Thing $thing): Action  
{  
 $actionRepo = $this->em->getRepository(Action::class);  
  
 */\*\** ***@var*** *Action $action \*/* $action = $actionRepo->findOneBy(['thing' => $thing->getId(), 'name' => $name]);  
 if ($action) {  
 return $action;  
 }  
 $action = new Action($thing, $name);  
 $this->em->persist($action);  
 $this->em->flush();  
 return $action;  
}

#### 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

### Ejemplo hexagonal: ejemplo de ThingConnected.

En esta sección vamos a ver una manera hexagonal de resolver como traer datos del Iot\_emulator.

#### ThingConnected en capa Dominio

En el dominio la Entidad Thing tiene una propiedad privada con getter y setter llamada thingConnected además hay definido un Repositorio

src/Domain/Repository/ThingConnectedRepository.php

ThingConnectedRepository define una interfaz de los métodos a implementar para obtener los datos del WT del Iot\_emulatro.

#### ThingConnected en capa Aplicación

En capa de Aplicación existen estos GetThingConnectedInfoCommand y GetThingConnectedCommandHandler relacionado con el Repositorio ThingConnectedRepository.

// Command

src/Application/Command/Thing/ThingConnected/GetThingConnectedInfoCommand

//CommandHandler

src/Application/CommandHandler/Thing/ThingConnected/GetThingConnectedCompleteHandler

El GetThingConnectedInfoCommand es un DTO muy simple que sólo transporta a la Entidad Thing. E GetThingConnectedCompleteHandler recibe por constructor una implementación de ThingconnectedRepository (luego veremos en capa deInfrastructura como se construye).

public function \_\_construct(ThingConnectedRepository $thingConnectedRepository,

Como vemos en el código del handle recibe el GetThingConnectedInfoCommand del que saca el Thing.Posteriorimente se usa un método del ThingConnectedRepository llamado getThingConnectedCompleteByIdOrException al que le pasa la información del Thing el enpoint (root) y las credenciales.

public function handle(GetThingConnectedInfoCommand $getThingConnectedInfoCommand)  
{  
 */\** ***@var*** *Thing $thing \*/* $thing = $getThingConnectedInfoCommand->getThing();  
 $thingConnected = $this->thingConnectedRepository->getThingConnectedCompleteByIdOrException($thing->getRoot(), $thing->getUser(), $thing->getPassword());

El CommandHandler está orquestando en cuanto que provee a métodos de la información que precisan para ejecutarlos. También porque está poniendo en juego varias capas, primero está usando una implementación de un Repositorio de Domino, sabemos que es una implementación porque es interfaz y como veremos más adelante la implementa la capa de Infraestructura.

#### ThingConnected en capa Infrastructura

Acabando con el ejemplo del ThingConnected vemos en el siguiente Repositorio de Infrastructura que es donde implementamos el Repositorio definido en Dominio.

src/Infrastructure/ThingConnected/CurlThingConnectedRepository.php.

Este Repositorio tienen la particularidad de que en su implementación usa Curl como fuente de sus datos y no MySQL como hacen el resto . En este código se puede ver como usa la implementación de Curl de PHP para hacer consulstas GET al Iot\_emulator.

private function sendCurlOrException($thingRoot, $thingUserName, $thingPassword)  
{  
 $ch = curl\_init($this->iotEmulatorHost . '/' . $thingRoot);  
 curl\_setopt($ch, *CURLOPT\_CUSTOMREQUEST*, "GET");  
 curl\_setopt($ch, *CURLOPT\_PORT*, $this->iotEmulatorPort);  
 curl\_setopt($ch, *CURLOPT\_RETURNTRANSFER*, true);  
 curl\_setopt($ch, *CURLOPT\_HTTPHEADER*, array(  
 'user: ' . $thingUserName,  
 'password: ' . $thingPassword  
 )  
 );

$json = curl\_exec($ch);

## Seguridad SAC

A la hora de tener seguridad, desarrollar y hacer las pruebas hemos seguido las siguientes premisas.

* Usar https. Facebook nos impuso esta condición.
* Sólo el Owner puede acceder al “Mapa web para Owner”
* Solo Friend puede acceder a su “Mapa web para Friend”, y no al de otro Friend
* Personas sin Facebook no podrán salir del login
* Personas que no son Friends de Owner no podrán salir del login
* Ningún Friend podrá ver Actions no compartidas con él.

# 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

Hemos dedicido usar Facebook al ser una red social muy usada y facilitar el trabajo de desarrolladores, por ejemplo con la creación de perfiles falsos.

### Facebook API

* Autenticación delegada.
* Consulta de lista de amigos.

### Facebook.developers

* Creación de usuarios de prueba
* Login
* Creación de aplicación web
* Creación de perfiles falsos

## 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 bagaje y centrado en la robustez. Se han usado las siguientes capacidades de Ubuntu

### variable de entorno

* $USER: almacena el valor del nombre del usuario logado. Lo usamos para poder desarrollar en distintas máquinas y poder compartir comandos
* $IOT\_EMULATOR: definimos esta variable con la url del servidor (local, pruebas o producción) que tuviese el iot\_emulator.
* $SAC: definimos esta variable con la url del servidor (local, pruebas o producción) que tuviese el SAC.

### sshfs:

Permite para montar en local vía ssh sistema de ficheros de AWS, así poder trabajar con phpstorm directamente en el servidor

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 -'

### httpie

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

/tests/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 ec2

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

## nginx

Hemos usado nginx por su facilidad a la hora configurar los subdmonios y el https.

### Configuración Iot\_emulator

# Virtual Host configuration for iot.socialaccesscontroller.tk  
#  
server {  
 listen 80;  
 listen [::]:80;  
  
 server\_name iot.socialaccesscontroller.tk;  
  
 root /var/www/iot\_emulator/public;  
 index index.php;  
  
 error\_log /var/log/nginx/iot.socialaccesscontroller\_error.tk debug  
 access\_log /var/log/nginx/iot.socialaccesscontroller\_access  
   
 location / {  
 try\_files $uri /index.php$is\_args$args;  
 }  
  
 location ~ \.php {  
 try\_files $uri =404;  
 fastcgi\_split\_path\_info ^(.+\.php)(/.+)$;  
 include fastcgi\_params;  
 fastcgi\_param SCRIPT\_FILENAME $realpath\_root$fastcgi\_script\_name;  
 fastcgi\_param SCRIPT\_NAME $fastcgi\_script\_name;  
 fastcgi\_index index.php;  
 fastcgi\_pass unix:/var/run/php/php7.2-fpm.sock;  
 }  
}

### Configuración SAC

server {  
 listen 443 ssl;  
 listen [::]:443 ssl;  
 include snippets/snakeoil.conf;  
 index index.php;  
 error\_log /var/log/nginx/socialaccesscontroller.error.log;  
 access\_log /var/log/nginx/socialaccesscontroller.access.log;  
  
 root /var/www/sac/public;  
 server\_name socialaccesscontroller.tk;  
 location / {  
 try\_files $uri /index.php$is\_args$args;  
 }  
  
 location ~ \.php {  
 try\_files $uri =404;  
 fastcgi\_split\_path\_info ^(.+\.php)(/.+)$;  
 include fastcgi\_params;  
 fastcgi\_param SCRIPT\_FILENAME $realpath\_root$fastcgi\_script\_name;  
 fastcgi\_param SCRIPT\_NAME $fastcgi\_script\_name;  
 fastcgi\_index index.php;  
 fastcgi\_pass unix:/var/run/php/php7.2-fpm.sock;  
 }  
}

## html5 , CSS, Bootstrap y moustache

Si bien este Trabajo Fin de Máster no tiene demasiado frontend. Al ser una aplicación web Hemos usado html en la parte del frontend. Para ser más maquetado con css y Bootstrap.

## javascript, Jquery y Jquery-UI

Hemos usado javascript porque se puede usar en todos los navegadores web. Además nos permite que algunos cálculos se hagan en ordenador de cliente y no en servidor liberando al servidor de carga de trabajo. Hemos usado Javascript para recorrer estrcturas de datos básicamente json o construir elementos HTML vía DOM.

Jquery es una biblioteca que facilita el uso de javascript. Jquery lo hemos usado para completar lo que hacíamos con javascript puro pero sobre todo para las llamadas Ajax.

Jquery-Ui es otra librería complementaría a jquery especializada en GUIs. La hemos usado a la hora de construir el pop-up donde aparecen los Friends de un Owner.

## symfony 4

Hemos usado este framework PHP por ser muy seguro y ofrecer componentes desacoplados y reutilizables.

## npm

Npm es un gestor de paquetes. Lo hemos usado para instalar en SAC moustache.

## mysql y Doctrine

MySQL es un sistema a de gestión de base de datos relacional gratuita muy usada en aplicaciones web. Lo hemos usado tanto en SAC como en Iot\_emulator para persistir datos que debían almacenarse.

Doctrine es un ORM muy integrado en Symfony

MysqlWorkbench es XXX. Hemos construido los esquemas en esta aplicación y hecho ingeniería inversa para mostrar las ilustraciones.

## COMPOSER

Composer es un gestor de librerías para PHP. Lo hemos usado para la instalación de:

* Symfony
* phpUnit
* para tener un servidor de PHP local
* Bootstrap
* Jquery
* Jquery-ui
* Html5-boilerplate
* Maker-bundle que proporciona comandos útiles al desarrollador

# Análisis resultados

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.

ssssssssssssssssssssssssssssssssssssssssss

Se ha conseguido crear un entorno seguro donde un Owner puede elegir qué Actions compartir y a quien compartirlas las relaciones en facebook se mantienen dentro de SAC.. Es posible almacenar las credenciales y los permisos de los WTs de manera segura en SAC y dejar la estructura de la red de contactos 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).

Usando SAC el Onwer no tiene porque compartir sus credenciales a Friends tampoco necesita buscar WTs multiusuarios. Como el SAC conoce la credencial del Owner no se necesitan credenciales extra, con una por WT basta. Además algo único del SAC es la capacidad de compartir cualquier Action y no el WT entero. Algo que cediendo las credenciales al Friend definitivamente pierdes.

## Relación con Asignaturas del máster

### Entorno Web

Esta asignatua nos ha servido en todas las fases del Proyecto. Durante el Desarrollo tuvimos que montar muchas veces el entorno. Primero se virtualizó con vagrant luego por falta de recursos se montaba en local. Como los requisitos aumentaban y la máquina local era insuficiente tuvimos que cambiar de ordenador. En el el cuerpo del trabajo no se ha mencionado por falta de automatismo pero en cada Proyecto existe este script con commandos para montarlos.

CI/amazon-provision.sh

Además en SAC existe una configuración de ansible que también la usamos. Pero por la misma razón de antes (falta de automatismo) no se ha incluido en el cuerpo de la memoría.

CI/deploy/playbook.yml

CI/deploy/Vagrantfile

Por otro lado el uso de git es básico aunque sea un desarrollo hecho por una persona. Sobre to

* AWS EC3
* variables entorno
* httpie
* nginx

### Bases de Datos

* MySQL
* Doctrine

### Maquetación web

Este Trabajo Fin de Máster no tiene una alta carga de trabajo en frontend. Aún asi hemos usado tecnologías web y de maquetación como son:

* html5
* javascript
* jquery
* jquery-ui
* moustache
* bootsctrap

### Frameworks

Hemos usado muchas características de Symfony como los Comandos, Fixtures, Event Susbscribers, enrutamiento, servicios.

### Emprendiduría

Este trabajo se puede definir como una prueba de concepto en un proceso muy temprano de emprendimiento.

### Desarrollo Eficiente

* Arquitectura hexagonal
* Patrones de diseño
  + :DTO
  + Inyección de dependencias

Lenguaje ubiquito, Nombres semánticos de métodos

### RENDIMIENTO

### SEO

Hemos respetado el SEO al generar las rutas de las APIs.

### Seguridad

La seguridad a la hora d

### PHP

Toda la lógica de negocio está programada en PHP.

# Conclusión

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 WtTs 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 WTs.

Internet of things

RRRRRRRRRRRRRRRRR

"No es el resultado 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

Mención a tencologias que más has usado y como las he usado: arquitectra hexagonal, framework web php: hacer comandos me ha ayudado. Bases de datos relacionales. “Un poco como” resumen de tecnologías usadas

## Trabajo futuro

### Multi-owner

Hacer que SAC pueda soportar varios Owner distintos. Habría que hacer una página al inicio que permitiese al usuario decidir cual rol quiere asumir. De tal manera que una misma persona pueda entrar como Owner y como Friend en distintas sesiones.

### 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.

# Analisis economico del proyecto

### Diseño

20 horas

### Implementación

60 horas

### Pruebas

15 horas

### Memoría

20 horas

# 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
* **WT Brand** (Brand): Es la compañía que ha construido el objeto físico
* **WT Name** (Nombre): Es el nombre que recibe el objeto físico, sirve para que diferenciarlo entre ellos.
* **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" https://vs.inf.ethz.ch/publ/papers/dguinard-sharin-2010.pdf

2- W3Consortiun WOT Model <https://www.w3.org/Submission/2015/SUBM-wot-model-20150824/>