



**Escola Tècnica Superior d'Enginyeria
Electrònica i Informàtica La Salle**

Trabajo Fin de Máster

Máster en Programción Web de Alto Rendimiento

Alumno
Daniel Salgado Población
Codina

Profesor Ponente
Víctor Caballero

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
TRIBUNAL

VOCAL DEL

PRESIDENTE DEL TRIBUNAL

SOCIAL ACCESS CONTROLLER

Universitat Ramon Llull – La Salle Barcelona

Daniel Salgado Población

1 de septiembre de 2019

GLOSARIO

- **Web Thing (WT).** Objeto con conexión a internet que ofrece información interna vía http(s) y arquitectura REST. En el presente proyecto *WT* devuelven formato de datos JSON. Posee dos zonas:
 1. Zona Pública
 - Nombre
 - Brand
 2. Zona Privada
 - Action
 - Property
- **Property:** Almacena un estado de un *WT*.
- **Action:** Es una iteracción con un *WT* que permite invocar una función en un *WT*. Una acción permite ver el estado de una *Property*.
- **lot_emulator:** recurso web donde configurar emulaciones de conjuntos de *lots*.
- **Owner:** persona que posee *lots*, conoce las credenciales para acceder a zona privada de *lot*. Además 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*.

RESUMEN

Internet of things: gestionar dispositivos, q la batería dure más

Web of Things: capa por encima de lot que soluciona como describir y comunicar los dispositivos q están conectados

Resumen de todo lo hecho

Web of Thing describe un modelo común para

1 Internet of Things (lot) 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 (Web Things).

2 El Web of Things propone un modelo común para describir contrapartes virtuales de los dispositivos físicos del Internet of Things.

3 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.

Por no disponer de *lot* y para abordar la accesibilidad de los *WT* hemos creado un **emulador de lot** que nos proporciona colecciones de *WTs*.

Quitar info sobre Owner Actions en siguiente párrafo

Para abordar la compartición de los *WTs* hemos creado el **Social Access Controller** hemos implementado un sistema para compartir *Wts*. WEB THING esta seguridad y delegarla en redes ya construidas para esto, como puede ser Facebook. Es esta aplicabilidad la que hemos querido implementar con el Social access controller (SAC). Donde poder compartir con otras personas información de internet of things (lot) propiedad de 1 dueño. Usando facebook para saber la red de amigos conectadas con un *Owner*. El *Owner* puede decidir cuales *Actions* muestra a cada *Friend*. SAC almacena la mínima informacion posible en su sistema, y consulta en tiempo real lo que el *Owner* o *Friend* están demandando.

El grueso de la información se encuentra en Facebook y en los propios lots. SAC es un intermediario entre la red de amigos y las credenciales en os lots.

ABSTRACT

ÍNDICE GENERAL

INTRODUCCIÓN

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

Entender mejor la complejidad

Poner introducción a lot y WT

1. 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*.

2. Problemas de compartición de *WT* a resolver

Para poder compartir *WT* con *Friend* el *Owner* tiene estas soluciones pero tienen varios problemas

I. *WT* multiusuario

Un *WT* multiusuario es aquel que puede tener varios usuarios accediendo y conceder permiso a cada *Friend* para dicho *WT*.

Problemas:

- *Owner* debe conocer la manera de dar de alta un usuario nuevo a su *WT*.
- Todos los *WT* deben ser multiusuario
- Permiso a nivel *WT*

II. Compartir credenciales

Compartir las credenciales con *Friend*.

Problemas:

- *Owner* debe confiar en la buena fe del *Friend* ya que pierde el control de la credencial
- *Owner* a llevar de alguna manera el control de a quién dejó cual *WT*
- Permiso a nivel de *WT*

3. Requisitos

Para resolver las anteriores problemáticas proponemos diseñar una aplicación. Que permita compartición a los distintos *WTs* de esta manera compartir los *WTs* entre *Owner* y *Friend*

Requisitos

1. Un *Owner* tiene un lugar único donde administrar la compartición de *WTs* poseídas.

2. La aplicación se loga al *WT* con las credenciales de *Owner*.
3. Un *Owner* puede compartir indistintamente distintos *Actions* de un mismo *WT*.
4. Las *Actions* compartidas pueden serlo a número indeterminado de *Friends*.
5. La aplicación almacena la cantidad mínima e imprescindible de información
 1. La red de amigos es almacenada externamente a la aplicación
 2. La información de *Actions* y *Properties* se consultan en caliente

4. Objetivos

El objetivo primario del Trabajo Fin de Máster es crear una aplicación que cumpla los requisitos anteriormente mencionados. Y así poder evaluar como compartir de manera eficiente y segura y fácil de *WTs* a *Friends*. Pero para llevarlo a cabo hemos construido una capa de accesibilidad

III. Compartición mediante Social Access Controller (SAC)

Siguiendo la idea de un SAC propuesta por Dominique Guirnard *et al* en [LINK1] y en concreto la idea de la Compartición basada en Redes Sociales

El *Owner* accederá al SAC por primera vez usando Facebook. SAC almacena el *user_id*. A partir de entonces SAC usa una autenticación delegada en Facebook para saber quién es *Friend* del *Owner*

El *Owner* da de alta los *WTs* en SAC. De esta manera SAC obtiene las credenciales para logarse como *Owner*

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*).

IV. Cumplimiento de requisitos

Al administrar la compartición desde SAC el *Owner* tendrá un lugar único para administrar su *WT*. Requisito 1

Como se loga con las credenciales del *Owner* para el *WT* SAC es el *Owner*

- Por lo que SAC podrá ver todos individualmente cualquier *Actions* de un *WT* permitiendo compartir indistintamente cualquier *Action*. Requisito 2.
- SAC podrá consultar en cualquier momento cualquier *Action* de un *WT*. Requisito 5.2

Usaremos autenticación de Facebook Delegada. La red de *Friends* está definida fuera del SAC. Requisito 5.1

V. Capa de accesibilidad mediante Iot Emulator

Para disponer de *WTs* accesibles y poder trabajar hemos creado una aplicación que funciona a modo de hub de *WTs*. Esta aplicación se llama *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.

DISEÑO GENERAL

En esta sección englobamos aquel conocimiento que es compartido por `lot_emulator` como por SAC. Tanto estructuras de datos, tecnologías o procedimientos.

Para realizar el trabajo fin de máster hemos simplificado los modelos propuestos tanto por [LINK1] y [LINK2]. Hemos hecho estas simplificaciones por restricciones de tiempo o por no añadir complejidad innecesaria.

En las siguiente sección repasamos

5. Acople al modelo *WT* del W3Consortium

Los diferentes requisitos

Requirimientos sacados de [LINK2] en apartado 5. *Web Thing requirements*
<https://www.w3.org/Submission/2015/SUBM-wot-model-20150824/#web-things-requirements>

Para las tabla

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

Nivel 0 – Un *WT* DEBE

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

Nivel 1 – *WT* DEBERÍA

Definición de requisito	Nivel de cumplimiento
A Web Thing SHOULD use secure HTTP connections (HTTPS)	Sí
A Web Thing SHOULD implement the WebSocket Protocol	No
A Web Thing SHOULD support the Web Things model	Sí

A Web Thing SHOULD return a 204 for all write operations	Sí
A Web Thing SHOULD provide a default human-readable documentation	No

Nivel 2 – WT PODRÍA

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

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

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

6. 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

Expicar arquitectura hexagonal

7. Comandos Symfony

En capa hexagonal de Infraestructura hemos hecho uso de Comandos Symfony que usan Commands y CommandHandlers de la capa de aplicación.

Se ejecutan desde la raíz del proyecto desde la terminal,

Todos tiene el prefijo **app** (dos puntos) seguidos del nombre del **dominio al que apliquen** (dos puntos) seguidos del **Commando de la aplicación** que ejecutan

```
php bin/console app:Dominio:Commando
```

8. Endpoints

Se puede encontrar una explicación detallada en cada secciones de arquitectura de *lot_emulator* (Arquitectura REST *lot_emulator*19 más adelante) como del SAC(Arquitectura REST SAC).Aquí mostramos simplemente un listado de todos los endpoints la explicación detallada se puede encontrar en las arquitecturas de cada subapartado

En las siguientes tablas () esta es la explicación de cada columna

- Verbo HTTP: método de petición para iniciar la acción indicada. POST, GET, PUT, DELETE.
- Endpoint: interfaz expuesta vía URL

VI. lot_emulator

Se puede consultar el detalle en esta sección (Arquitectura REST lot_emulator19 más adelante)

Verbo HTTP	Endpoint
GET	/
GET (sin credenciales)	/id
GET (con creenciales)	/id
POST	/create
GET	/id/actions/action_name
GET	/id/properties/property_name
GET	/id/actions
GET	/url

VII. SAC

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

Verbo HTTP	Endpoint
	/
	/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
	/conditions
	/error
	/friend/thing/thingId/action/actionId
	/friend
	/owner
	/owner/Friends
	/owner/things
	/owner/create
	/success
	/thing/create
	/thing/thingd

9. Lanzamiento Peticiones HTTP

Es necesario lanzar peticiones HTTP A la hora de desarrollar una API lanzar peticiones es

VIII. Herramientas

- Curl
- Cliente HTTP de phpstorm
- Httpie

La inconveniencias del cliente HTTP de phpstorm es la necesidad de phpstorm para lanzar peticiones. Por eso progresivamente lo

<code>docs/request</code>

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
- lot_emulator

10. 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 Facebook]. 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.

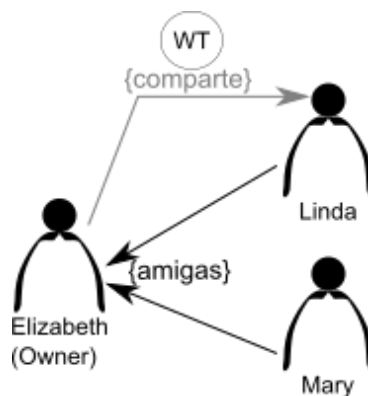


Ilustración 2
Compartición entre amigas de
Owner

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]

11. WT de pruebas lot_emulator:

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

Hemos creado script fixture/create_things.php que mediante peticiones POST puebla la base de datos de *iot_emulator* con things.

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

```
{
  "name": "thing_name1",
  "brand": "thing_brand1",
  "links": {
    "actions": ["action_name1"],
    "properties": [{"action_name1": "property_value1"}]
  }
}

{
  "name": "thing_name2",
  "brand": "thing_brand2",
  "links": {
    "actions": ["action_name1", "action_name2"],
    "properties": [{"action_name1": "property_value1"}, {"action_name2": "property_value2"}]
  }
}

{
  "name": "thing_name3",
  "brand": "thing_brand3",
  "links": {
    "actions": ["action_name1", "action_name2", "action_name3"],
    "properties": [{"action_name1": "property_value1"}, {"action_name2": "property_value2"}, {"action_name3": "property_value3"}]
  }
}
```

12. Datos pruebas SAC

Hemos poblado la base de datos de SAC con herramienta propia de symfony.

Hemos testado peticiones en carpeta ‘docs/requests’

Para lanzar peticiones http usamos cliente integrado en symfony y tambien httpie. Estas se encuentran en la carpeta docs/requests

- TODO mirar y determinar si se usa o no el src/DataFixtures/Sac.php

- usamos symfony fixtures. Estos datos se usaron durante el desarrollo pero no se recomiendan para las pruebas funcionales. Ya que no guardan consistencia con los datos de *iot_emulator*

Son malos

13.

14. Sadfasd

TESTS

15. iot_emulator:

Hemos hecho 2 tipos de tests.

- Basados en phpunit
- Hecho en PHP

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

IX. no usan phpunit

```
php tests/notPHPUnit/get_actions/get_actions.php  
php tests/notPHPUnit/get_thing/get_thing.php.php  
php tests/notPHPUnit/isIntegrityValidOnCreate.php
```

X. usan phpunit

TODO configurar para ejecutar phpunit

16. SAC

BACKEND IOT_EMULATOR

URL: <http://iot.socialaccesscontroller.tk/>

Hemos usando 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

17. Estructura básica *WT*

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

XI. Zona pública y Zona privada

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

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

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

XII. Respuesta JSON

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

```
http://iot.socialaccesscontroller.tk/1
```

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

```
{
  "id": 1,
  "name": "thing_name1",
  "brand": "thing_brand1",
  "links": {
    "actions": {
      "link": "\/actions",
      "resources": {
```

```

        "action_name1": {
            "values": "property_value1"
        }
    }
}

```

XIII. 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* coincide con el valor de *Action*. Siguiendo el ejemplo anterior [LINK al código anterior]

```
url/actions/action_name1
```

Devolvería

```
property_value1
```

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

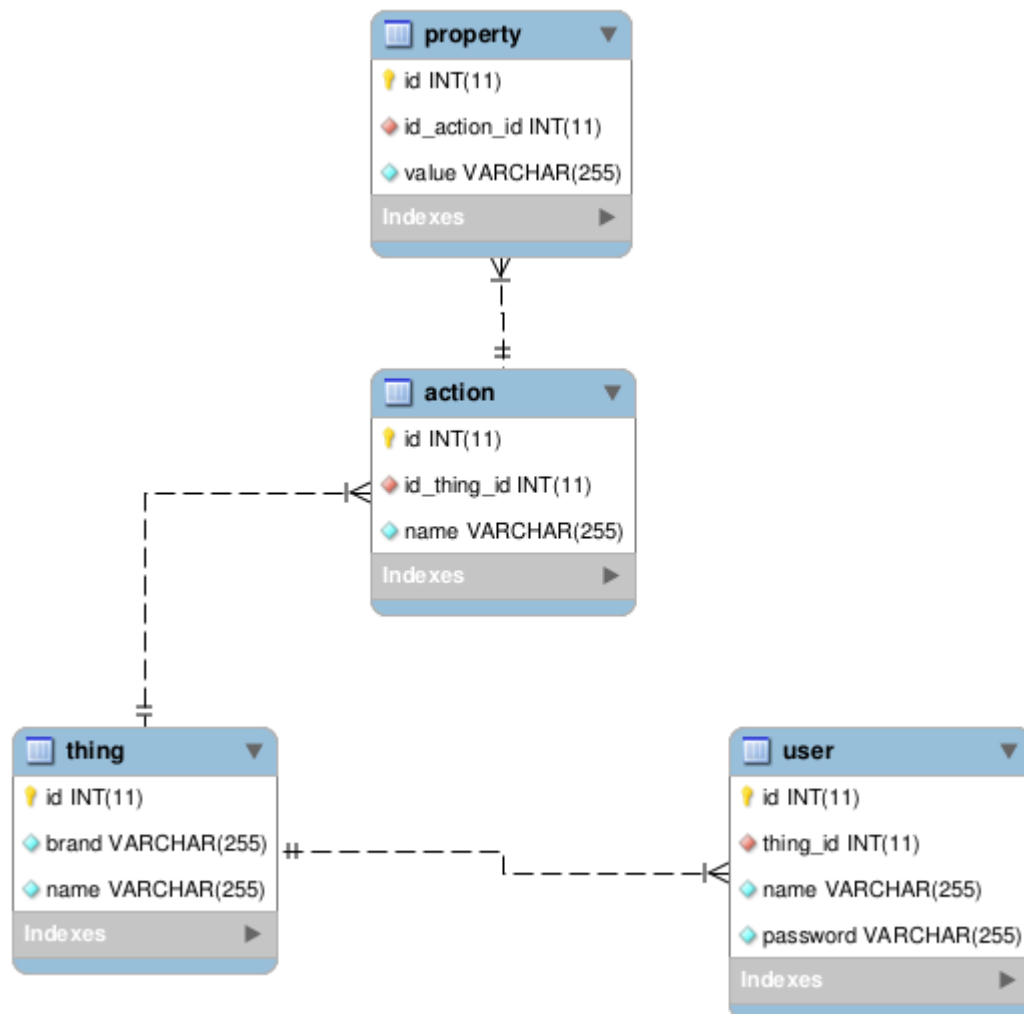
XIV. *WT* Endpoints

Los endpoints de los *WT* equivalen a las claves primarias de la tabla thing de la base de datos. Los iots se identifican con ids numéricos que coinciden con el id interno de la base de datos. Así thing_1 será /1 y será el id con pk=1 en tabla thing.

18. Esquema Base de datos lot_emulator

Explicar TABLAS

El usuario para acceder a esta base de datos debe tener estos permisos: DDL ALTER, AND DML SELECT, INSERT, UPDATE, DELETE



19. Arquitectura REST *lot_emulator*

TODO explicar mejor cada endpoint

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

La url `/` que no está c

Verbo HTTP	Endpoint	Descripción
	<code>/</code>	Index de las zonas públicas de los <i>WTs</i>
	<code>/id</code>	
	<code>/id</code>	
	<code>/create</code>	
	<code>/id/actions/{action_name}</code>	
	<code>/id/properties/{property_name}</code>	
	<code>/id/actions</code>	
	<code>/url</code>	

XV. ¿COMO ACCEDER A LISTADO IOTS?

La raíz de '/' `iot_emulator` muestra una lista de la parte pública de todos los iots almacenados

20. Arquitectura hexagonal de `lot_emulator`

XVI. DOMINIO

Para mapear las entidades con la base de datos hemos las @anotations interpretadas por el ORM Doctrine

```
src/Domain/Entity/Action.php
src/Domain/Entity/Property.php
src/Domain/Entity/Thing.php
src/Domain/Entity/User.php

src/Domain/Repository/ActionRepository.php
src/Domain/Repository/PropertyRepository.php
src/Domain/Repository/ThingRepository.php
src/Domain/Repository/UserRepository.php
```

XVII. APLICACIÓN

Estructura de Command y CommandHandlers

Existe una relación 1 a 1 entre todos los Commands y sus CommandHandlers

```
// 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

Usamos este patrón de diseño para transmitir la información de las credenciales recibidas, es una estructura de datos independiente a nuestro modelo de datos


```
src/Application/Dto/UserCredentialsDto.php
```

XVIII. Infraestructura

Controladores

Procesan la Request y para construir una Response

```
src/Infrastructure/FallbackController.php  
src/Infrastructure/ThingController.php
```

Repositorios

Implementan las interfaces del Dominio

```
src/Infrastructure/MySQLActionRepository.php  
src/Infrastructure/MySQLPropertyRepository.php  
src/Infrastructure/MySQLThingRepository.php
```

Comando de Symfony

Se ejecuta desde la terminal

```
src/Infrastructure/Thing/Command/SearchThingByThingIdCommand
```

Ejemplo de ejecución del Comando de Symfony

```
php bin/console app:Thing:S 1 user password | jq  
{  
  "id": 1,  
  "name": "thing_name1",  
  "brand": "thing_brand1",  
  "links": {  
    "actions": {  
      "link": "/actions",  
      "resources": {  
        "action_name1": {  
          "values": "property_value1"  
        }  
      }  
    }  
  }  
}
```

```
}

```

Serializadores

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

```
src/Infraestructura/Thing/Command/Serializer/ThingActions
src/Infraestructura/Thing/Command/Serializer/ThingWithCredetials
src/Infraestructura/Thing/Command/Serializer/ThingWithoutCredentials
```

21. Seguridad *lot_emulator*

Las peticiones con credenciales incorrectas

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

Las peticiones sin credenciales

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

En caso contrario y para no dar información sensible a un posible atacante devolvemos HTTP 400 "Resource not found"

FRONTEND SAC

URL: <https://socialaccesscontroller.tk>

Desde un punto de vista funcional y visual el SAC tiene tres varias partes diferenciadas. La primera es la “Raíz del proyecto” donde se desencadena el proceso de Login [31]. Las segunda es para usuario con rol *Owner* y la tercera para usuario con rol *Friend*.

En apartado “Raíz del proyecto” [27] mostramos el frontend de la entrada para cualquier rol. En apartado “Mapa web para *Owner*” [27] mostramos y explicamos las funcionalidades disponibles para *Owner*. En apartado “Mapa web para *Friend*” [27] mostramos y explicamos las funcionalidades disponibles para *Friend*.

Tal como explicamos en backend [31] solo el *Owner* y sus *Friends* podrán acceder más allá de “Raíz del proyecto”

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

22. Raíz del proyecto

La ilustración [27] es una captura de pantalla que 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 sección backend [31] y la información sobre los datos guardados por SAC está en [34]

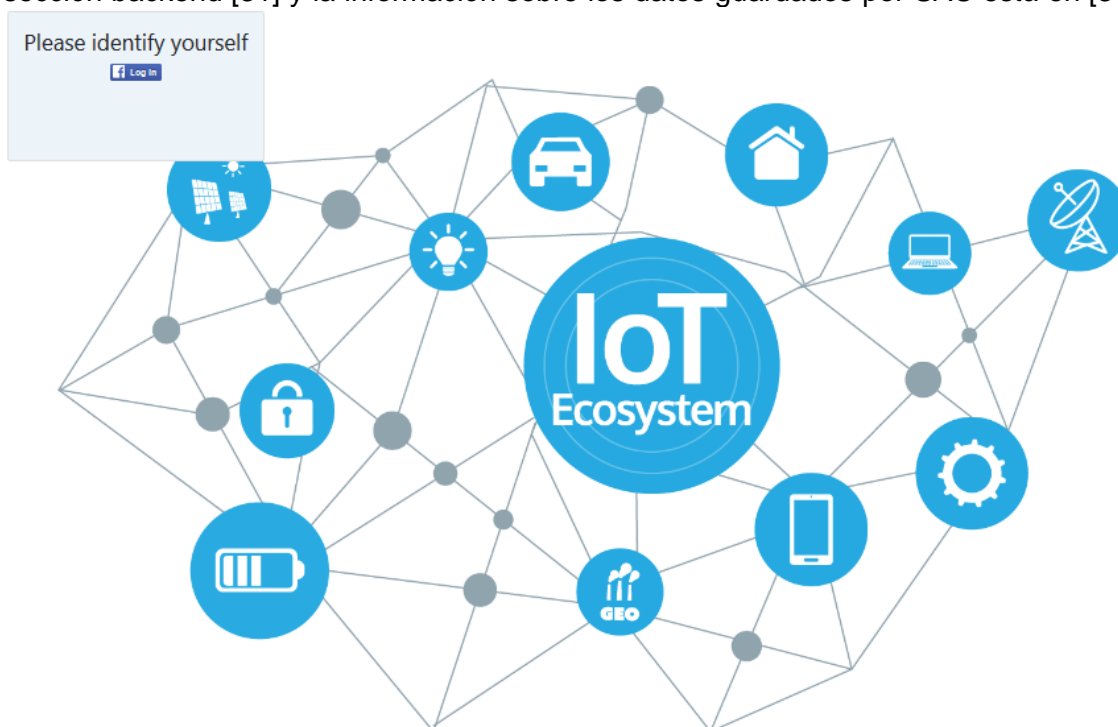


Ilustración 3 captura de pantalla de la Raíz del proyecto

23. Mapa web para Owner

XIX. Index del Owner

Endpoint: <https://XXXXTODO>

La ilustración [] es una captura de pantalla que un Owner. Desde esta pantalla un Owner puede ver información recabada general del Owner []. Listado de todos los WTs dados de alta en SAC[] con su información sobre el estado de conexión. Un formulario para dar de alta nuevo WTs.

El detalle de como se acomenten estas funcionalidades se explica en sección backend []

Información general del Owner

La información general del Owner se ve en ilustración[] es obtenida de Facebook durante la creación de nuevo Owner [32] y almacenada en SAC.

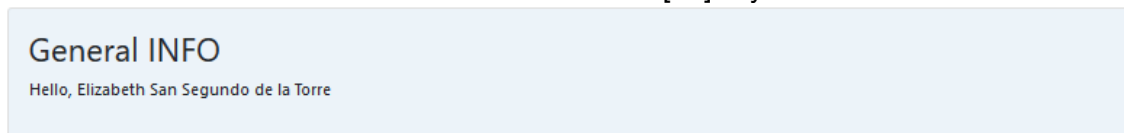


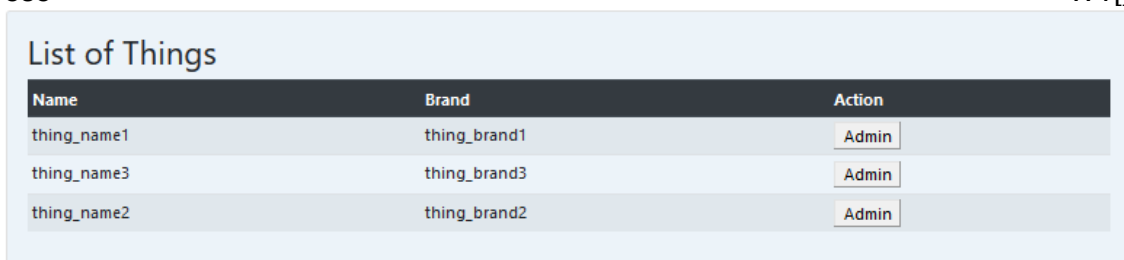
Ilustración 4 Información general del Owner vista en Index del Owner

Listado de WTs dados de alta en SAC

La ilustración[] es una captura de pantalla donde se ve el listado de todos los WT dados de alta por Owner en SAC. En este ejemplo existen tres WT dados de alta cuyos nombres son *thing_name1*, *thing_name2* y *thing_name3*. Corresponden con los datos de prueba de lot_emulator usados durante el desarrollo y explicado en este apartado [18].

Cada WT tienen la información “nombre del WT” “Brand del WT”. El color de las letras determina si SAC ha conectado correctamente con el WT. Siendo la letra negra si SAC ha podido conectarse al WT (conexión exitosa). El texto en rojo muestra el error encontrado.

Cada WT muestra un botón “Admin” para navegar a la página donde compartir ese WT[].



Name	Brand	Action
thing_name1	thing_brand1	Admin
thing_name3	thing_brand3	Admin
thing_name2	thing_brand2	Admin

Ilustración 5 Listado de todos los WT que se muestran en Index del Owner

Formulario para dar de alta nuevo WT

La ilustración[] es una caputra de pantalla donde se muestra el formulario para dar de alta nuevo WT. El Owner debe conocer el endpoint y las credenciales para cada WT e introducirlas en este formulario. En caso de éxito SAC mostrará la página de “Success” en caso de Error mostrará la página de “Error” informando del mismo.

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

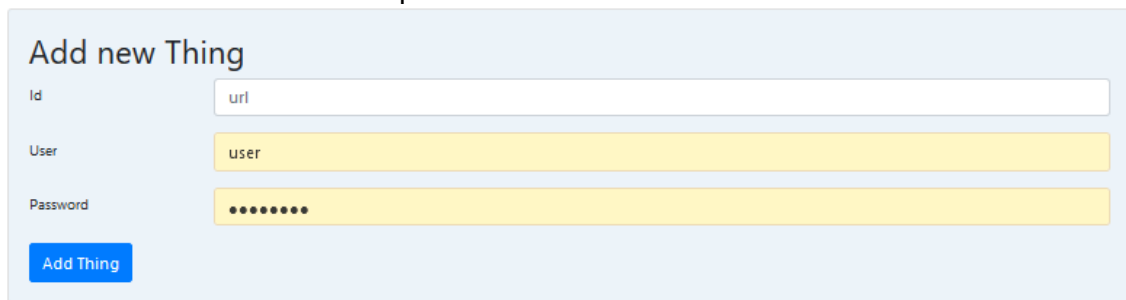
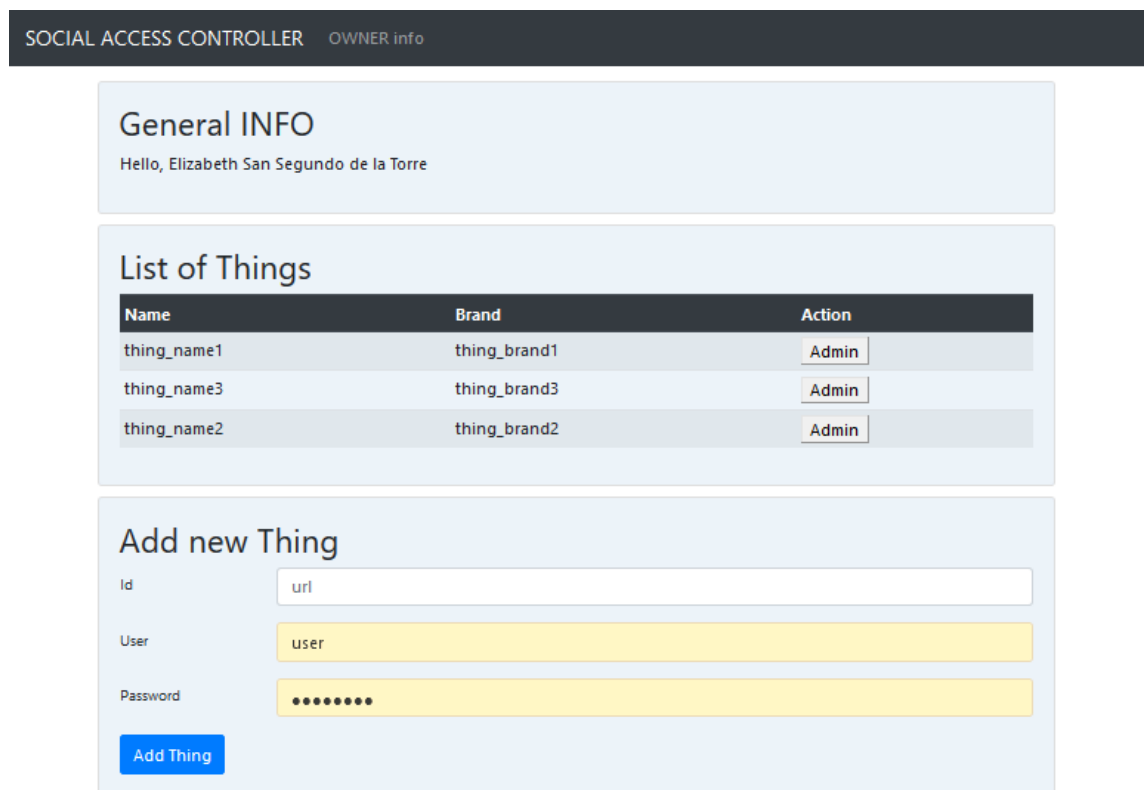


Ilustración 6 Formulario para dar de alta nuevo WT que se ve en Index del Owner



Name	Brand	Action
thing_name1	thing_brand1	Admin
thing_name3	thing_brand3	Admin
thing_name2	thing_brand2	Admin

Ilustración 7 Visión general del Index del Owner

24. Index de Thing

Endpoint: <https://socialaccesscontroller.tk/thing/{id}>

Se accede a esta página dando al botón “Admin” para un WT concreto.

LISTADO DE IOTS

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

DAR DE ALTA 1 IOT

Un dueño puede dar de alta iot rellenando el endpoint del iot, el usuario y contraseña. Estos datos son almacenados en base de datos del sac.

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

COMPARTIR 1 ACCIÓN

Desde el listado de iot se accede a la parte de "Admin" de un iot donde se ve el listado de acciones. En el listado de acciones se puede compartir 1 acción dando, de nuevo a otro botón "Admin". Se muestra el listado de amigos y se comparte dando al botón de "Share". Se muestra el path para compartir con tu amigo.

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

25. 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.

BACKEND SAC

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

Sac almacena el mínimo posible sobre el iot esto es:

- Endpoint
- Usuario
- Contraseña

26. Autenticación Delegada en Facebook

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

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

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

Lo que en Facebook se llama token en SAC es almacenado con el nombre de "fb_delegated"

27. Proceso de Login

En la siguiente ilustración [Ilustración 8 - Caso de uso durante LoginIlustración 8] 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:

XX. Camino "Raíz del proyecto"

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

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

XXI. Camino “Crear Owner”

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

XXII. Camino “Index de Owner”

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

XXIII. Camino “Index de Friend”

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

XXIV. Camino “Página de error”

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

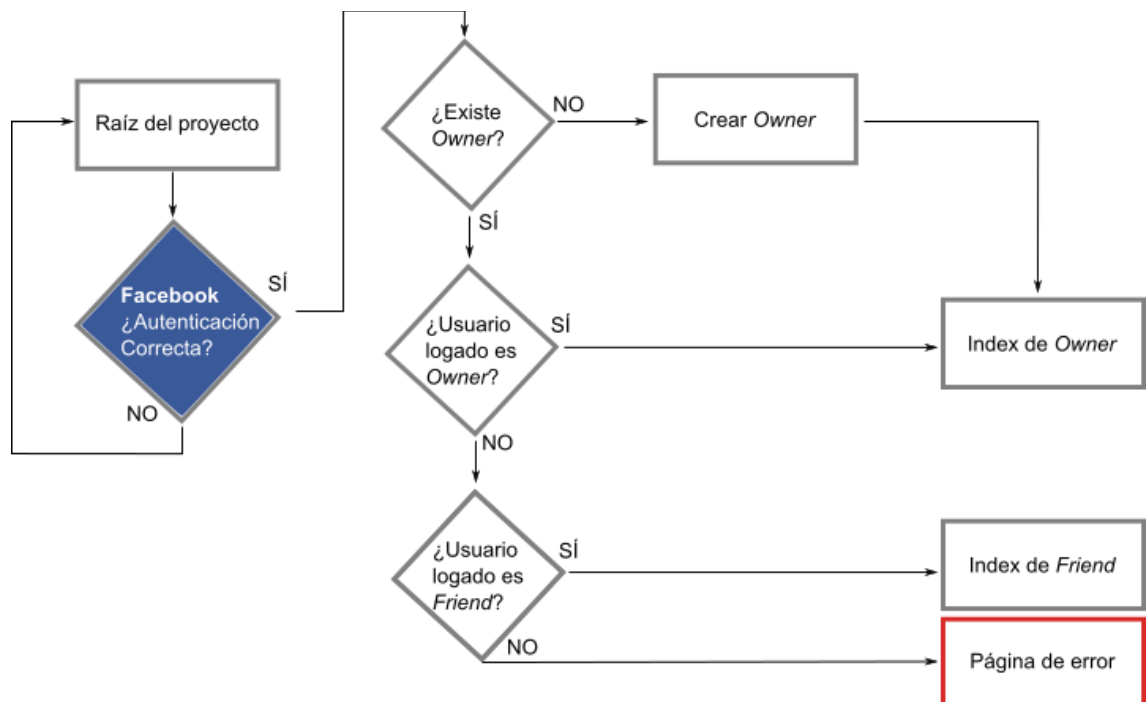


Ilustración 8 - Caso de uso durante Login

28. Crear Owner

Cuando SAC crea al Owner ocurren varias cosas. Primero se guarda en tabla owner el fb_delegated (token de Facebook) y nombre del Owner. Segundo consultamos

a Facebook el listado de *Friends* y lo guardamos en SAC guardando todos los `fb_delegated` (tokens de Facebook) de cada *Friend*

Una mejora propuesta es poder actualizar la lista de *Friends* [44]

29. Listado de *WTs*

El listado de *WTs* se obtiene de la tabla `thing` a través de la Entidad de Dominio Owner. El listado es la agrupación de información de los JSON obtenidos para cada *WT* [33].

30. Obtener información de un *WT*

Cuando SAC necesita la complementar la información de un *WT* necesita consultarla al *WT* a través del `lot_emulator`.

31. Dar de alta nuevo *WT*

32. Mostrar *Actions* de un *WT*

33. Compartir un *Action* con un friend

34. API SAC

SAC incluye una API por varias razones.

- La necesidad de traer datos actualizados de *WTs* vía `lot_emulator`.y delegar esta obtención al frontend que lo pide vía ajax.

XXV. Obtención de datos vía Ajax de `lot_emulator`

TODO explicar ThingConnected

Como la información de los *WT* está en los propios *WT* hemos decidido crear una API en SAC que conecte con `lot_emulator`

XXVI. Usar el generador de URLs propio de Symfony integrado en Twig

TODO explicar URLProvided

35. Esquema Base de datos SAC

El usuario para acceder a esta base de datos debe tener estos permisos: DDL, ALTER, DML, SELECT, INSERT, UPDATE, DELETE.

En la imagen [Ilustración 9] mostramos el esquema usado en SAC. En ella se ven las tablas y campos usados. Pensamos que el esquema es auto explicativo pero

queremos explicar los campos `fb_delegated` de tabla `owner` y tabla `friend`.y el campo `root` de tabla `thing`. También la existencia de relación `n:m` entre `owners` y `things`

XXVII. Explicacion `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 [31].

El token proporcionado por Facebook lo persistimos en base de datos y se usa para identificar al usuario logado en los campos explicado haciendo posible reconocer 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.**].

XXVIII. Explicación Tabla `thing` campo `root`

Este campo obedece al cumpliendo el requisito de almacenar la mínima cantidad de información por parte de SAC [5] y es usada 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 actualizados

XXIX. 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.

Esto está así por falta de tiempo ya que tener muchos *Owner* nos hubiera implicado más pruebas y más desarrollo.

Una ampliación del proyecto propuesto en Mejoras es esta [44]

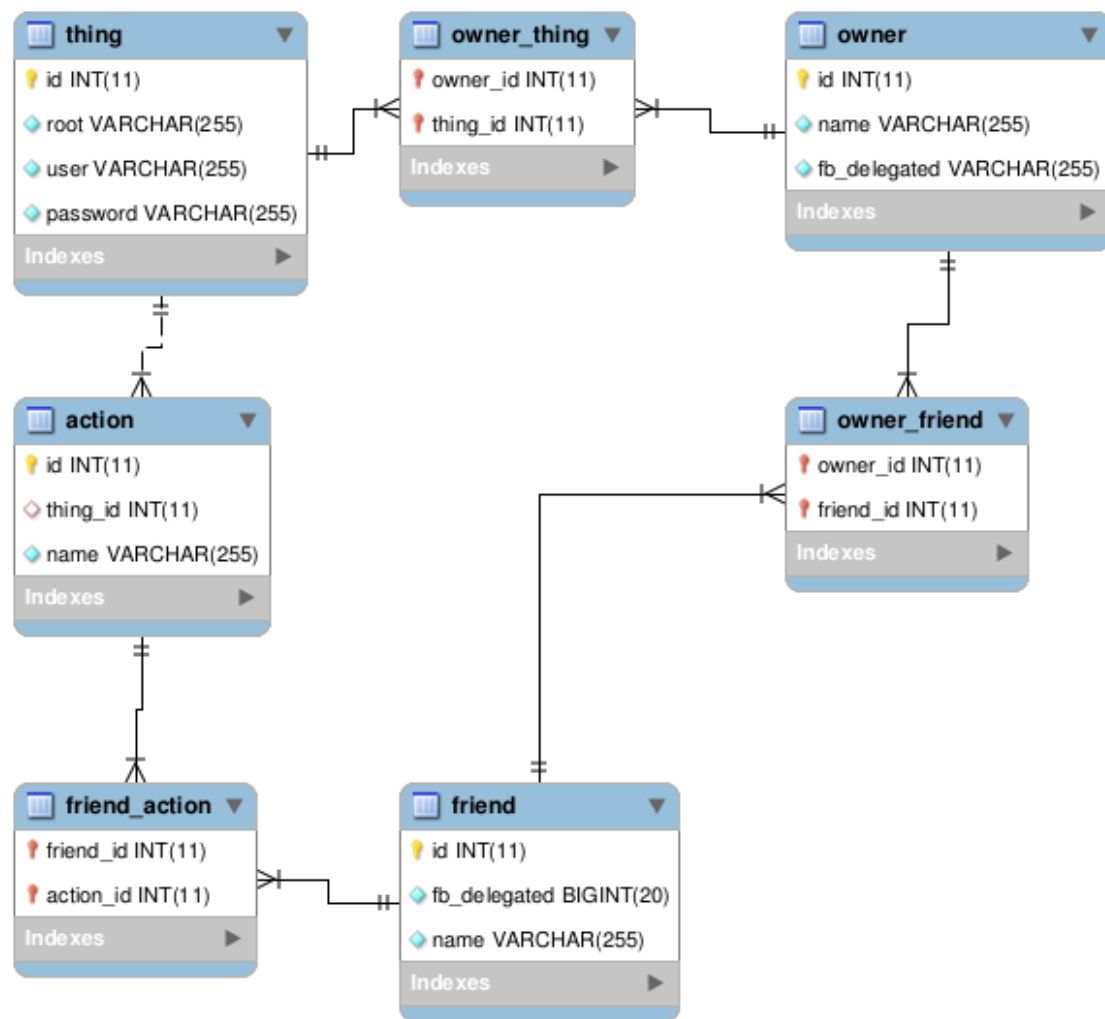


Ilustración 9 Esquema de SAC

36. Arquitectura REST SAC

37. Symfony

XXX. Comandos Symfony

XXXI. “Lo de interfaz para eventos”

38. SEGURIDAD

XXXII. HTTPs

Certificado creado con XXXX (TODO buscar)

No podrá acceder a zona Friend:

- Ninguna persona que no sea amigo
- Friends sin acciones compartidas

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

TECNOLOGÍAS USDAS

39. 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
- Javascript
- Twig
- Symfony
- Cliente HTTP

Asi como ayudas que ofrece mejorar el código:

- PSR
- Creación de servicios symfony
- Búsqueda inteligente de
 - definiciones de métodos
 - implementaciones de interfaces

40. Facebook

XXXIII. Facebook API

Autenticación delegada

Consulta de lista de amigos

XXXIV. Facebook.developers

Creación de usuarios de prueba

Login

Creación de aplicación web

41. Sistema operativo

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

XXXV. - variable de entorno

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

XXXVI. sshfs:

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

```
sudo sshfs ubuntu@35.180.227.177:/var/www/iot_emulator
/mnt/iot_emulator -o
IdentityFile=/home/${USER}/dev/sac_sandbox/docs/socialaccesscontrolle
er-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/socialaccesscontrolle
er-paris.pem -o allow_other
```

XXXVII. alias

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

```
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_schema_drop_and_create_fixtures_load_NOT_symfony='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:fixtures:load -n && cd -'

alias sac_fixtures_load_append='sac && php bin/console
doctrine:fixtures: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:fixtures:load -n && cd -'

alias sac_sandbox_fixtures_load_append='sac_sandbox && php
bin/console doctrine:fixtures: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_cr  
eate && sac_fixtures_load && cd -'
```

XXXVIII. shell-script para provisionamiento

TOD copiar Shell script

XXXIX. httpie

[LINK-INTERNO] DISeNYO-FIXTURES

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

XL. - jq

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

XLI. git:

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

42. 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

43. AWS:

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

XLII. Características de máquina desplegada en producción

TODO

44. nginx:

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

- 45. [html5](#)
- 46. [javascript](#)
- 47. [jquery](#)
- 48. [moustache](#)
- 49. [symfony 4](#)
- 50. [npm](#)
- 51. [mysql](#)
- 52. [Doctrine](#)

ANÁLISIS RESULTADOS

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

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

Se consigue simplificar la manera de compartir *WTs*

RELACIÓN CON ASIGNATURAS DEL MÁSTER

CONCLUSIONES

"Lo que acabas de explicar"

El proyecto esta hecho en el marco de IOT.

Se ha hecho *iot_emulator* en capa de accesibilidad y *sac* en capa de compartición

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

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

INSTALACIÓN

60. ---- INSTALLATION SYSTEM REQUIREMENTS

Hemos usado EC2 con conexiones *ssh* (administración del sistema), *http* (*iot_emulator*) *https* (*sac*).

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

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

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

- sudo apt-get install php-curl (actually in iot_emulator, fixtures are done in a php script via curl)

- sudo a2enmod rewrite

Development requirements

- sudo apt-get install httpie jq

61. ----- INSTALACION SAC

(sac) App scaffold:

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

```
composer install
```

Facebook, create project and get FACEBOOK_APP_ID and FACEBOOK_SECRET

(sac) Create .env.local

- fill FACEBOOK_APP_ID and FACEBOOK_SECRET

- fill DATABASE_URL

Create .env.local with FB and Mysql credentials

```
php bin/console doctrine:database:create
```

```
php bin/console doctrine:schema:create
```

62. ----- INSTALACION IOT_EMULATOR

(iot_emulator) App scaffold:

```
git clone https://github.com/danielsalgadop/iot_emulator
```

composer install

npm install

(iot_emulator)

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

php bin/console doctrine:database:create

php bin/console doctrine:schema:create

MEJORAS

XLIII. Multi-owner

XLIV. 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 no necesitan ser actualizados puedan ser almacenados en sistema de chacheo estilo Redis. Mientras que los otros sí deban ser consultados en cada momento.

XLV. 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.

XLVI. 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 *lot_emulator* ofrece los endpoints públicos de todos los *WTs* emulados en un JSON haciendo una petición GET a su raíz.

BIBLIOGRAFÍA

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

2- <https://www.w3.org/Submission/2015/SUBM-wot-model-20150824/> El de w3Consortium