

**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

# 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*.
* **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*.

# Resumen

Internet of Things (Iot) 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 estos entre personas de confianza.

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

Para abordar la compartición de los *Iots* 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 aplicacbilidad la que hemos querido implementar con el Social access controller (SAC). Donde poder compartir con otras personas informacion de internet of things (Iot) 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 Iots. SAC es un intermediario entre la red de amigos y las credenciales en os Iots.

# Abstract

# Índice General

# Introducción

## 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*.

## Problemas a compartición de *WT* a resolver

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

### *WT* multiusuario

Usar *WT* multiusuarios y condecer 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*

### 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*

## 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*

Requistios*:*

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 puedes 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

## Objetivos

El objetivo primario del Trabajo Fin de Máster es crear una aplicación que cumpla los requistios anteriormente mencionados. Y asi 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

### 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*)*.*

### Cumplimiento de requistios

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

### Capa de accesibilidad mediante Iot Emuator

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

Para acometer el proyecto hemos simplificado los modelos propuestos tanto por [LINK1] y [LINK2]. Aqui abajo explicamos las simplificaciones.

## Acople al modelo *WT* del W3Consortium

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

Nivel 0 – Un *WT* DEBE

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

Nivel 1 – *WT* DEBERÍA

|  |  |
| --- | --- |
|  | 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

|  |  |
| --- | --- |
|  | 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

## Diseño Bases de Datos

### SAC

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



### Esquema Iot\_emulator

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



## Arquitectura REST

TODO poner VerboHTTP

### Iot\_emulator

|  |  |  |
| --- | --- | --- |
| Verbo HTTP | Endpoint | Descripción |
|  | / | Index de las zonas públicas de los *WTs* |
|  | /{id} |  |
|  | /{id} |  |
|  | /create |  |
|  | /{id}/actions/{action\_name} |  |
|  | /{id}/properties/{property\_name} |  |
|  | /{id}/actions |  |
|  | /{url} |  |

### SAC

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

## Arquitectura Hexagonal

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

Cabe destacar que en capa de infrastrucutra hemos usado Comandos Symfony que usan Commandos y CommandHandlers de la aplicacion.

# 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” será *Owner* en SAC .

Como se ve en parte inferior derecha de [LINK-foto-facebook\_index\_elizabeth] Hemos creado 3 amigas:

* Susan
* Mary
* Linda de las Mareas

.

Tal como muestra [LINK-imagen-facebook-amigas]. Elisabeth tiene dos *Friends*:

* Linda a la *Owner* sí comparte *WT.*
* Mary a la que **no** comparte ningún *WT*

Estas relaciones se crean directamente en SAC

## Datos pruebas SAC

Hemos populado 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

## WT de pruebas Iot\_emulator:

- 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"}]}}

# Tests

## iot\_emulator:

Hemos hecho 2 tipos de tests.

* Basados en phpunit
* Hecho en PHP

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

### no usan phpunit

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

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

php tests/notPHPUnit/isIntegrityValidOnCreate.php

### usan phpunit

TODO configurar para ejecutar phpunit

## SAC

# Arquitectura Iot\_emulator

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

Hemos usando symfony4 para crear un API con arquitectura REST y estructura de datos JSON.

Es una API REST que emula la capa de conexión entre los *WT* emulados

## Estructura básica *WT*

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

### Respuesta JSON

{

"id": 1,

"name": "thing\_name1",

"brand": "thing\_brand1",

"links": {

"actions": {

"link": "\/actions",

"resources": {

"action\_name1": {

"values": "property\_value1"

}

}

}

}

}

### Acoplamiento 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*.

### Zona pública y Zona privada

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

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

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

### *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. Asi thing\_1 sera /1 y será el id con pk=1 en tabla thing.

## Arquitectura REST Iot\_emulator

|  |  |  |
| --- | --- | --- |
| Verbo HTTP | Endpoint | Descripción |
|  | / | Index de las zonas públicas de los *WTs* |
|  | /{id} |  |
|  | /{id} |  |
|  | /create |  |
|  | /{id}/actions/{action\_name} |  |
|  | /{id}/properties/{property\_name} |  |
|  | /{id}/actions |  |
|  | /{url} |  |

### ¿COMO ACCEDER A LISTADO IOTS?

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

## Arquitectura hexagonal de Iot\_emulator

### DOMINIO

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

### APLICACIÓN

#### Estructura de Command y 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

### Infrastructura

#### 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

#### Serializadores

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

src/Infrastructura/Thing/Command/Serializer/ThingActions

src/Infrastructura/Thing/Command/Serializer/ThingWithCredetials

src/Infrastructura/Thing/Command/Serializer/ThingWithoutCredentials

# Arquitectura SAC

# Bibliografía

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

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