



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*.
- **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 información 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 los lots.

ABSTRACT

CONTENIDO

Glosario	6
Resumen.....	7
Abstract	8
Introducción	12
Motivación.....	12
Problemas de compartición de <i>WT</i> a resolver.....	12
Requisitos	12
Objetivos	13
Diseño General.....	15
Acople al modelo <i>WT</i> del W3Consortium	15
Diseño de Software. Arquitectura Hexagonal.....	16
Comandos Symfony	17
Endpoints.....	18
Lanzamiento Peticiones HTTP.....	19
Datos de prueba (Fixtures).....	22
Usuarios de prueba en Facebook.....	22
WT de pruebas lot_emulator:.....	23
Datos pruebas SAC	23
Sadfasd	24
Tests.....	25
lot_emulator	25
SAC.....	25
Backend lot_emulator.....	26
Estructura básica <i>WT</i>	26
Esquema Base de datos lot_emulator	27
Arquitectura REST lot_emulator.....	28

Arquitectura hexagonal de lot_emulator	29
Seguridad lot_emulator.....	30
Frontend SAC.....	32
Raíz del proyecto.....	32
Mapa web para <i>Owner</i>	33
Admin de un Thing	34
Ver listado de <i>Friends</i>	35
Resultado de compartición.....	35
Mapa web para <i>Friend</i>	36
BACKEND sac	37
Autenticación Delegada en Facebook.....	37
Proceso de Login.....	37
Crear Owner	39
Acciones sobre los WTs	39
API SAC.....	39
Esquema Base de datos SAC	44
Arquitectura REST SAC	46
Arquitectura Hexagonal	46
Symfony	48
SEGURIDAD	50
Tecnologías usadas.....	51
PHPStorm:	51
Facebook.....	51
Sistema operativo.....	51
github:	53
AWS:.....	53
nginx:.....	53
html5.....	53

javascript.....	53
jquery	54
Jquery-UI.....	54
moustache	54
symfony 4.....	54
npm.....	54
mysql.....	54
Doctrine.....	54
<i>Análisis resultados</i>	55
<i>Relación con Asignaturas del máster</i>	55
Frameworks	56
Emprendiduría	56
Entorno Web	56
Seguridad.....	56
Frontend.....	56
PHP	56
Bases de Datos.....	56
<i>Conclusiones.....</i>	56
<i>Instalación</i>	56
--- INSTALLATION SYSTEM REQUIREMENTS	56
----- INSTALACION SAC	57
----- INSTALACION IOT_EMULATOR	58
<i>Mejoras</i>	59
<i>Bibliografía</i>	59

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

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 DE COMPARTICIÓN DE *WT* A RESOLVER

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

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*

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*

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

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

COMPARTICIÓN MEDIANTE SOCIAL ACCESS CONTROLLER (SAC)

Siguiendo la idea de un SAC propuesta por Dominique Guinard *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 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

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

ACOPLE AL MODELO *WT* DEL W3CONSORTIUM

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

Nosotros hemos adaptado el model de 2015.

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

PATRÓN DE INTEGRACIÓN DE LOS *WT* EN W3CONSORTIUM

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

REQUERIMIENTOS PARA UN *WT* EN W3CONSORTIUM

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

Para las tabla

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

Nivel 0 – Un *WT* DEBE

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>lot_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

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

DOMINIO

SSSS

APLICACIÓN

SSS

INFRAESTRUCTURA

SSSS

Expicar arquitectura hexagonal

Controladores

Procesan la Request, llaman al CommandHandler (capa de Aplicación) y construyen una Response.

Respositorios

Las implementaciones de los repositorios responden a sus correspondientes contratos de la capa de Dominio.

Serializadores

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

Comandos Symfony

En capa de Infraestructura hemos hecho uso de Comandos Symfony que usan Commandos y CommandHandlers de la capa de aplicacion.

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
```

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 Docntine. 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.

ENDPOINTS

Se puede encontrar una explicación detallada en cada secciones de arquitectura de lot_emulator (Arquitectura REST lot_emulator0 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

IOT_EMULATOR

Se puede consultar el detalle en esta sección (Arquitectura REST lot_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
	/
	/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}

LANZAMIENTO PETICIONES HTTP

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

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

docs/request

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 obeitos PHP. Y lo hacemos con arquitectura hexagonal (TODO ejemplo arquitectura hexagonal: Orm doctrine)

DIFERENCIAS ENTRE IOT_EMULATOR Y SAC

Mientras que en lot_emulator hemos usado los @annotations como manera de definir las relaciones entre entidades. En SAC hemos usado ficheros .yaml.

Uso de @annotations en lot_emulator

Ejemplo de @annotation sacado de src/Domain/Entity/Property.php de Property del lot_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 lot. Pero al estar metido en capa Infraestructura 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
- lot_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.

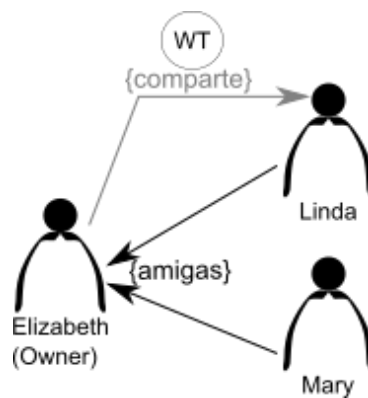


Ilustración 2 Compartición entre amigas de Owner

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

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

```

{"name":"thing_name1","brand":"thing_brand1","links":{"actions":["action_name1"],"properties":[{"action_name1":"property_value1"}]}}
{"name":"thing_name2","brand":"thing_brand2","links":{"actions":["action_name1","action_name2"],"properties":[{"action_name1":"property_value1"}, {"action_name2":"property_value2"}]}}
{"name":"thing_name3","brand":"thing_brand3","links":{"actions":["action_name1","action_name2","action_name3"],"properties":[{"action_name1":"property_value1"}, {"action_name2":"property_value2"}, {"action_name3":"property_value3"}]}}
  
```

DATOS PRUEBAS SAC

Hemos 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

SADFASD

TESTS

IOT_EMULATOR

Hemos hecho 2 tipos de tests.

- Basados en phpunit
- Hecho en PHP

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

NO USAN PHPUNIT

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

USAN PHPUNIT

TODO configurar para ejecutar phpunit

SAC

BACKEND IOT_EMULATOR

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

ESTRUCTURA BÁSICA *WT*

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

ZONA PÚBLICA Y ZONA PRIVADA

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

RESPUESTA JSON

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

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

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

```
{
  "id": 1,
  "name": "thing_name1",
  "brand": "thing_brand1",
  "links": {
    "actions": {
      "link": "\actions",
      "resources": {
        "action_name1": {
          "values": "property_value1"
        }
      }
    }
  }
}
```

RELACIÓN ENTRE ACTIONS Y PROPERTIES

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

url/actions/action_name1

Devolvería

property_value1

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

WT ENDPOINTS

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

ESQUEMA BASE DE DATOS IOT_EMULATOR

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

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

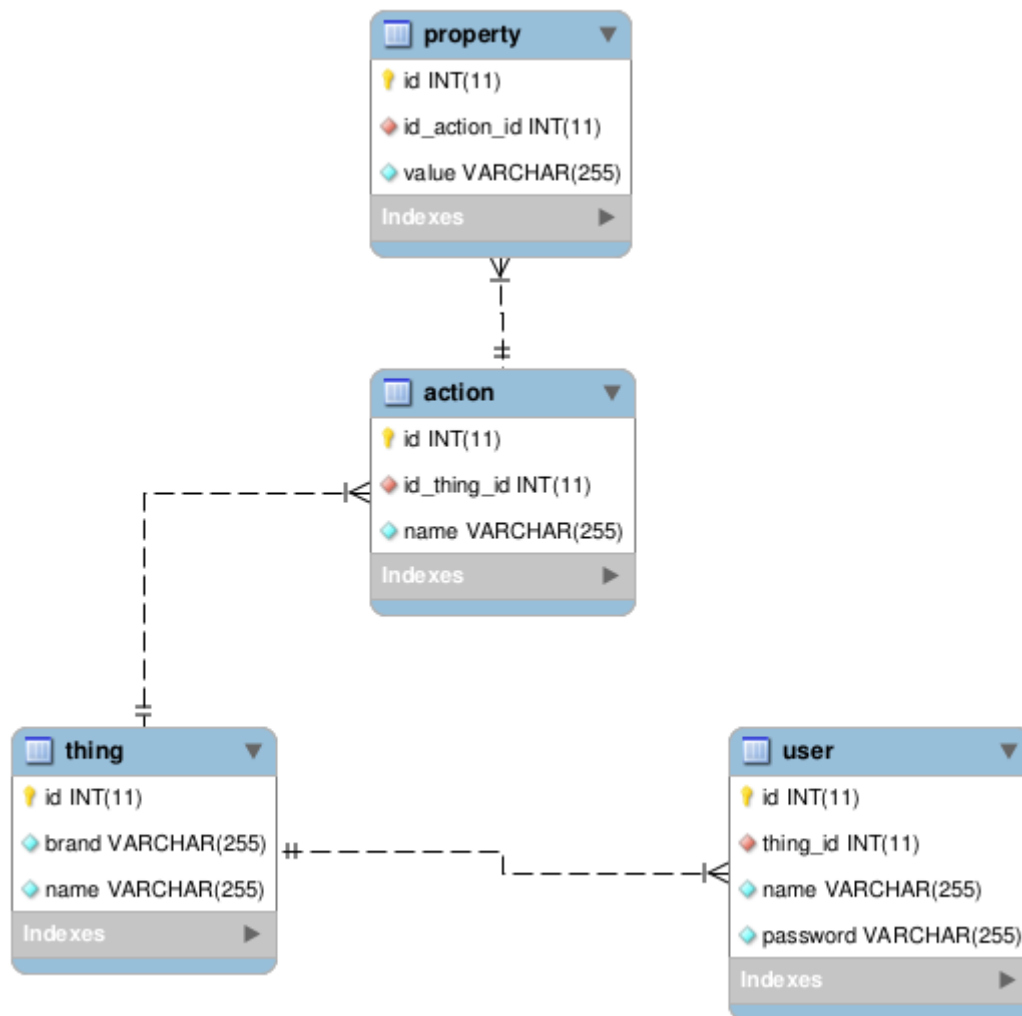


Ilustración 3. Esquema *lot_emulator*.

ARQUITECTURA REST IOT_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
GET	/	Index de las zonas públicas de los <i>WTs</i>
	/id (Sin credenciales)	Acceder a zona Pública de <i>WT</i>
	/id (Con credenciales)	Acceder a zona Privada de <i>WT</i>
	/create	Crear un <i>WT</i>
	/id/actions/{action_name}	Ejecutar un <i>Action</i>
	/id/properties/{property_name}	Acceder a una <i>Property</i>

<code>/id/actions</code>	Listado de <i>Actions</i>
<code>/url</code>	¿?

ENDPOINT - /

¿COMO ACCEDER A LISTADO IOTS?

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

ARQUITECTURA HEXAGONAL DE IOT_EMULATOR

DOMINIO

Explicación de esta capa en (17).

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

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

APLIACIÓN

Explicación de esta capa en (17).

Estructura de Command y CommandHandlers

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

```
// Commands
src/Application/Command/Thing/CreateThingCommand.php
src/Application/Command/Thing/ExecuteActionCommand.php
src/Application/Command/Thing/SearchThingByIdCommand.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 y solo contiene datos y ninguna lógica.

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

INFRAESTRUCTURA

Explicación de esta capa en (17).

Controladores

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

Repositorios

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

Comandos de Symfony

Ejecución del Comando de 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

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
```

SEGURIDAD IOT_EMULATOR

Las peticiones con credenciales incorrectas

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

Las peticiones sin credenciales

- No pueden acceder a la Zona Privada de un *WT*
- No pueden 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 [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 para *Friend*” [32] 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 los listado de *WTs* o listado de *Friends*.

RAÍZ DEL PROYECTO

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

El detalle de como gestiona SAC la Autenticación delegada se puede ver en sección backend [37] y la información sobre los datos guardados por SAC está en [44]

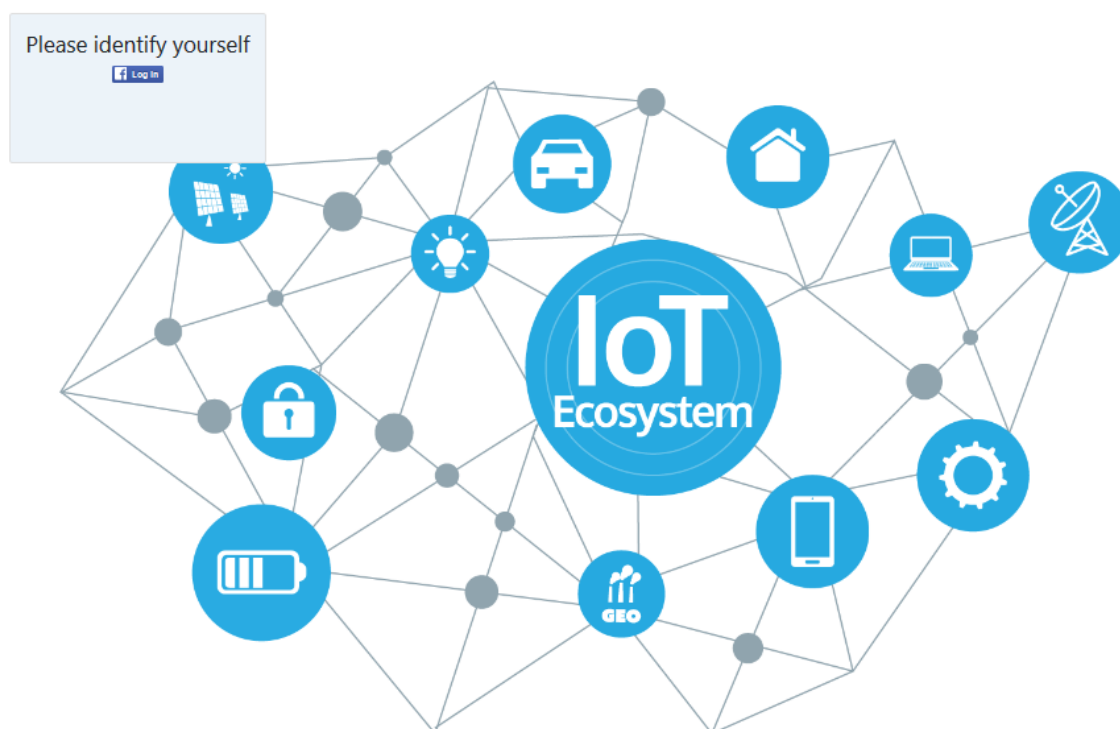


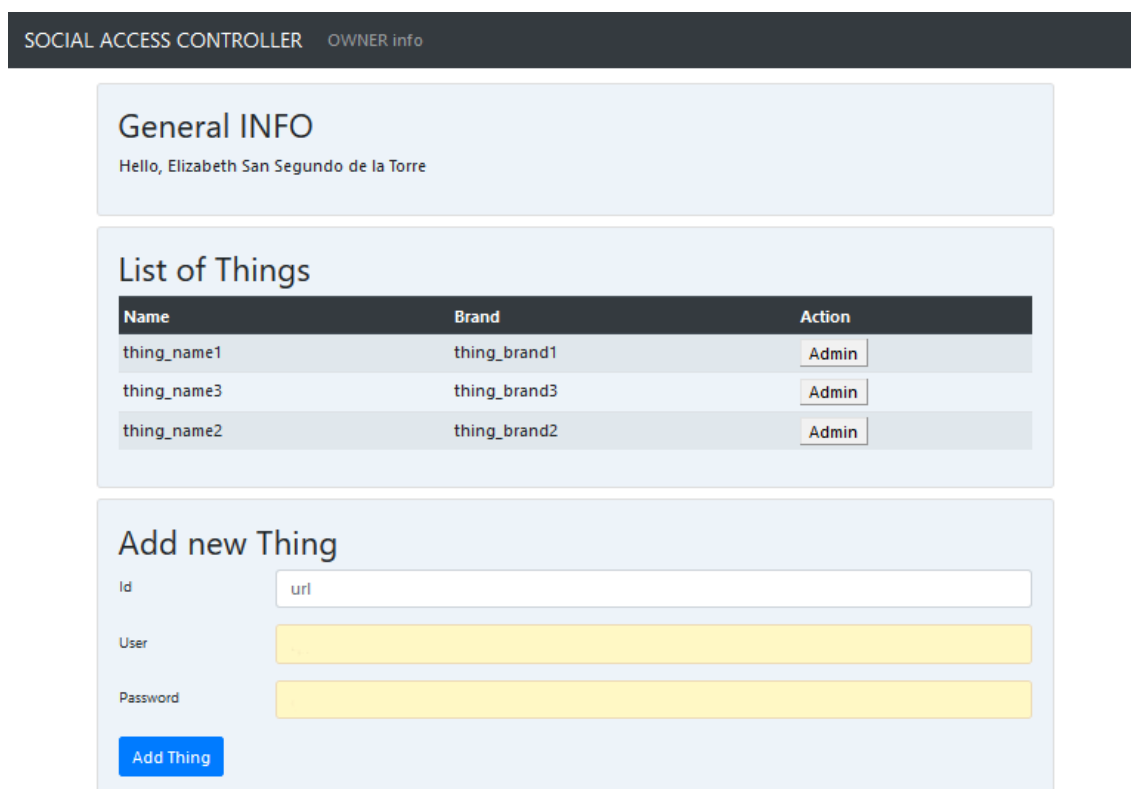
Ilustración 4. Captura de pantalla de la Raíz del proyecto.

MAPA WEB PARA OWNER

INDEX DEL OWNER

Endpoint: <https://socialaccesscontroller.tk/owner>

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



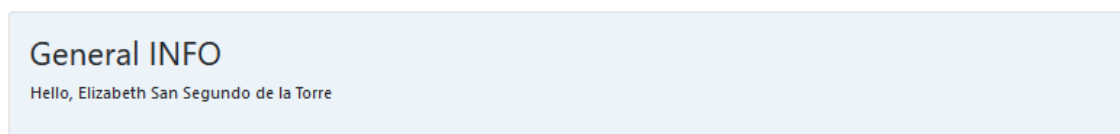
The screenshot shows the 'OWNER info' page of the 'SOCIAL ACCESS CONTROLLER'. It features three main sections: 'General INFO' with a greeting, 'List of Things' with a table of three items, and 'Add new Thing' with input fields for Id, User, and Password, and an 'Add Thing' button.

Name	Brand	Action
thing_name1	thing_brand1	<button>Admin</button>
thing_name3	thing_brand3	<button>Admin</button>
thing_name2	thing_brand2	<button>Admin</button>

Ilustración 5. Visión general del Index del Owner.

Información general del Owner

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



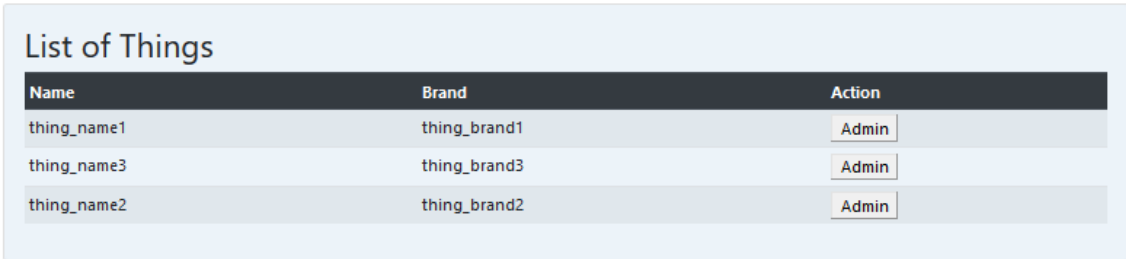
This screenshot shows the 'General INFO' section, displaying the text 'Hello, Elizabeth San Segundo de la Torre'.

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

Listado de WTs dados de alta en SAC

La ilustración[] es una captura de pantalla donde se ve el listado de todos los WT dados de alta por Owner en SAC. En este ejemplo existen tres WT dados de alta cuyos nombres son *thing_name1*, *thing_name2* y *thing_name3*. Corresponden con los datos de prueba de lot_emulator usados durante el desarrollo y explicado en este apartado [23].

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



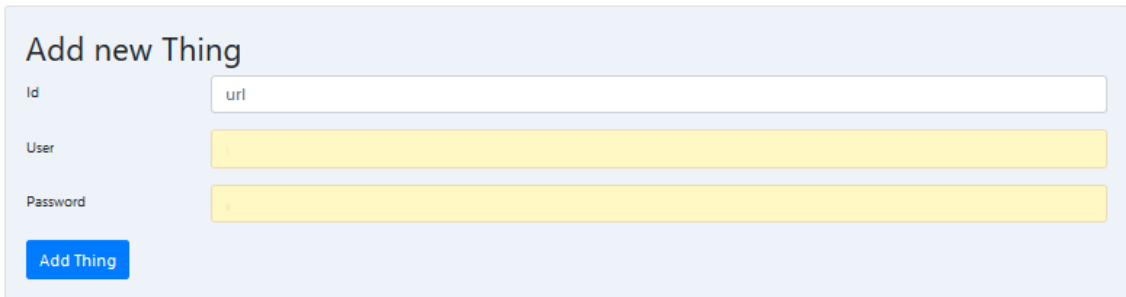
Name	Brand	Action
thing_name1	thing_brand1	Admin
thing_name3	thing_brand3	Admin
thing_name2	thing_brand2	Admin

Ilustración 7. 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_emulador* y construya una lista con los endpoints descubiertos. Haciendo más cómodo este proceso de dar de alta.



Add new Thing

Id

url

User

Password

Add Thing

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

ADMIN DE UN *WT*

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

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

SOCIAL ACCESS CONTROLLER OWNER info		
thing_name3 thing_brand3		
Action Name	Property Value	Admin
action_name1	property_value1	Admin
action_name2	property_value2	Admin
action_name3	property_value3	Admin

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)”.

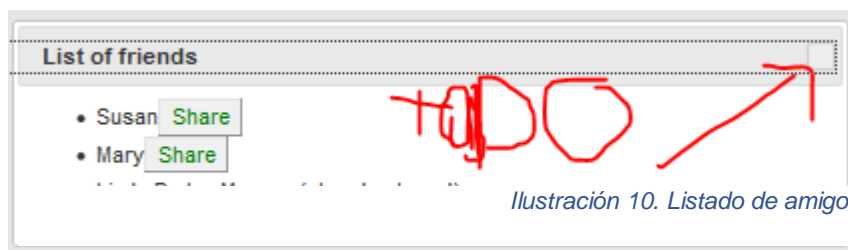
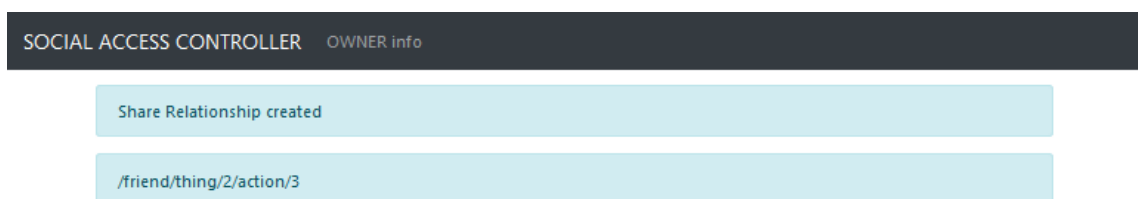


Ilustración 10. Listado de amigos.

RESULTADO DE COMPARTICIÓN

En la (ilustración) se 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*.



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

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

COMPARTIR 1 ACCIÓN

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.

MAPA WEB PARA *FRIEND*

El punto de entrada del amigo hay:

- información general sobre el amigo
- listado de acciones compartidas por owner

VER UNA PROPIEDAD COMPARTIDA

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

PÁGINA DE ERROR Y DE ÉXITO

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

PÁGINA DE ÉXITO

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

PÁGINA DE ERROR

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

BACKEND SAC

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

Sac almacena el mínimo posible sobre el iot esto es:

- Endpoint
- Usuario
- Contraseña

AUTENTICACIÓN DELEGADA EN FACEBOOK

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

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

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

PROCESO DE LOGIN

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

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

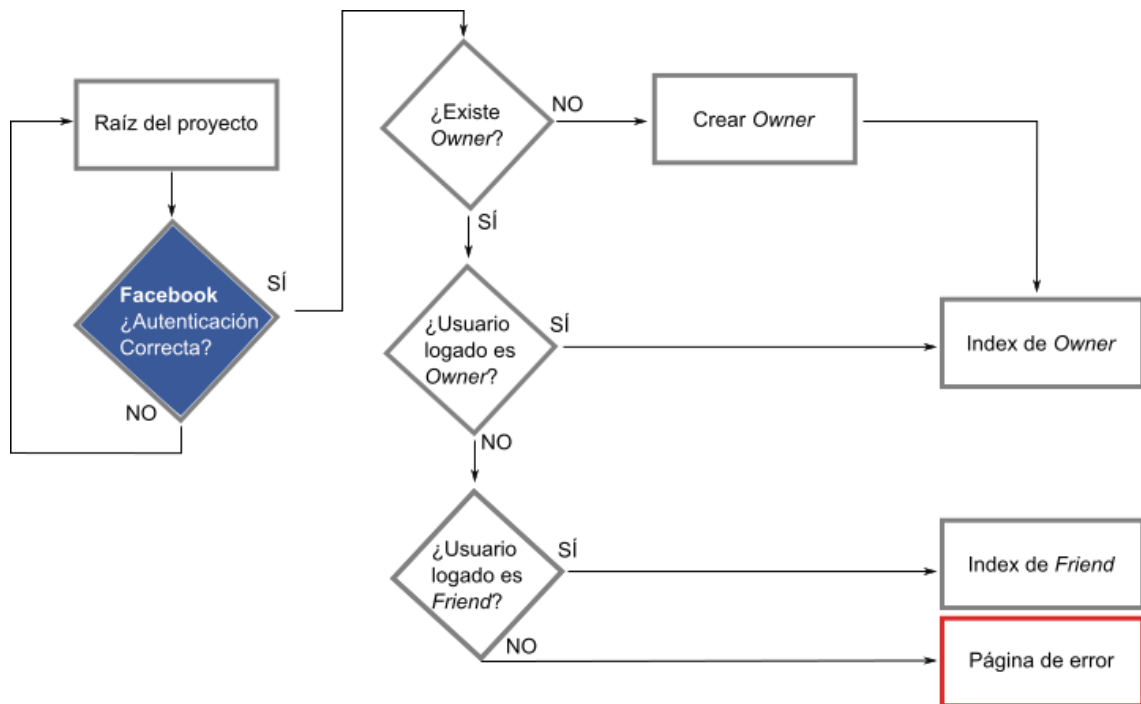


Ilustración 11. Caso de uso durante Login.

CAMINO “RAÍZ DEL PROYECTO”

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

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

CAMINO “CREAR OWNER”

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

CAMINO “INDEX DE OWNER”

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

CAMINO “INDEX DE FRIEND”

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

CAMINO “PÁGINA DE ERROR”

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

CREAR OWNER

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* [59]

ACCIONES SOBRE LOS WTs

DAR DE ALTA NUEVO WT

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

OBTENER INFORMACIÓN DE UN WT

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

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

MOSTRAR ACTIONS DE UN WT

ssss

COMPARTIR UN ACTION CON UN FRIEND

Crear una relación entre *friend* y *action*

CREAR URL DE COMPARTICIÓN

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

API SAC

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

OBTENCIÓN DE DATOS VÍA AJAX DE IOT_EMULATOR

En la (Ilustración 12) vemos los pasos.

1. Request inicial
2. Response inicial.
3. Frontend llama a SAC API vía Ajax
4. Request al lot_emulator
5. Response del lot_emulator
6. Respuesta Ajax para rellenar el listado

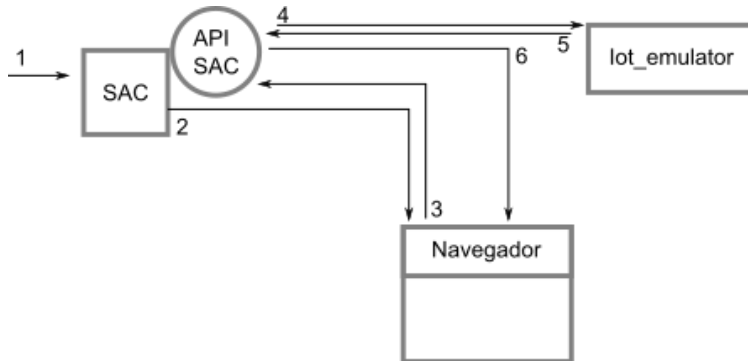


Ilustración 12 Obtención de datos via Ajax del lot_emulator

USAR EL SISTEMA DE ENRUTAMIENTO DE SYMFONY DESDE FRONTEND

Symfony ofrece una manera robusta de definir los paths de enrutamiento. La ruta concreta se definen en ficheros .yaml y se usan alias para ser usados. De esta manera el enruta se puede modificar sencillamente en un único punto. Poniendo por ejemplo una ruta con un parámetro definida con alias "nombre_ruta" y ruta con un parámetro tal que "/ruta/{key_param}". Dese el backend en un AbstractController en el backend se resuelve con método ->generateUrl() asi

```
// dentro de AbstractController
$url = $this->generateUrl('nombre_alias', ['key_param' => 'value_param'])
print $url // /ruta/value_param;
```

En el frontend Twig ofrece se resuelva con función *path()* asi

```
<!-- dentro de un template -->
<a href = "{% path('nombre_alias',{ 'key_param':value_param }) %}">Link</a>
<-- <a href="/ruta/value_param">Link</a>
```

Problema encontrado en el uso del sistema de enrutado en Twig

Nos hemos visto en la incapacidad de poder usar la función específica en Twig *path()*. Ya que cuando Twig resuelve la función *path()* necesita el valor concreto el parámetro. En nuestra arquitectura no disponemos de ese dato ellos en ese momento, ya que se traen a posteriori vía Ajax.

Solución: crear un Generador de URL en API SAC

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 endpoint 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 lot_emulator) pero sin los pasos 4 y 5 ya que la

definición de la ruta está en las rutas explicadas en (pág 48) que forman parte de SAC por lo que no hace falta llamar al `lot_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 tanto la obtención de datos vía `lot_emulator()` como la generación de URLs ().La (Ilustración 13) es una captura de pantalla tras la carga completa de la página.

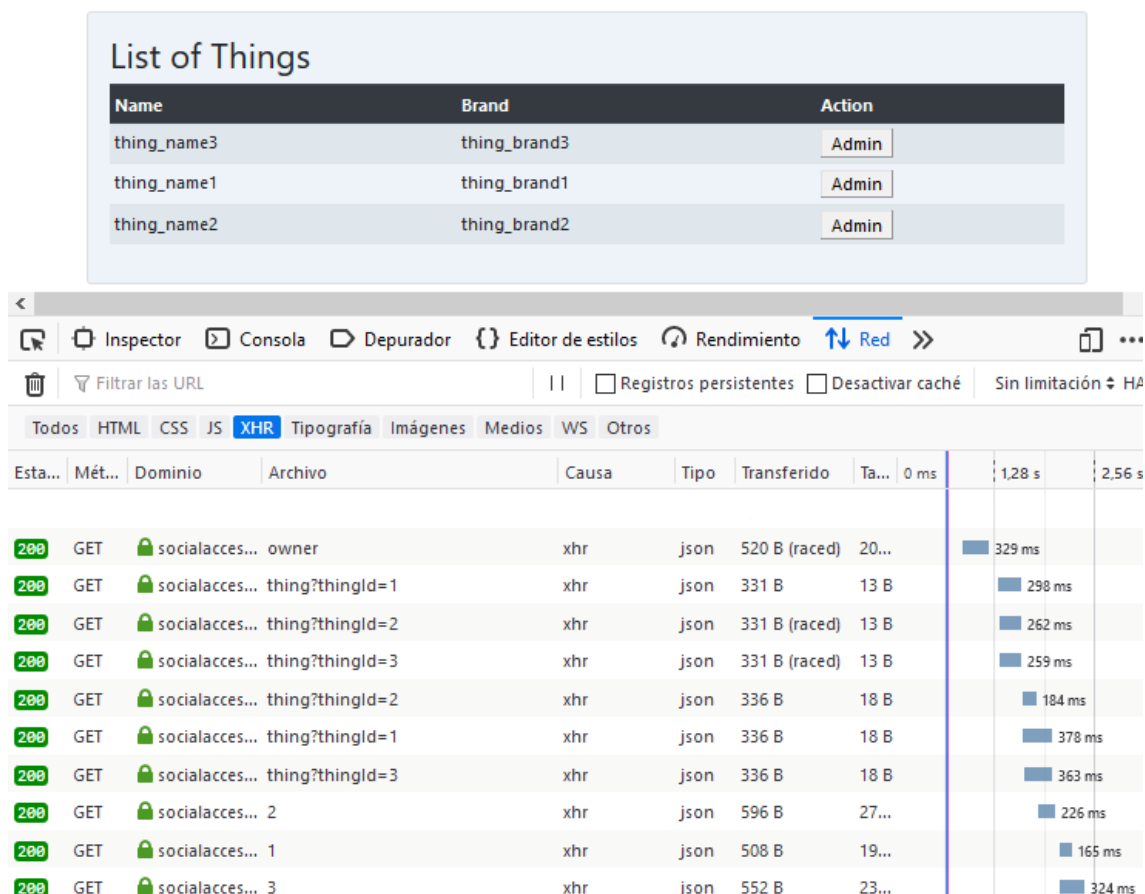


Ilustración 13. Ejemplo de uso de API SAC al generar el Listado de WT.

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

El requets inicial

Petición GET a <https://socialaccesscontroller.tk>.

Response inicial

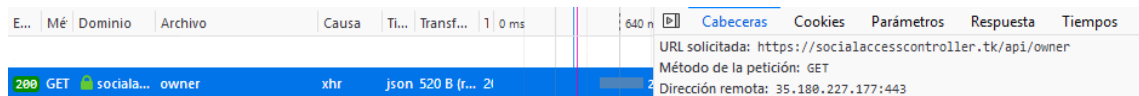
En Symfony en `App\Infrastructure\Controllers\OwnerController` devuelve un esqueleto de la página pero sin apenas información. Inmediatamente empieza a pedir información.

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

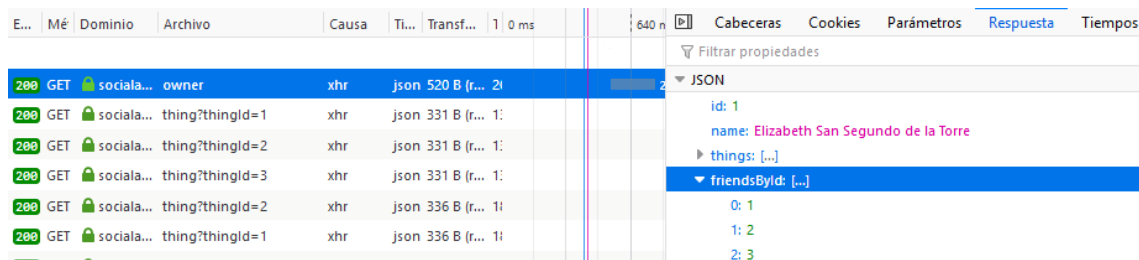
La (ilustracionX) es una captura de pantalla donde podemos ver la llamada a el endpoint api/owner de API SAC.



E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		640 n	Cabeceras	Cookies	Parámetros	Respuesta	Tiempos
200	GET	sociale...	owner	xhr	json	520 B (r...	2l							URL solicitada: https://socialaccesscontroller.tk/api/owner Método de la petición: GET Dirección remota: 35.180.227.177:443	

Ilustración 14XXX

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.

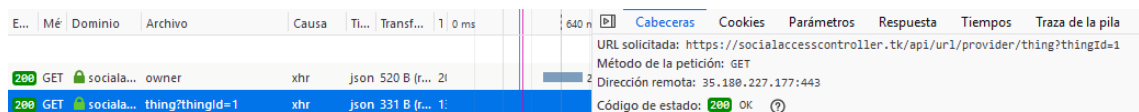


E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		640 n	Cabeceras	Cookies	Parámetros	Respuesta	Tiempos
200	GET	sociale...	owner	xhr	json	520 B (r...	2l							JSON	
200	GET	sociale...	thing?thingId=1	xhr	json	331 B (r...	1:							id: 1 name: Elizabeth San Segundo de la Torre things: [...]	
200	GET	sociale...	thing?thingId=2	xhr	json	331 B (r...	1:							friendsByid: [...]	
200	GET	sociale...	thing?thingId=3	xhr	json	331 B (r...	1:							0: 1 1: 2 2: 3	
200	GET	sociale...	thing?thingId=2	xhr	json	336 B (r...	1l								
200	GET	sociale...	thing?thingId=1	xhr	json	336 B (r...	1l								
200	GET	sociale...	thing?thingId=2	xhr	json	336 B (r...	1l								

Ilustración 15xxx

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

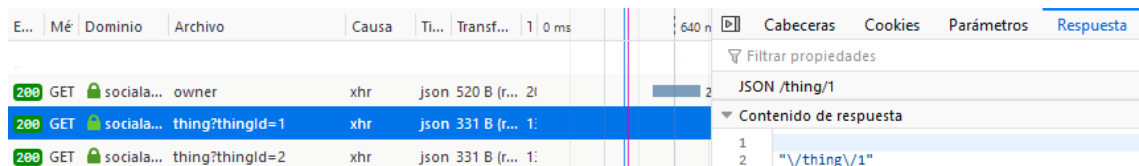
La (ilustracionX) es una captura de pantalla donde podemos ver la request 'api/url/provider/thing' para obtener una URL.



E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		640 n	Cabeceras	Cookies	Parámetros	Respuesta	Tiempos	Traza de la pila
200	GET	sociale...	owner	xhr	json	520 B (r...	2l							URL solicitada: https://socialaccesscontroller.tk/api/url/provider/thing?thingId=1 Método de la petición: GET Dirección remota: 35.180.227.177:443		
200	GET	sociale...	thing?thingId=1	xhr	json	331 B (r...	1:							Código de estado: 200 OK		

Ilustración 16xx

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



E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		640 n	Cabeceras	Cookies	Parámetros	Respuesta	Tiempos
200	GET	sociale...	owner	xhr	json	520 B (r...	2l							JSON /thing/1	
200	GET	sociale...	thing?thingId=1	xhr	json	331 B (r...	1:							Contenido de respuesta	
200	GET	sociale...	thing?thingId=2	xhr	json	331 B (r...	1:							1 2 "\/thing\/1"	

Ilustración 17. Xxxx.

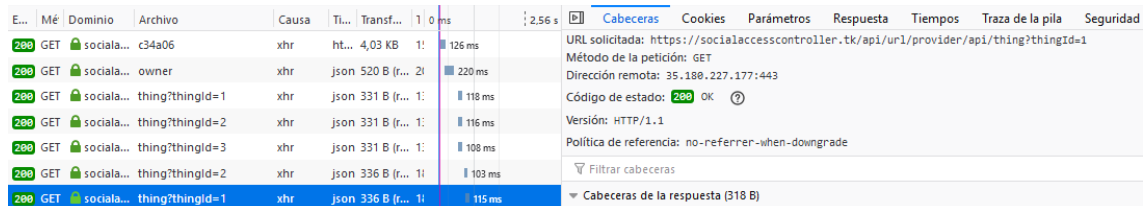
Esta información será usada como action del botón "Admin". Como puede verse esta ruta devuelta en la response es la que está definida en fichero de enrutado src/Infraestructure/Resources/config/routes/thing.yaml..De esta manera hemos conseguido usar sistema de enrutado de Symfony desde el frontend que 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 obtener una URL. Como funcionan igual y son muy parecidos no vamos a pararnos a explicar con mismo nivel de detalle.

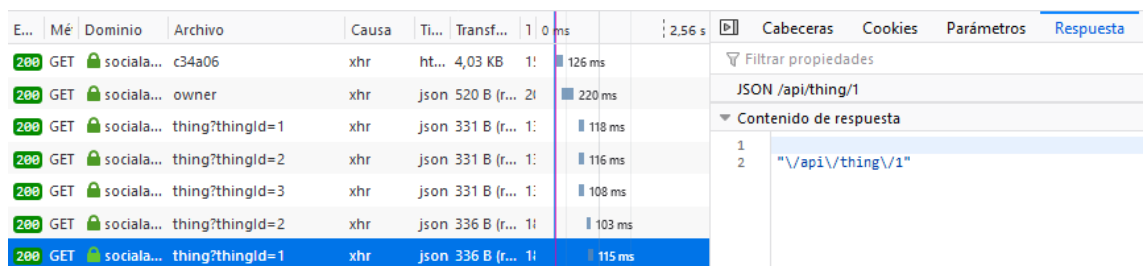
La (ilustracionX) es una captura de pantalla donde podemos ver la request “api/url/provider/api/thing” para obtener una URL.



E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		2,56 s		Cabeceras	Cookies	Parámetros	Respuesta	Tiempos	Traza de la pila	Seguridad
200	GET	sociala...	c34a06	xhr	ht...	4,03 KB	1!	126 ms				URL solicitada: https://socialaccesscontroller.tk/api/url/provider/api/thing?thingId=1			Método de la petición: GET			
200	GET	sociala...	owner	xhr	json	520 B	(r...	2!	220 ms			Dirección remota: 35.180.227.177:443			Código de estado: 200 OK			
200	GET	sociala...	thing?thingId=1	xhr	json	331 B	(r...	1!	118 ms			Versión: HTTP/1.1			Política de referencia: no-referrer-when-downgrade			
200	GET	sociala...	thing?thingId=2	xhr	json	331 B	(r...	1!	116 ms									
200	GET	sociala...	thing?thingId=3	xhr	json	331 B	(r...	1!	108 ms									
200	GET	sociala...	thing?thingId=2	xhr	json	336 B	(r...	1!	103 ms									
200	GET	sociala...	thing?thingId=1	xhr	json	336 B	(r...	1!	115 ms									

Ilustración 18xxxx

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

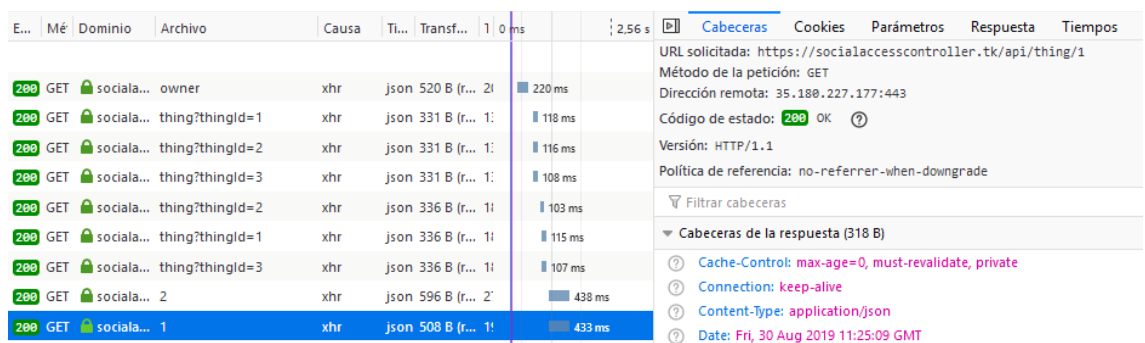


E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		2,56 s		Cabeceras	Cookies	Parámetros	Respuesta	Tiempos	Traza de la pila	Seguridad
200	GET	sociala...	c34a06	xhr	ht...	4,03 KB	1!	126 ms										
200	GET	sociala...	owner	xhr	json	520 B	(r...	2!	220 ms									
200	GET	sociala...	thing?thingId=1	xhr	json	331 B	(r...	1!	118 ms									
200	GET	sociala...	thing?thingId=2	xhr	json	331 B	(r...	1!	116 ms									
200	GET	sociala...	thing?thingId=3	xhr	json	331 B	(r...	1!	108 ms									
200	GET	sociala...	thing?thingId=2	xhr	json	336 B	(r...	1!	103 ms									
200	GET	sociala...	thing?thingId=1	xhr	json	336 B	(r...	1!	115 ms									

Ilustración 19xxxx

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

La (ilustracionX) 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 obtener URL del SAC API). Este request es quien desencadena la llamada al lot_emulator.



E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1	0 ms		2,56 s		Cabeceras	Cookies	Parámetros	Respuesta	Tiempos	Traza de la pila	Seguridad
200	GET	sociala...	owner	xhr	json	520 B	(r...	2!	220 ms			URL solicitada: https://socialaccesscontroller.tk/api/thing/1			Método de la petición: GET			
200	GET	sociala...	thing?thingId=1	xhr	json	331 B	(r...	1!	118 ms			Dirección remota: 35.180.227.177:443			Código de estado: 200 OK			
200	GET	sociala...	thing?thingId=2	xhr	json	331 B	(r...	1!	116 ms			Versión: HTTP/1.1			Política de referencia: no-referrer-when-downgrade			
200	GET	sociala...	thing?thingId=3	xhr	json	331 B	(r...	1!	108 ms									
200	GET	sociala...	thing?thingId=2	xhr	json	336 B	(r...	1!	103 ms									
200	GET	sociala...	thing?thingId=1	xhr	json	336 B	(r...	1!	115 ms									
200	GET	sociala...	thing?thingId=3	xhr	json	336 B	(r...	1!	107 ms									
200	GET	sociala...	2	xhr	json	596 B	(r...	2'	438 ms									
200	GET	sociala...	1	xhr	json	508 B	(r...	1!	433 ms									

Ilustración 20xxxx

Request al lot_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" 200 162
"" ""
```

Response del lot_emulator

Lo que aquí ocurre recae totalmente en el lot_emulator y en concreto en endpoint 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 lot_emulator la serializamos en App\Infraestructura\Controllers\Api\ThingApiController 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.

E...	Mé	Dominio	Archivo	Causa	Ti...	Transf...	1 0 ms	2,56 s		Cabeceras	Cookies	Parámetros	Respuesta
200	GET	sociale...	owner	xhr	json	520 B (r...	21	220 ms					<div> Filtrar propiedades </div> <div> JSON </div> <div> <pre> status: true message: null data: {...} id: 1 name: thing_name1 brand: thing_brand1 links: {...} actions: {...} link: /actions resources: {...} action_name1: {...} values: property_value1 </pre> </div>
200	GET	sociale...	thing?thingId=1	xhr	json	331 B (r...	1:	118 ms					
200	GET	sociale...	thing?thingId=2	xhr	json	331 B (r...	1:	116 ms					
200	GET	sociale...	thing?thingId=3	xhr	json	331 B (r...	1:	108 ms					
200	GET	sociale...	thing?thingId=2	xhr	json	336 B (r...	1:	103 ms					
200	GET	sociale...	thing?thingId=1	xhr	json	336 B (r...	1:	115 ms					
200	GET	sociale...	thing?thingId=3	xhr	json	336 B (r...	1:	107 ms					
200	GET	sociale...	2	xhr	json	596 B (r...	2:	438 ms					
200	GET	sociale...	1	xhr	json	508 B (r...	1:	433 ms					
200	GET	sociale...	3	xhr	json	552 B (r...	2:	431 ms					

Ilustración 21XXXX

ESQUEMA BASE DE DATOS SAC

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

En la imagen [Ilustración 22] mostramos el esquema usado en SAC. En ella se ven las tablas y campos usados. Pensamos que el esquema es auto explicativos 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

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 [37].

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.].

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

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).

CAMPO NAME EN TABLA ACTION

Se rellena en la creación de un *WT* y se usa para identificar cual *Action* se esta compartiendo.

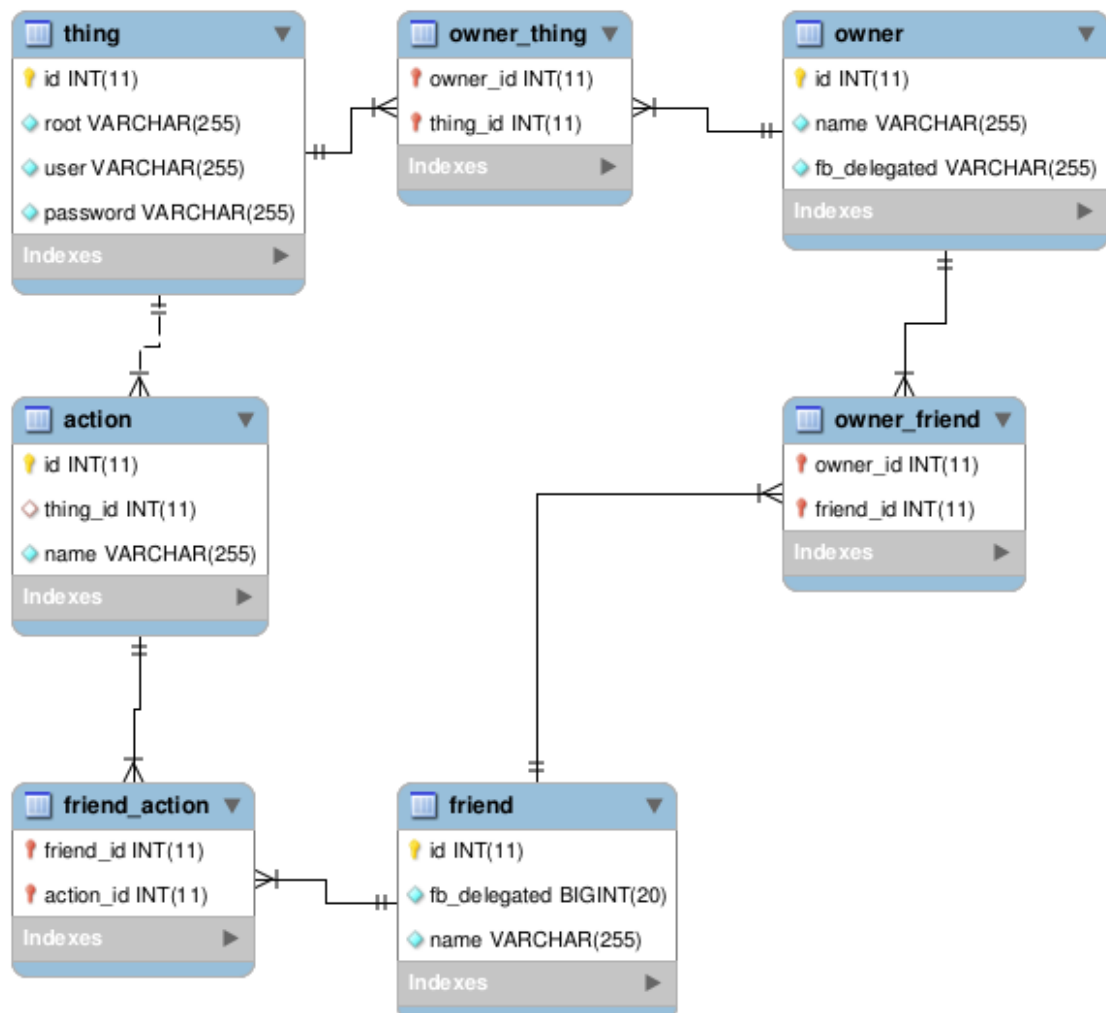


Ilustración 22. Esquema de SAC.

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/{thingId}	

ARQUITECTURA HEXAGONAL SAC

DOMNIO

Explicación de esta capa en (17).

```
// Entities
src/Domain/Entity/Action.php
src/Domain/Entity/Friend.php
src/Domain/Entity/Owner.php
src/Domain/Entity/Thing.php

// Repositories
src/Domain/Repository/OwnerRepository.php
src/Domain/Repository/ThingConnectedRepository.php
src/Domain/Repository/ThingRepository.php

src/Domain/Repository/Action:
src/Domain/Repository/Action/ActionRepository.php
src/Domain/Repository/Friend:
src/Domain/Repository/Friend/FriendRepository.php
```

ThingConnected

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

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

APLICACIÓN

Explicación de esta capa en (17).

Commands

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

```
// Commandos para Action
./src/Application/Command/Action/CreateActionCommand.php
./src/Application/Command/Action/SearchActionByIdCommand.php

// Comandos para Friend
./src/Application/Command/Friend/CreateFriendCommand.php
./src/Application/Command/Friend/SearchFriendByFbDelegatedCommand.php
./src/Application/Command/Friend/SearchFriendByIdCommand.php

// Comandos para Owner
./src/Application/Command/Owner/AddFriendToOwnerCommand.php
./src/Application/Command/Owner/AddThingToOwnerCommand.php
./src/Application/Command/Owner/CreateOwnerCommand.php
./src/Application/Command/Owner/GetFbSharingStatusByOwnerCommand.php
./src/Application/Command/Owner/GetListThingsByOwnerCommand.php
./src/Application/Command/Owner/IsActualUserAnOwnerCommand.php
./src/Application/Command/Owner/SearchOwnerByFbDelegatedCommand.php
./src/Application/Command/Owner/ShareActionWithFriendCommand.php

// Comandos para Thing
./src/Application/Command/Thing/CreateThingCommand.php
./src/Application/Command/Thing/GetActionsByThingIdCommand.php
./src/Application/Command/Thing/GetThingConnectedInfoCommand.php
./src/Application/Command/Thing/MergeThingWithThingConnectedByIdCommand.php
p
./src/Application/Command/Thing/SearchThingByIdCommand.php
./src/Application/Command/Thing/ThingConnected
./src/Application/Command/Thing/ThingConnected/GetThingConnectedCompleteBy
IdCommand.php
```

CommandHandlers

```
// CommandHandler para Action
./src/Application/CommandHandler/Action/CreateActionHandler.php
./src/Application/CommandHandler/Action/SearchActionByIdHandler.php

// CommandHandler para Friend
./src/Application/CommandHandler/Friend/CreateFriendHandler.php
./src/Application/CommandHandler/Friend/SearchFriendByFbDelegatedHandler.php
./src/Application/CommandHandler/Friend/SearchFriendByIdHandler.php

// CommandHandler para Owner
./src/Application/CommandHandler/Owner/AddFriendToOwnerHandler.php
./src/Application/CommandHandler/Owner/AddThingToOwnerHandler.php
./src/Application/CommandHandler/Owner/CreateOwnerHandler.php
./src/Application/CommandHandler/Owner/GetFbSharingStatusByOwnerHandler.php
./src/Application/CommandHandler/Owner/GetListThingsByOwnerHandler.php
./src/Application/CommandHandler/Owner/IsActualUserAnOwnerHandler.php
./src/Application/CommandHandler/Owner/SearchOwnerByFbDelegatedHandler.php
./src/Application/CommandHandler/Owner/ShareActionWithFriendHandler.php

// CommandHandler para Thing
./src/Application/CommandHandler/Thing/CreateThingHandler.php
./src/Application/CommandHandler/Thing/GetActionsByThingIdHandler.php
./src/Application/CommandHandler/Thing/MergeThingWithThingConnectedByIdHandler.php
./src/Application/CommandHandler/Thing/SearchThingByIdHandler.php

// CommandHandler para ThingConnected
./src/Application/CommandHandler/Thing/ThingConnected/GetThingConnectedCompleteHandler.php
./src/Application/CommandHandler/Thing/ThingConnected/SearchThingConnectedActionsHandler.php
./src/Application/CommandHandler/Thing/ThingConnected/SearchThingConnectedBrandHandler.php
./src/Application/CommandHandler/Thing/ThingConnected/SearchThingConnectedNameHandler.php
```

INFRASTRUCTURA

Explicación de esta capa en (17).

Resources

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

La manera que frontend consulta al SAC API para generar URLs para backend esta explicada en (20).


```
// Configuraciones de rutas
./src/Infrastructure/Resources/config/routes/api.yaml
./src/Infrastructure/Resources/config/routes/credentials.yaml
./src/Infrastructure/Resources/config/routes/error.yaml
./src/Infrastructure/Resources/config/routes/friend.yaml
./src/Infrastructure/Resources/config/routes/owner.yaml
./src/Infrastructure/Resources/config/routes/success.yaml
```

Mapeo de ORM

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

```
// Mapeo de ORM
./src/Infrastructure/Resources/mappings/Action.orm.yaml
./src/Infrastructure/Resources/mappings/Friend.orm.yaml
./src/Infrastructure/Resources/mappings/Owner.orm.yaml
./src/Infrastructure/Resources/mappings/Thing.orm.yaml
```

Controladores

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

Controladores de API

ApiUrlGeneratorController es el generador de Urls explicado en (40). OwnerApiController es donde SAC consulta información del *Owner* se puede ver un request y una response a este endpoint en (42). Un ejemplo de unos de ThingApiController también se puede ver en (43) y está relacionado con la entidad ThingConnected vista en (46).

```
// 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) así como endpoints pedidos por Facebook como son /privacy y /conditions. OwnerController se encarga de la parte vista en “Mapa web para *Owner*” (33) del endpoint /owner. FriendController la parte vista en “Mapa web para *Friend*” (36) del endpoint /friend. Por último ThingController muestra “Admin de un Wt” (34) en endpoint thing/info también procesa thing/create.

```
./src/Infrastructure/Controllers/CredentialsController.php
./src/Infrastructure/Controllers/FriendController.php
./src/Infrastructure/Controllers/HasFbSessionController.php <<<<< DUDA
./src/Infrastructure/Controllers/OwnerController.php
./src/Infrastructure/Controllers/ThingController.php
```

Respositorios

```
./src/Infrastructure/Action/MySQLActionRepository.php
./src/Infrastructure/Friend/MySQLFriendRepository.php
./src/Infrastructure/Owner/MySQLOwnerRepository.php
./src/Infrastructure/Thing/MySQLThingRepository.php
./src/Infrastructure/ThingConnected/CurlThingConnectedRepository.php
```

Serializadores

Comandos Symfony

SYMFONY

“LO DE INTERFAZ PARA EVENTOS”

A la hora de crear una zona segura donde solo se pueda llegar con un Usuario logado en Facebook hemos usado “XX”

SEGURIDAD

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 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
- 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

FACEBOOK

FACEBOOK API

Autenticación delegada

Consulta de lista de amigos

FACEBOOK.DEVELOPERS

Creación de usuarios de prueba

Login

Creación de aplicación web

SISTEMA OPERATIVO

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

- VARIABLE DE ENTORNO

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

SSHFS:

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

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

```
alias iot_emulator='cd ~/dev/iot_emulator'
alias iot_emulator_clean_http_requests='rm /home/${USER}/dev/iot_emulator/.idea/httpRequests/*'
alias iot_emulator_php_server_run='iot_emulator && php bin/console server:run'
alias
iot_emulator_shcema_drop_and_create_fixtures_load_NOT_symfonys='iot_emulator && php bin/console doctrine:schema:drop --force && php bin/console doctrine:schema:update --force && php fixture/create_things.php && cd -'
alias sac_clean_http_requests='rm /home/${USER}/dev/sac/.idea/httpRequests/*'
alias sac_fixtures_load='sac && php bin/console doctrine:fixture:load -n && cd -'
alias sac_fixtures_load_append='sac && php bin/console doctrine:fixture:load -n --append && cd -'
alias sac_php_server_run='sac && php bin/console server:run'
alias sac_sandbox='cd /home/${USER}/dev/sac_sandbox/sac_sandbox'
alias sac_sandbox_fixtures_load='sac_sandbox && php bin/console doctrine:fixture:load -n && cd -'
alias sac_sandbox_fixtures_load_append='sac_sandbox && php bin/console doctrine:fixture:load -n --append && cd -'
alias sac_schema_drop_and_create='sac && php bin/console doctrine:schema:drop --force && php bin/console doctrine:schema:update --force && cd -'
alias
sac_schema_drop_and_create_and_fixtures_load='sac_schema_drop_and_create && sac_fixtures_load && cd -'
```

SHELL-SCRIPT PARA PROVISIONAMIENTO

TOD copiar Shell script

HTTPIE

[LINK-INTERNO] DISeNYO-FIXTURES

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

- JQ

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

GIT:

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

GITHUB:

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

- <https://github.com/danielsalgadop/sac>
- https://github.com/danielsalgadop/iot_emulator

AWS:

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

CARACTERÍSTICAS DE MÁQUINA DESPLEGADA EN PRODUCCIÓN

TODO

NGINX:

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

HTML5

JAVASCRIPT

JQUERY

JQUERY-UI

MOUSTACHE

SYMFONY 4

NPM

MYSQL

DOCTRINE

ANÁLISIS RESULTADOS

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

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

Se consigue simplificar la manera de compartir *WTs*

RELACIÓN CON ASIGNATURAS DEL MÁSTER

----- ENTORNO WEB

uso alias para llamar comandos symfony (doctrine)

git

AWS

variable entorno

httpie request

nginx

----- FRONT

html5

javascript

jquery

moustache

----- FRAMEWORKS

symfony 4

----- DESARROLLO EFICIENTE

arquitectura hexagonal

----- SQL

mysql

Doctrine

----- SEO

rutas SEO

----- RENDIMIENTO

----- EMPRENDEDURIA

FRAMEWORKS

EMPRENDIDURÍA

ENTORNO WEB

SEGURIDAD

FRONTEND

PHP

BASES DE DATOS

CONCLUSIONES

"Lo que acabas de explicar"

El proyecto esta hecho en el marco de IOT.

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

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

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

INSTALACIÓN

---- INSTALATION 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

----- 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
```

----- INSTALACION IOT_EMULATOR

(iot_emulator) App scaffold:

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

```
composer install
```

```
npm install
```

(iot_emulator)

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

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

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

MEJORAS

MULTI-OWNER

XXX

CACHEADO INTELIGENTE DE DATOS DE CADA WT

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

Proponemos que aquellos que no necesitan 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 *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