



Escola Tècnica Superior d'Enginyeria
Electrònica i Informàtica La Salle

Trabajo Fin de Máster

Máster en Programción Web de Alto Rendimiento

Alumno

Daniel Salgado Población

Profesor Ponente

Víctor Caballero Codina

ACTA DEL EXAMEN

DEL TRABAJO FIN DE MÁSTER

Reunido el Tribunal calificador en la fecha indicada, el alumno

D. Daniel Salgado Población

expuso su Trabajo Fin de Máster, titulado:

Social Access Controller

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

Barcelona,

VOCAL DEL TRIBUNAL

VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL

Social Access Controller

Universitat Ramon Llull – La Salle Barcelona

Daniel Salgado Población

1 de septiembre de 2019

RESUMEN

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

Este nuevo marco anticipa varios retos, entre los que se suele destacar la accesibilidad y la manera de compartir (o compartición) de los objetos físicos conectados a la red y disponibles vía web llamados Web Things (WTs). El W3Consortium [1] propone un modelo común para describir contrapartes virtuales de los dispositivos físicos del IoT.

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

Para abordar la compartición de los WTs hemos implementado un sistema para compartir WTs con otras personas llamado **Social Access Controller (SAC)** usando la estructura social en redes ya construidas con esta información, como puede ser Facebook.

ABSTRACT

The Internet of Things (IoT) is arupation and and interconnection of devices and objects over a network. It is one of the most outstanding technological advances in recent times. It is increasingly present in everyday life and its applicability in the future is out of any discussion.

This new framework anticipates several challenges, including the accessibility and sharing of physical objects connected to the network and available via web called Web Things (WTs). The W3Consortium [1] proposes a common model for describing virtual counterparts to physical IoT devices.

This work replicates a Web of Things-based controller that allows you to share virtual counterparts between different friends on a social network. To do this, and because you don't have physical devices, you've also created a Web of Things device emulator software.

To address the sharing of WTs we have implemented a system to share WTs with others called Social Access Controller (SAC) using the social structure in networks already built with this information, such as Facebook.

CONTENIDO

RESUMEN	I
ABSTRACT	II
INTRODUCCIÓN	1
Motivación	1
Requisitos	2
Objetivos	2
DISEÑO GENERAL	4
Acople al modelo WT del W3Consortium	4
Acople al SAC de Dominique Guinard	6
Diseño de Software. Arquitectura Hexagonal	6
Diseño de Software. API REST	8
Diseño de Base de Datos	9
DATOS DE PRUEBA (FIXTURES)	12
Usuarios de prueba en Facebook	12
WT de pruebas lot_emulator:	13
Datos pruebas SAC	13
TESTS	15
lot_emulator	¡Error! Marcador no definido.
SAC	16
BACKEND IOT_EMULATOR	17
Estructura básica WT	17
Esquema Base de datos lot_emulator	18
Arquitectura REST lot_emulator	19
Arquitectura hexagonal de lot_emulator	21
Seguridad lot_emulator	23
FRONTEND SAC	24
Raíz del proyecto	24
Mapa web para Owner	25
Admin de un WT	26
Ver listado de Friends	27
Resultado de compartición	27
Mapa web para Friend	27
Página de Error y de Éxito	27
BACKEND SAC	29
Autenticación Delegada en Facebook	29
Proceso de Login	29

Crear Owner en SAC	31
Crear Actions en SAC	31
Acciones sobre los WTs.....	31
API SAC	32
Esquema Base de datos SAC	37
Arquitectura REST SAC del API SAC.....	39
Arquitectura Hexagonal SAC.....	40
Seguridad SAC	51
TECNOLOGÍAS USADAS	52
PHPStorm.....	52
Facebook.....	52
Sistema operativo	52
github.....	54
aws ec2	54
nginx.....	54
html5, CSS, Bootstrap y moustache	55
javascript, JQuery y JQuery-UI, Twig	55
symfony 4	56
npm	56
mysql y Doctrine	56
Composer	56
ANÁLISIS RESULTADOS.....	57
Relación con Asignaturas del máster.....	57
CONCLUSIÓN	59
Trabajo futuro	60
ANALISIS ECONOMICO DEL PROYECTO.....	61
GLOSARIO	61
BIBLIOGRAFÍA	61

INTRODUCCIÓN

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

Hemos seguido la manera de compartir los WT sugerido en el trabajo [612] mediante un Social Access Controller (SAC). Al compartir mediante SAC hemos conseguido compartir WT con las siguientes características. SAC actúa de proxy de compartición permitiendo compartir de manera granulada ciertas partes de los WTs. SAC administra el acceso a los WTs basándose en la estructura de redes sociales ya existentes, como puede ser Facebook.

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

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

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

MOTIVACIÓN

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

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

WT multiusuario

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

Compartir Credenciales

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

REQUISITOS

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

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

OBJETIVOS

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

Compartición mediante Social Access Controller (SAC)

Siguiendo la idea de un SAC propuesta por Dominique Guirnard et al en [2] basándonos en su idea de “Compartición basada en Redes Sociales” nos fijamos los siguientes objetivos.

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

Cumplimiento de requisitos

Vamos a analizar como SAC cumple con los requisitos expuestos en pag 2.

Cumplimos el 2 al administrar la compartición desde la aplicación única llamada SAC. Al guardar las credenciales al dar de alta el WT en SAC cumplimos también el 2. Como SAC usa las credenciales del Owner puede ver todos los Actions y mostrar solo aquellos recursos a los que el Owner haya dado el permiso, cumplimos así el 2. La explicación de cómo cumplir el 2 queda más clara al explicar la estructura de base de datos en la 37 al ver como existe una relación N:M entre action y friend. El 2 de almacenar la mínima información posible se cumple en la parte del WT al consultar

sus Actions en caliente y la parte de Facebook porque es Facebook quien nos provee de la estructura de red social, para más detalle se puede consultar en 38 al ver los datos almacenados en SAC.

Capa de accesibilidad mediante lot Emuator

Para poder desarrollar SAC vimos la necesidad de tener tanto WTs como una capa que los conectase entre ellos. Para ello hemos creado una aplicación que funciona a modo de hub de WTs. Esta aplicación la hemos llamado lot_emulator. Se apoya en el modelo de WT propuesto por W3Consortium [1]. Emula la capa web por encima de los WTs y hace que estos sean homogéneos. En los siguientes apartados explicamos el nivel de acople llegado en comparación con el propuesto por W3Consortium.

DISEÑO GENERAL

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

En las siguientes secciones comenzamos hablando del nivel de acople o ajuste de las referencias bibliográficas Para llevar a cabo el Trabajo Fin de Máster hemos simplificado los modelos propuestos tanto por [1] y [2]. Estas simplificaciones las hemos hecho por restricciones de tiempo y por no añadir complejidad innecesaria. A continuación, veremos qué partes sí hemos construido y explicaremos cuáles no.

En ambos proyectos hemos usado diseño de software siguiendo la arquitectura hexagonal; sus capas y centrándonos en capa Infraestructura para mirar qué componentes se han creado. Segundo el servicio REST donde explicaremos tanto los endpoints generados como las herramientas usadas para sus respectivas APIs. En la parte de Diseño de Base de datos nos fijaremos en las similitudes y diferencias de ambos proyectos.

ACOPLE AL MODELO WT DEL W3CONSORTIUM

W3Consortium propone una serie de requerimientos y modela qué y cómo debe ser una WT. En el documento de referencia [1] determina en punto 4 el patrón de integración. Y en el punto 5 los requerimientos para un WT fijando tanto los requests que deben comunicarse hacia ellos como los responses de estos. A continuación, explicamos en qué medida nos hemos acoplado.

Patrón de Integración de los WT en W3Consortium

W3Consortium refleja de tres maneras de distintas la conectividad de los WTs: Directa, Gateway, o Cloud. Nosotros no podemos acoplarnos estrictamente a ninguna de ellas ya que emulamos la capa de accesibilidad con el lot_emulator. Como veremos en la 20 esta falta de patrón de integración tiene impacto a la hora de definir las rutas REST.

Requerimientos para un WT en w3Consortium

Tomando como referencia el punto 5 del W3Consortium -Web Thing requirements-. Donde determinan que requisitos debe cumplir un objeto conectado a una red para poder ser considerado con un WT. Hemos elaborado las siguientes tablas.

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

Nivel 0 – Un WT DEBE

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

Tenemos un alto grado de adaptación con estos requisitos. Cada WT no es un único servidor porque todos los WT emulados se encuentran detrás del lot_emulator que funciona de hub.

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

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

Nivel 2 – WT PODRÍA

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

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

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

ACOPLE AL SAC DE DOMINIQUE GUINARD

Realmente el SAC creado en este Trabajo Fin de Máster se basa al propuesto por Dominique Guinard [2]. Por aún quedan partes no acopladas. La primera diferencia es el nivel que dependencia total de Facebook mientras que Guinard habla de administración basada en estructura social basada en varias redes sociales Otra mejora es la hora de descubrir WTs Guinard propone que el SAC lo haga de manera automática rastreando la red. Tampoco ofrecemos la posibilidad de discernir qué verbo HTTP quiere el Owner ofrecer al Friend.

DISEÑO DE SOFTWARE. ARQUITECTURA HEXAGONAL

El código de SAC y de lot_emulator se ha hecho siguiendo una arquitectura hexagonal. Construyendo las siguientes capas **Dominio, Aplicación e Infraestructura**. Permitiendo desacoplar la lógica de cada capa. Cada capa corresponde con una carpeta en la raíz del proyecto. Cada capa depende únicamente de la capa que tiene por debajo. Las capas internas no conocen las capas externas.

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

Dominio

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

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

Aplicación

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

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

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

Infraestructura

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

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

Controladores

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

Implementación de Repositorios

Las implementaciones de los repositorios responden a sus correspondientes contratos de la capa de Dominio. Esto permite que sea fácil cambiar implementaciones de repositorios usando diferentes tecnologías, por ejemplo, en cuanto al framework PHP cambiarlo por Laravel o ZendFramework o cambiar la base de datos a Postgresql o mongo. Estos cambios se harían sin tener que cambiar el código de Dominio.

Serializadores

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

Comandos Symfony

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

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

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

Ejemplo de arquitectura hexagonal: ORM Doctrine

Un ejemplo es la manera en que desacoplamos las capas es el uso de MySQL y más en concreto con el ORM Doctrine. En Dominio Las entidades se definen como entidades puras y ofrecen interfaces en los Repositorios para interactuar sobre ellos.

La capa de aplicación implementa estos interfaces y además usa el ORM Doctrine. 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 capas de Dominio y Aplicación se mantuviesen intactas.

DISEÑO DE SOFTWARE. API REST

API REST es un interfaz entre sistemas que usa HTTP para obtener datos. Es un protocolo cliente/servidor sin estado. Usa los verbos HTTP y URI para obtener o manipular datos. Como veremos más adelante hemos construido 2 API REST una para lot_emulator y otra para SAC. 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). Ahora vamos a mostrar un listado de los endpoint generados.

Endpoints

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

lot_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

Como vemos más adelante SAC posee una API SAC, se puede consultar el detalle en esta sección (Arquitectura REST SAC). El resto de las rutas quedan explicadas en resto de documento. En el frontend SAC o en backend SAC.

Verbo HTTP	Endpoint
GET	/
GET	/loginOk
GET	/api/owner
GET	/api/thing/thingId
POST	/owner/share/action/actionId/friend/friendId
GET	/api/url/provider/thing
GET	/api/url/provider/api/thing

GET	/api/url/provider/api/share/action
GET	/privacy
GET	/conditions
GET	/error
GET	/friend/thing/{thingId}/action/{actionId}
GET	/friend
GET	/owner
GET	/owner/things
POST	/owner/create
GET	/success
POST	/thing/create
GET	/thing/{thingId}

Lanzamiento Peticiones HTTP

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

```
tests/request
```

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

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

DISEÑO DE BASE DE DATOS

Similitudes entre lot_emulator y SAC

En ambos proyectos usan el ORM Doctrine para relacionar entidades de base de datos con objetos PHP. Y lo hacemos con arquitectura hexagonal. Al usar Doctrine la capa de Infraestructura tiene acceso al entityManager de Doctrine que posee algunos métodos muy útiles como find o findOneBy.

Diferencias entre lot_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 relación 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/Infraestructure/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. Hemos creado tres perfiles ficticios que funcionan como Friends de nuestra Owner Elisabeth.

- Linda De las Mareas
- Susan
- Mary



Ilustración 1 Usuarios de prueba definidos en Facebook

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

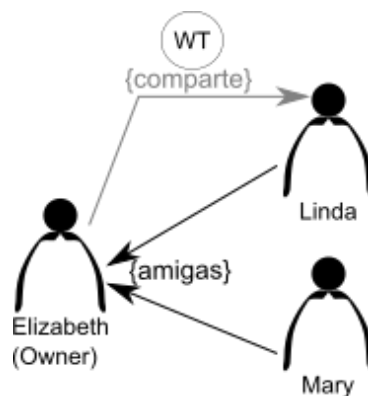


Ilustración 2 Compartición entre amigas de Owner

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

WT DE PRUEBAS IOT_EMULATOR:

Para todos los WT usados en pruebas el usuario esperado en tests es “user” y la contraseña es “password”. Hemos creado script `create_things.php`. Este script lanza peticiones HTTP para crear WT de prueba. Estos datos quedan persistidos en la base de datos.

Se lanza desde la raíz del proyecto

```
php fixture/create_things.php
```

Los datos de prueba tienen esta estructura de naturaleza incremental. Nótese que cada nuevo id incrementa el número de action y properties. Así 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 doctrine para popular sus bases de datos que son los Fixtures. Se lanza con este Comando de Symfony

```
php bin/console doctrine:fixture:load
```

El código se puede encontrar en

```
src/DataFixtures/Sac.php
```

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

TESTS

IOT_EMULATOR

Hemos hecho 2 tipos de tests. basados en PHP puro y los hechos con PHPUnit-

No usan PHPUnit

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

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

Usan PHPUnit

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

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

Peticiones HTTP

Hemos creado estas peticiones para testear que la API funcionaba como esperábamos. Son tests que actuan sobre WT están divididas en tests que deben fallar (`must_work`) y tests que deben funcionar (`must_fail`).

```

tests/requests/get_value_of_property/must_fail/get_value_of_inexistent_pro
perty.http
tests/requests/get_value_of_property/must_fail/get_value_of_inexistent_thi
ng.http
tests/requests/get_value_of_property/get_value_of_property.http
tests/requests/execute_action/must_work/exectue_action.http
tests/requests/execute_action/must_fail/incorrect_action_name_in_payload.h
ttp
tests/requests/execute_action/must_fail/incorrect_action_name_in_url.http
tests/requests/get_actions/must_work/get_actions.http
tests/requests/get_actions/must_fail/get_actions_empty_credentials.http
tests/requests/get_actions/must_fail/get_actions_incorrect_password.http
tests/requests/get_thing/must_work/get_thing.http
tests/requests/get_thing/must_fail/invalid_credentials.http
tests/requests/get_thing/must_fail/no_credentials.http
tests/requests/get_thing/must_fail/search_non-existent.http
tests/requests/list_of_things/must_work/list_of_things.http
tests/requests/delete/delete_thing_by_id.http
tests/requests/update/must_work/update_id_1_action_jsonAction1.http
tests/requests/update/must_fail/non-existing_property.http
tests/requests/update/must_fail/non-existing_action_nor_property.http
tests/requests/update/must_fail/non-existing_action.http
tests/requests/create/must_work/create_thing.http
tests/requests/create/must_fail/noActionPropertyConcordance/wrong_property
_key.http
tests/requests/create/must_fail/noActionPropertyConcordance/missing_proper
ty_key.http
tests/requests/create/must_fail/noActionPropertyConcordance/wrong_action_k
ey.http
tests/requests/create/must_fail/noActionPropertyConcordance/missing_action
_key.http
tests/requests/create/must_fail/wrongCredentials/no_user_nor_password.http
tests/requests/create/must_fail/wrongCredentials/no_user.http
tests/requests/create/must_fail/wrongCredentials/no_password.http
tests/requests/create/must_fail/insufficient_data/no_payload.http
tests/requests/create/must_fail/insufficient_data/no_name.http
tests/requests/create/must_fail/insufficient_data/empty_json.http
tests/requests/create/must_fail/insufficient_data/no_brand.http

```

SAC

Peticiones HTTP

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

```

./tests/requests/thing/create/create_thing.http
./tests/requests/owner/create/must_work
./tests/requests/owner/create/must_work/create.http
./tests/requests/api/thing/must_work/GetThingById.http
./tests/requests/api/owner/must_work/GetBasicInfo.http

```


BACKEND IOT_EMULATOR

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

ESTRUCTURA BÁSICA WT

Nuestro WT tiene características propias definidas por nosotros que no son del W3Consortium. Aquí explicamos la diferencia.

Zona pública y Zona privada

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

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

Respuesta JSON zona privada

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

```
http http://iot.socialaccesscontrller.tk/1 user:user password:password
```

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

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

Respuesta JSON zona pública

Ahora lanzamos la petición sin las credenciales

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

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

```
{
  "id": 1,
  "name": "thing_name1",
  "brand": "thing_brand1"
}
```

Relación entre Actions y Properties

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

url/actions/action_name1

Devolvería

property_value1

Es decir, nuestros Actions sirven para modificar el valor de la propety. Son punteros a los valores de Properties.

WT Endpoints

Los endpoints de los WT equivalen a las claves primarias de la tabla thing de la base de datos. Los WTs se identifican con ids numéricos que coinciden con el id interno de la base de datos. Así thing_1 tendrá endpoint /1 y será el id con primary key de base de datos igual a 1 en tabla thing.

ESQUEMA BASE DE DATOS IOT_EMULATOR

En la Ilustración 3 se muestra el esquema de base de datos que da soporte a la aplicación lot_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 thing, es la entidad que engloba a las demás.

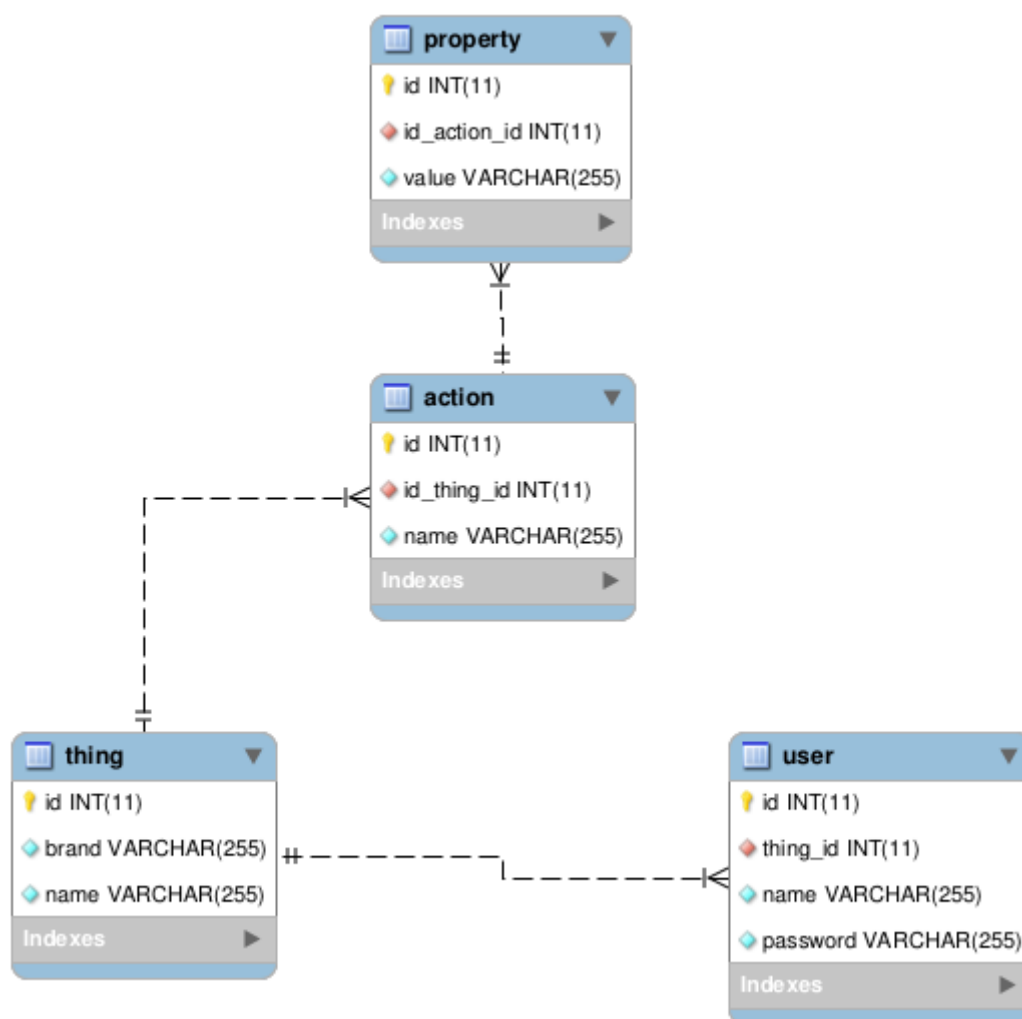


Ilustración 3. Esquema de la base de datos del lot_emulator.

ARQUITECTURA REST IOT_EMULATOR

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

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

Endpoint

GET /

Este endpoint es propio de nuestro SAC. No está definido ni en [LINK1] ni [LINK2] y rompe la naturaleza REST del API. Existe por la naturaleza de Hub que posee el lot_emulator. Muestra un listado con todos los WT mostrando su parte pública.

GET /{id} Sin credenciales

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

GET /{id} Con credenciales

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

POST /create

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

```
tests/requests/create/must_work/create_thing.http
```

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

```
POST ${IOT_EMULATOR}/create
Content-Type: application/json
user: user
password: password

{
  "name": "jsonName",
  "brand": "jsonBrand",
  "links": {
    "properties": [
      {"property_name1": "property_value1"},
      {"property_name2": "property_value2"}
    ],
    "actions": ["property_name1", "property_name2"]
  }
}
```

POST /{id}/actions/{action_name}

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

GET /{id}/properties/{property_name}

Muestra el valor de la Property con nombre {property_name} del WT con id igual a {id}

GET /{id}/actions

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

ARQUITECTURA HEXAGONAL DE IOT_EMULATOR

Dominio lot_emulator

Hemos creado Entidades y Repositorios.

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

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

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

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

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

/**
 * @ORM\OneToOne(targetEntity="App\Domain\Entity\User", mappedBy="thing",
 cascade={"persist", "remove"})
 */
private $user;
```

Aplicación lot_emulator

Explicación de capa de Aplicación en 6. Hemos creado Commands como CommandHandlers y DTO

Estructura de Command y CommandHandlers lot_emulator

Existe una relación 1 a 1 entre todos los Commands y sus CommandHandlers. El siguiente código muestra un listado de Comandos y CommandHandlers respectivamente.

```
// Commands
src/Application/Command/Thing/CreateThingCommand.php
src/Application/Command/Thing/ExecuteActionCommand.php
src/Application/Command/Thing/SearchThingByIdCommand.php

// CommandHandlers
src/Application/CommandHandler/Thing/CreateThingHandler.php
src/Application/CommandHandler/Thing/ExecuteActionHandler.php
src/Application/CommandHandler/Thing/SearchThingByIdHandler.php
```

DTO específico de lot_emulator

Además de los DTOs creados en Application/Command hemos usado uno para transmitir las credenciales recibidas. es una estructura de datos independiente a nuestro modelo de datos y solo contiene datos y ninguna lógica. Además, hemos usado un DTO específico para encapsular las credenciales de WT.

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

Infraestructura lot_emulator

Existe una explicación a esta capa en Diseño General.

Controladores lot_emulator

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

Repositorios lot_emulator

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

Comandos Symfony lot_emulator

Namespace

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

```
src/Infrastructure/Thing/Command/SearchThingByThingIdCommand.php
```

Listado completo

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

```
$ php bin/console
app:Thing:SearchThingById          Given a (int) id, (int) user and
(int) password searches Thing
```

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

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

Resultado

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

Serializadores lot_emulator

Para serializar; Thing con y sin Credenciales serializar y Action.

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

SEGURIDAD IOT_EMULATOR

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

Las peticiones con credenciales incorrectas o sin credenciales.

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

FRONTEND SAC

URL: <https://socialaccesscontroller.tk>

Desde un punto de vista funcional y visual el SAC tiene tres varias partes diferenciadas. La primera es la “Raíz del proyecto” donde se desencadena el proceso de Login visto en [29. La segunda es para usuario con rol Owner y la tercera para usuario con rol Friend.

En apartado “Raíz del proyecto” 24 mostramos el frontend de la entrada para cualquier rol. En apartado “Mapa web para Owner” de la 24 mostramos y explicamos las funcionalidades disponibles para Owner. En apartado “Mapa web para Friend” mostramos y explicamos las funcionalidades disponibles para Friend.

Tal como explicamos en backend de la [29 solo el Owner y sus Friends podrán acceder más allá de “Raíz del proyecto”

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

RAÍZ DEL PROYECTO

La Ilustración 4 es una captura de pantalla donde se muestra en la raíz del proyecto, pide al usuario logarse vía Facebook.

El detalle de como gestiona SAC la Autenticación delegada se puede ver en la 29 para el detalle sobre los datos guardados en base de datos de SAC en la 38.

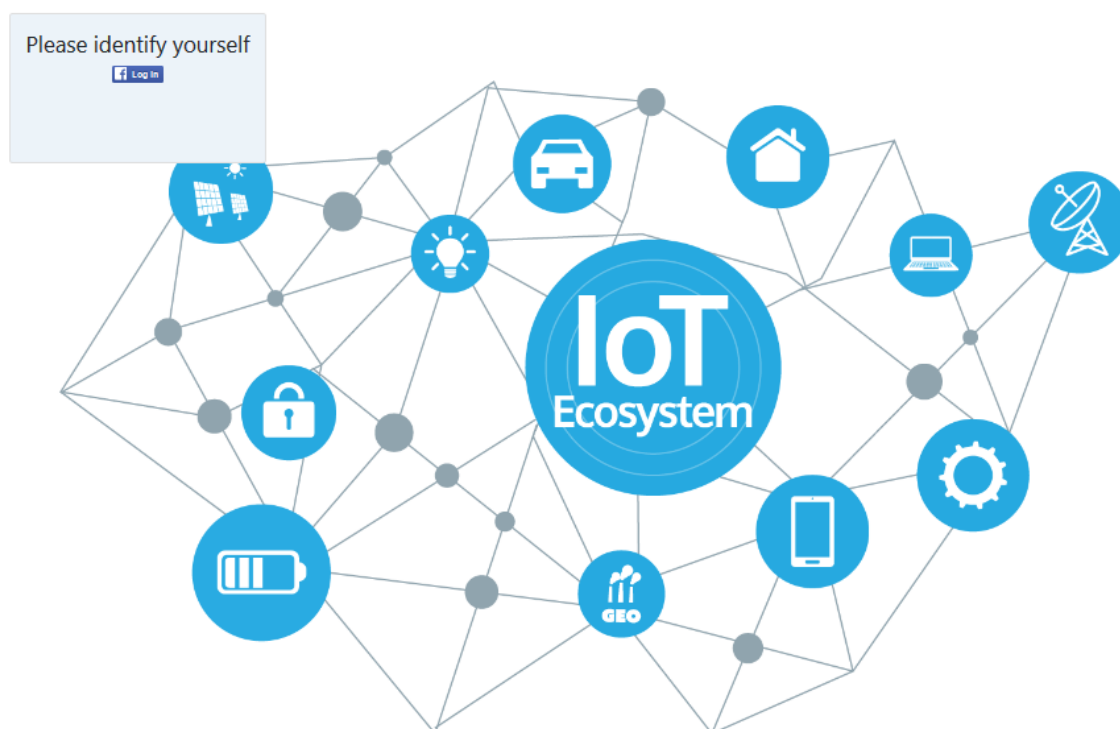
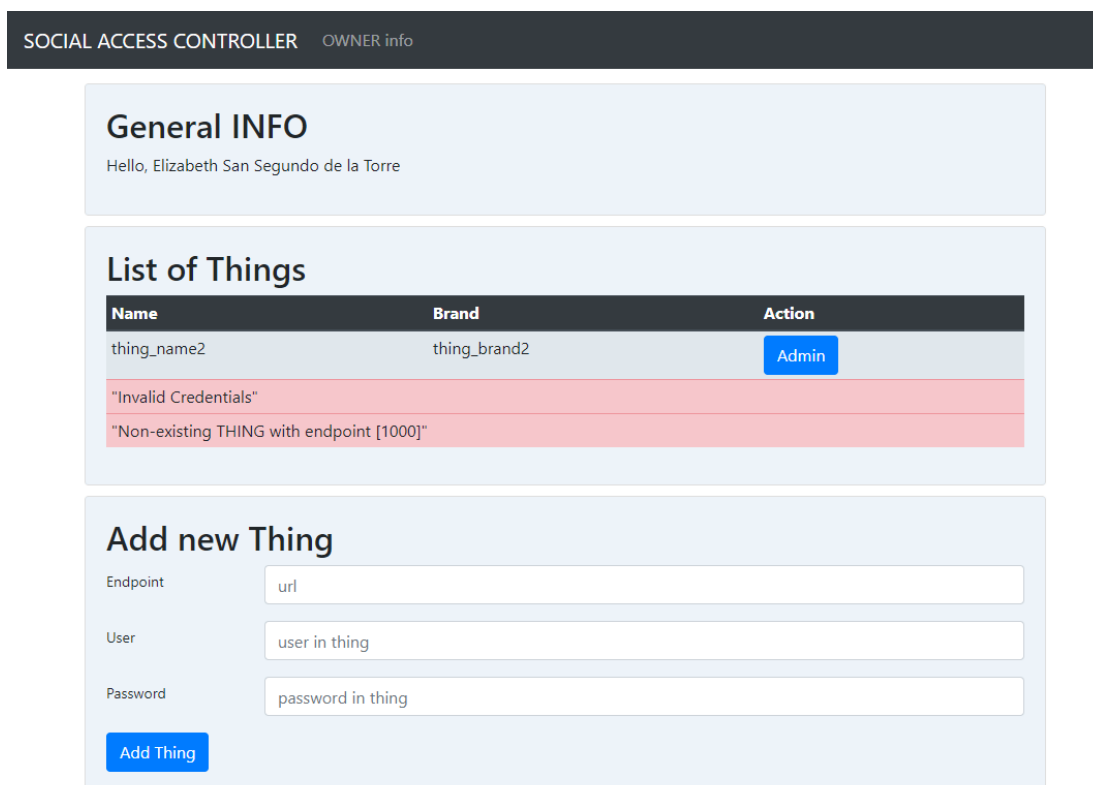


Ilustración 4. Captura de pantalla de la Raíz del proyecto.

MAPA WEB PARA OWNER

Index del Owner

La Ilustración 5 es una captura de pantalla al índice del Owner encontrado en <https://socialaccesscontroller.tk/owner>. Owner accede a él después de logarse. Desde aquí puede ver su propia información general 25, el listado de todos los WTs dados de alta en SAC 25 con información sobre cada WT y el estado de conexión y finalmente un formulario para dar de alta nuevo WTs 26.



The screenshot shows the 'OWNER info' page of the 'SOCIAL ACCESS CONTROLLER'. It is divided into three main sections:

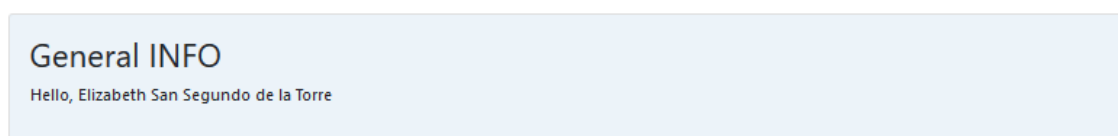
- General INFO:** Displays a greeting: 'Hello, Elizabeth San Segundo de la Torre'.
- List of Things:** A table listing items with columns 'Name', 'Brand', and 'Action'.

Name	Brand	Action
thing_name2	thing_brand2	<button>Admin</button>
"Invalid Credentials"		
"Non-existing THING with endpoint [1000]"		
- Add new Thing:** A form with three input fields: 'Endpoint' (containing 'url'), 'User' (containing 'user in thing'), and 'Password' (containing 'password in thing'). Below the fields is a blue 'Add Thing' button.

Ilustración 5. Visión general del Index del Owner.

Información general del Owner

La información general del Owner es la parte del index que mostramos en la Ilustración 6. Estos datos son obtenidos de Facebook durante la creación de nuevo Owner 31.



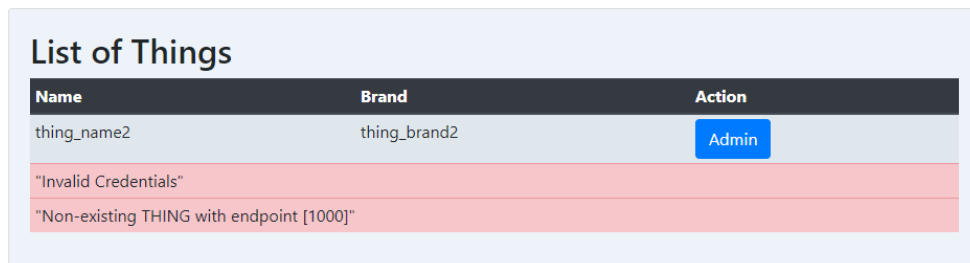
This screenshot shows only the 'General INFO' section of the page, displaying the text: 'Hello, Elizabeth San Segundo de la Torre'.

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

Listado de WTs dados de alta en SAC

La Ilustración 7 es una captura de pantalla donde se ve el listado de todos los WT dados de alta por Owner en SAC. En este ejemplo existen tres WT dados de alta cuyos nombres son thing_name1, thing_name2 y thing_name3. Corresponden con los datos de prueba de lot_emulator usados durante el desarrollo vistos en 13.

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



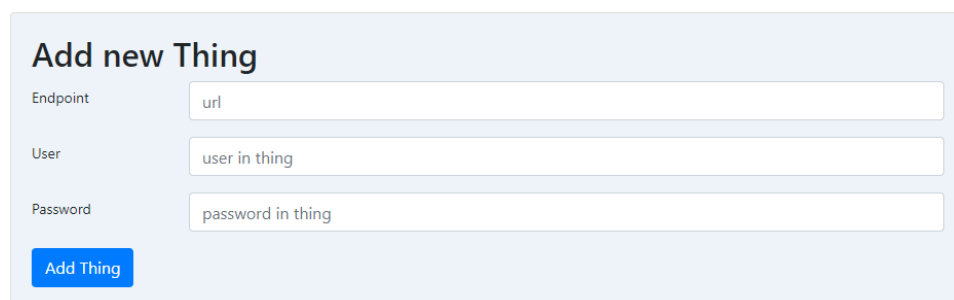
Name	Brand	Action
thing_name2	thing_brand2	<button>Admin</button>
"Invalid Credentials"		
"Non-existing THING with endpoint [1000]"		

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

Formulario para dar de alta nuevo WT

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

Una de las mejoras explicada y propuesta en la 60 es la posibilidad de que SAC descubra WTs para hacer más cómodo el proceso de alta.

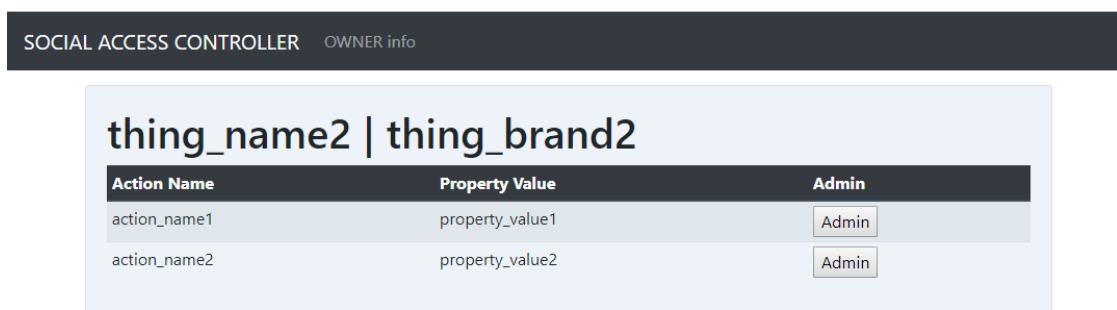


Add new Thing	
Endpoint	<input type="text" value="url"/>
User	<input type="text" value="user in thing"/>
Password	<input type="text" value="password in thing"/>
<button>Add Thing</button>	

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

ADMIN DE UN WT

La Ilustración 9 es una captura de pantalla de administración de un WT en <https://socialaccesscontroller.tk/thing/{id}>. Se accede tras dar al clicar “Admin” para un WT concreto desde el “Index del Owner”. Mostramos el nombre del WT el Brand y un listado de Actions y sus Properties propios del WT.



SOCIAL ACCESS CONTROLLER OWNER info		
thing_name2 thing_brand2		
Action Name	Property Value	Admin
action_name1	property_value1	<button>Admin</button>
action_name2	property_value2	<button>Admin</button>

VER LISTADO DE FRIENDS

Al presionar botón de “Admin” en el “Index de un Thing” como muestra la Ilustración 10 aparece un popup con el listado de Friends. Si esta Action puede ser compartida aparece un botón con “Share” en caso contrario aparece “(already shared)”

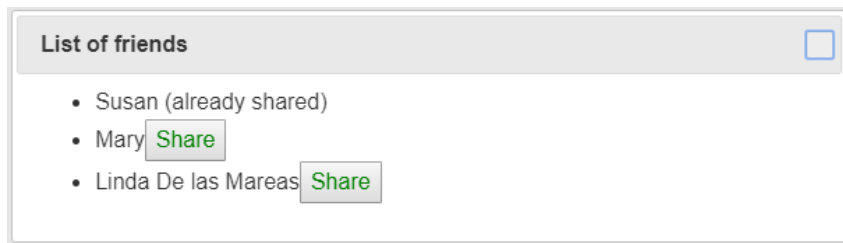


Ilustración 10. Listado de amigos.

RESULTADO DE COMPARTICIÓN

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

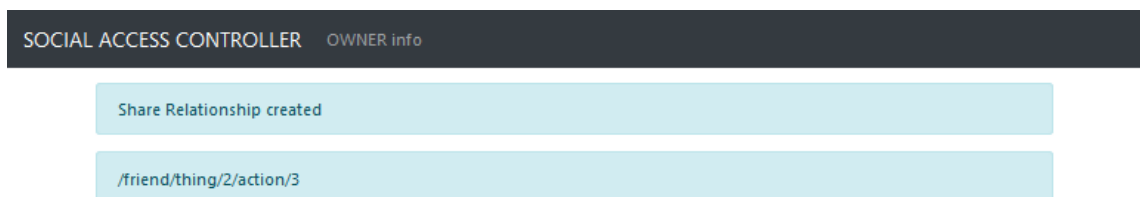


Ilustración 11. Resultado de la Compartición

MAPA WEB PARA FRIEND

El punto de entrada para Friend <http://socialaccesscontroller.tk/friend> es muy similar al del Owner, pero con menos funcionalidades. Se muestra información general del Friend (nombre) y dispone de un listado de Actions compartidas por el Owner. Cuando pulsa el botón “Show” se puede ver el Action. 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

Cuando un evento acaba satisfactoriamente, por ejemplo, cuando un Action se comparte o cuando se añade un WT aparece esta página. Una captura de pantalla se puede encontrar en la Ilustración 11.

Página de Error

La página de error es muy parecida a la de Éxito, salvo que el error se muestra en rojo y no tiene header definido, se llega a ella por ejemplo cuando un Friend intenta ver un Action que no ha sido compartido.

BACKEND SAC

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

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

AUTENTICACIÓN DELEGADA EN FACEBOOK

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

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

El token persistente de Facebook en SAC lo hemos llamado `"fb_delegated"`.

PROCESO DE LOGIN

En la siguiente ilustración Ilustración 12 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:

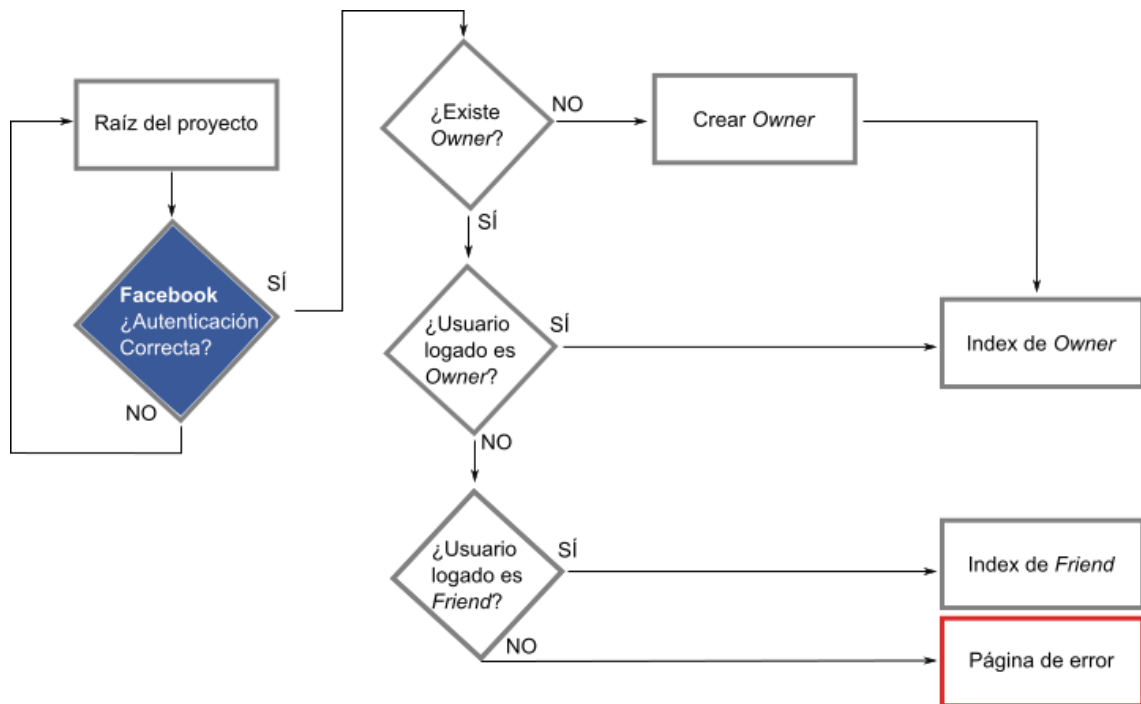


Ilustración 12. Caso de uso durante Login.

Camino “Raíz del proyecto”

SAC no dejará pasar ningún usuario de “Raíz de proyecto” si Facebook no devuelve un accessToken. Es este punto donde se pone de manifiesto el acceso delegado.

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

Camino “Crear Owner”

Este camino acaba de la misma manera que “Index Owner” pero con el paso extra de “Crear Owner”. La diferencia con el camino “Index Owner” es que SAC en este caso no tiene un Owner definido. Como SAC está preparado para que solo exista un único Owner de WT es el primer usuario de Facebook logado quien asume el papel de Owner.

Camino “Index de Owner”

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

Camino “Index de Friend”

SAC reconoce al usuario logado como Friend de Owner y muestra “Index de Friend”.

Camino “Página de error”

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

CREAR OWNER EN SAC

Cuando SAC crea al Owner ocurren varias cosas. Primero se guarda en tabla owner el fb_delegated (token persistente de Facebook) y el nombre del Owner. Segundo consultamos a Facebook el listado de Friends y lo guardamos en SAC guardando todos los fb_delegated (tokens de Facebook) de cada Friend en tabla friend. Una mejora propuesta en 60 es el poder actualizar la lista de Friends en distintos momentos.

CREAR ACTIONS EN SAC

Se persisten las Actions en SAC al consultar por los Actions de un WT desde la página <http://socialaccesscontroler/thing/{id}> y es la primera vez que SAC consulta al lot_emulator por estos datos. En caso contrario evitamos volver a guardarlos.

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

ACCIONES SOBRE LOS WTs

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

Dar de alta nuevo WT

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

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

Obtener información de un WT

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

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

Compartir un Action con un friend

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

API SAC

Hemos visto la necesidad de crear un API SAC por varias razones. Primero la necesidad de conseguir una respuesta rápida del backend. Tener un frontend liviano que haga consultas Ajax por datos que necesite y no cargar el Response con datos que quizás no se usen. Segundo por seguridad, para evitar enviar información sensible al frontend como son el user y password de cada WT. Además, como veremos más adelante para poder usar el sistema de rutas de Symfony en peticiones Ajax.

Diagrama de flujo entre Navegador SAC y lot_emulator

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

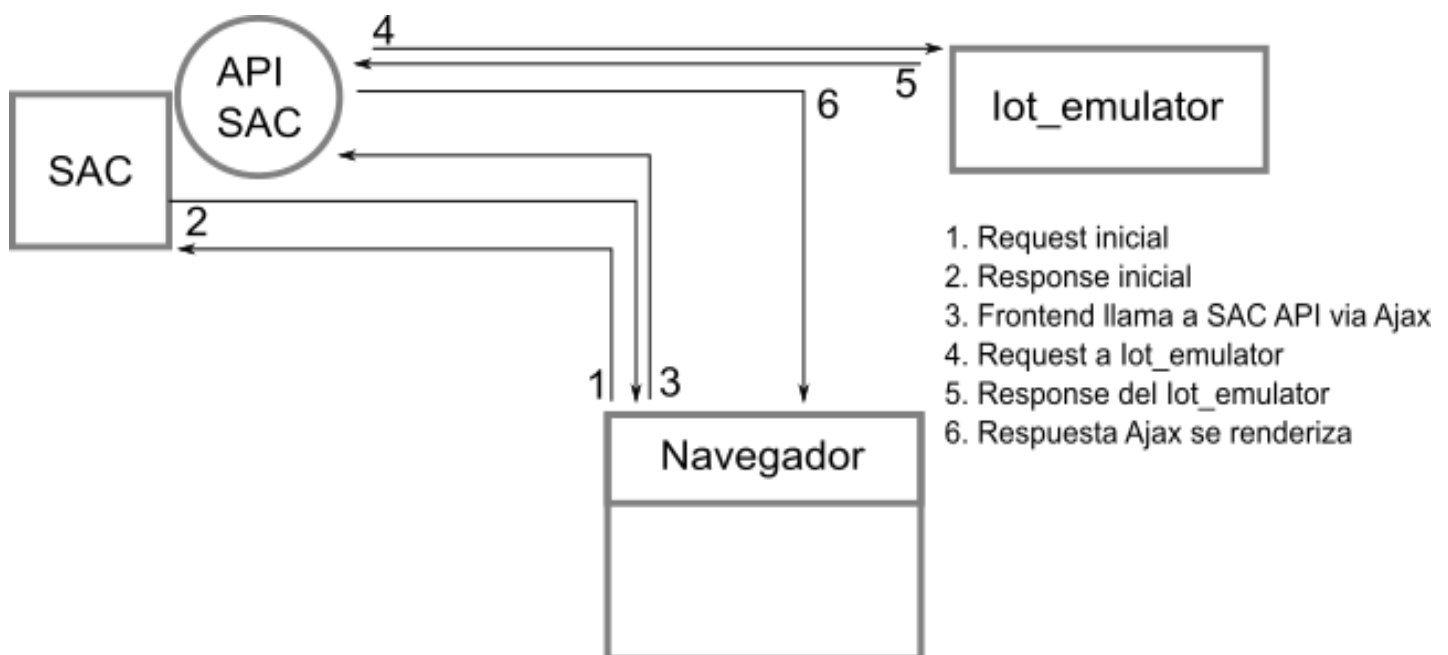


Ilustración 13 Obtención de datos vía Ajax del lot_emulator

Usar el Sistema de enrutamiento de symfony desde frontend

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

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

Desde el backend en un AbstractController se usaría el método de Symfony `generateUrl`:

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

Desde el frontend con función de Twig `path`:

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

Vemos como en ambos caso la ruta se resuelve al “/ruta/value_param”

Problema en twig al usar Ajax

Nos hemos visto en la incapacidad de poder usar la función específica en Twig `path()`. Esta función de Twig para resolver la url necesita en ese momento el valor concreto del parámetro. Tener un frontend que pide valores vía Ajax no impide poder usarla.

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 pero sin los pasos 4 y 5 ya que la definición de la ruta está en SAC y no en `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 visto en 25 este listado se construye en “Index de Owner”. Es un buen ejemplo porque usamos los dos tipos de uso explicados anteriormente, estos son; la obtención de datos vía `lot_emulator` como la generación de urls.

La Ilustración 14 es una captura de pantalla tras la carga completa de la página. Es el resultado final del proceso pero nos sirve para ver las peticiones que han ocurrido. Aunque el listado incluye tres WT nos vamos a centrar en el WT con id igual a 1. Las llamadas para id 2 y 3 son equivalentes.

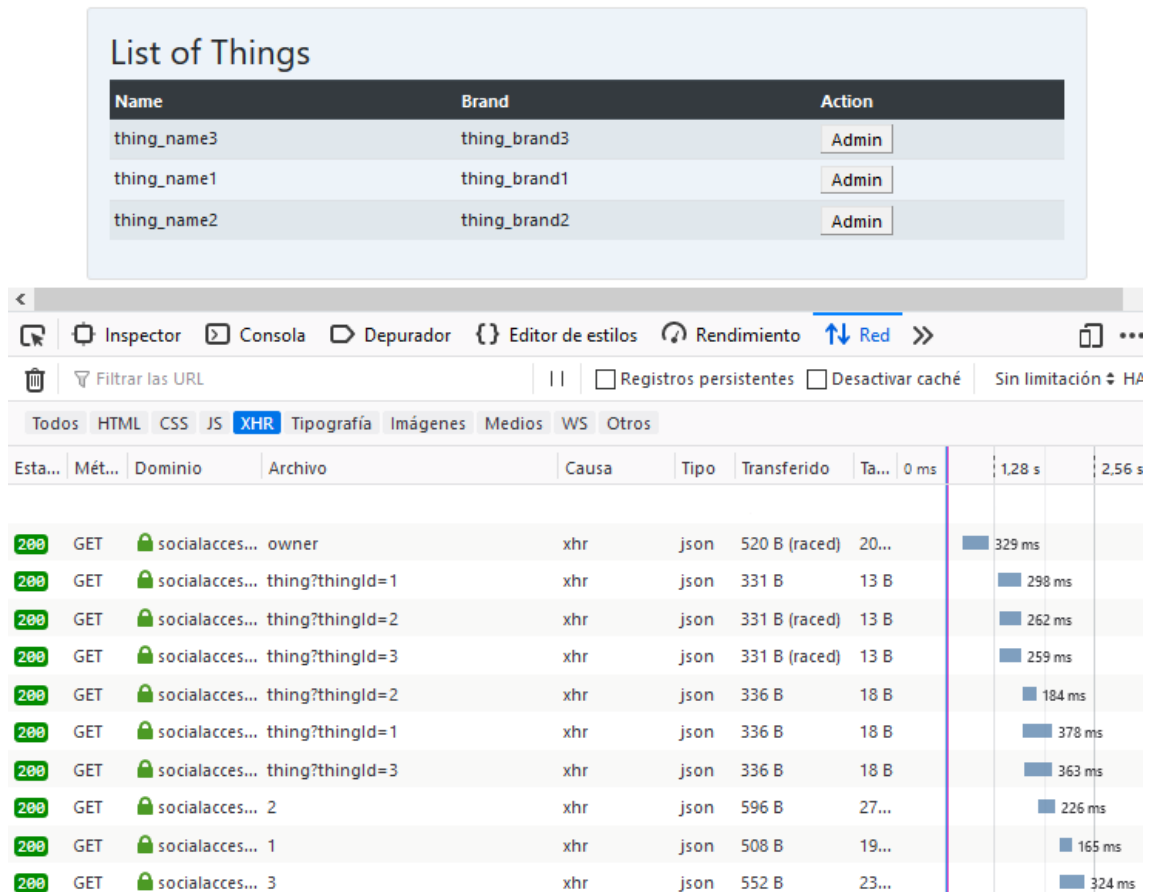


Ilustración 14. Ejemplo de uso de API SAC al generar el Listado de WT.

El request inicial

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

Response inicial

El response inicial lo genera Symfony en

```
App\Infrastructure\Controllers\OwnerController
```

Que devuelve un html liviando de la página 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
- url para administrar cada WT
- 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 Ilustración 15 es una captura de pantalla donde podemos ver la llamada a el endpoint api/owner de API SAC.

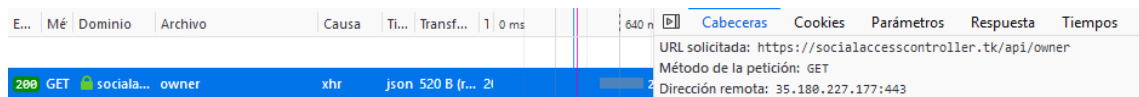


Ilustración 15. Request a api/owner

La Ilustración 16 es una captura de pantalla donde podemos ver la response que devuelve: Vemos que devuelve un json con el nombre del Owner y un array llamado things con información de los WTs, la clave friendsByld no nos importa ahora. Toda esta la información contenida en esta respuesta se encuentra almacenada en base de datos de SAC.

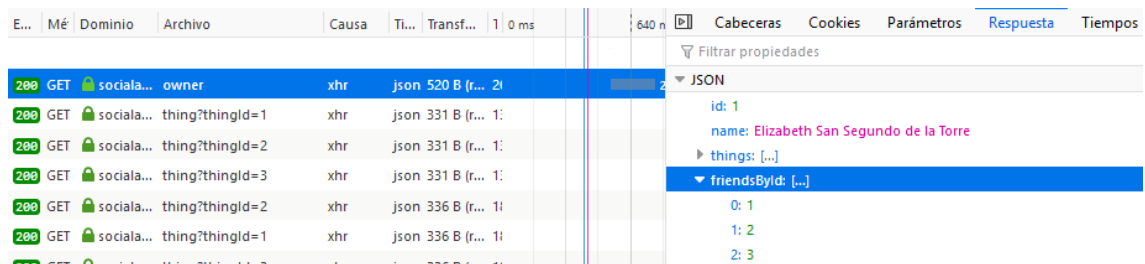


Ilustración 16. Response api/owner

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

La Ilustración 17 es una captura de pantalla donde podemos ver la request 'api/url/provider/thing' para obtener una URL.

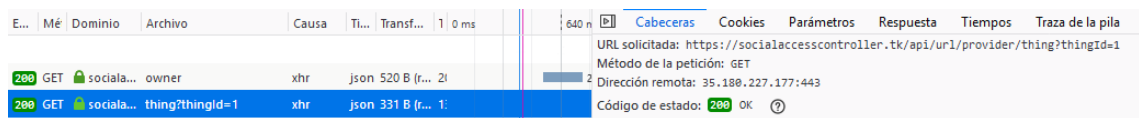


Ilustración 17. Requests api/url/provider/thing.

La Ilustración. 18 es una captura de pantalla donde podemos ver la response: "thing/1".

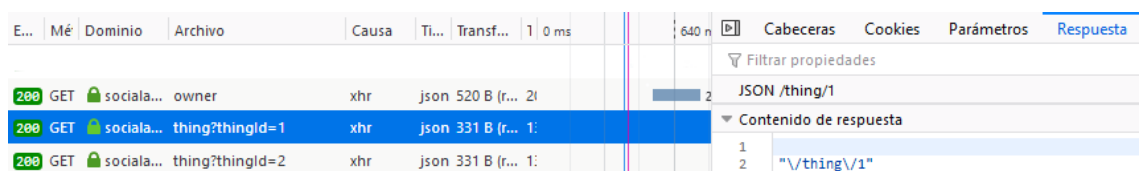


Ilustración. 18 Response api/url/provider/thing.

Esta información será usada como action del botón "Admin". La response corresponde con la ruta definida en fichero de enrutado.

```
src/Infrastructure/Resources/config/routes/thing.yaml.
```

De esta manera hemos conseguido usar sistema de enrutado de Symfony desde el frontend que era la solución buscada.

```
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 a la comunicación anterior, esto es obtener una url. Como funcionan igual y son muy parecidos no vamos a paramos a explicar con mismo nivel de detalle. Es suficiente con saber que la url generada y devuelta es "api/thing/1"

La Ilustración 19 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	socia...	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	socia...	owner	xhr	json	520 B (r...	2!	220 ms		Dirección remota: 35.180.227.177:443			Código de estado: 200 OK			
200	GET	socia...	thing?thingId=1	xhr	json	331 B (r...	1:	118 ms		Versión: HTTP/1.1			Política de referencia: no-referrer-when-downgrade			
200	GET	socia...	thing?thingId=2	xhr	json	331 B (r...	1:	116 ms								
200	GET	socia...	thing?thingId=3	xhr	json	331 B (r...	1:	108 ms								
200	GET	socia...	thing?thingId=2	xhr	json	336 B (r...	1!	103 ms								
200	GET	socia...	thing?thingId=1	xhr	json	336 B (r...	1!	115 ms								

Ilustración 19. Requests url/provider/api/thing.

La Ilustración 20 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	socia...	c34a06	xhr	ht...	4,03 KB	1!	126 ms								
200	GET	socia...	owner	xhr	json	520 B (r...	2!	220 ms								
200	GET	socia...	thing?thingId=1	xhr	json	331 B (r...	1:	118 ms								
200	GET	socia...	thing?thingId=2	xhr	json	331 B (r...	1:	116 ms								
200	GET	socia...	thing?thingId=3	xhr	json	331 B (r...	1:	108 ms								
200	GET	socia...	thing?thingId=2	xhr	json	336 B (r...	1!	103 ms								
200	GET	socia...	thing?thingId=1	xhr	json	336 B (r...	1!	115 ms								

Ilustración 20. Response url/provider/api/thing

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

La Ilustración 21 es una captura de pantalla donde podemos ver la request “api/thing/1”. Nótese que la URL de este request corresponde con el response anterior. 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	socia...	owner	xhr	json	520 B (r...	2!	220 ms		URL solicitada: https://socialaccesscontroller.tk/api/thing/1			Método de la petición: GET			
200	GET	socia...	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	socia...	thing?thingId=2	xhr	json	331 B (r...	1:	116 ms		Versión: HTTP/1.1			Política de referencia: no-referrer-when-downgrade			
200	GET	socia...	thing?thingId=3	xhr	json	331 B (r...	1:	108 ms								
200	GET	socia...	thing?thingId=2	xhr	json	336 B (r...	1!	103 ms								
200	GET	socia...	thing?thingId=1	xhr	json	336 B (r...	1!	115 ms								
200	GET	socia...	thing?thingId=3	xhr	json	336 B (r...	1!	107 ms								
200	GET	socia...	2	xhr	json	596 B (r...	2'	438 ms								
200	GET	socia...	1	xhr	json	508 B (r...	1!	433 ms								

Ilustración 21. Request api/thing/1

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" 2001 162
"_" "c_"
```

Response del lot_emulator

Lo que aquí ocurre recae totalmente en el backend del lot_emulator en concreto en en endpoint GET /thing/{id} visto en 20.

Respuesta Ajax para rellenar frontend

La respuesta del lot_emulador la serializamos en

App\Infraestructura\Controllers\Api\ThingApiController

En la Ilustración 22 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	socia...	owner	xhr	json	520 B (r...	21	220 ms				
200	GET	socia...	thing?thingId=1	xhr	json	331 B (r...	1:	118 ms				
200	GET	socia...	thing?thingId=2	xhr	json	331 B (r...	1:	116 ms				
200	GET	socia...	thing?thingId=3	xhr	json	331 B (r...	1:	108 ms				
200	GET	socia...	thing?thingId=2	xhr	json	336 B (r...	1:	103 ms				
200	GET	socia...	thing?thingId=1	xhr	json	336 B (r...	1:	115 ms				
200	GET	socia...	thing?thingId=3	xhr	json	336 B (r...	1:	107 ms				
200	GET	socia...	2	xhr	json	596 B (r...	2:	438 ms				
200	GET	socia...	1	xhr	json	508 B (r...	1:	433 ms				
200	GET	socia...	3	xhr	json	552 B (r...	2:	431 ms				

JSON

```
status: true
message: null
data: {
  id: 1
  name: thing_name1
  brand: thing_brand1
}
links: {
  actions: {
    link: /actions
  }
  resources: {
    action_name1: {
      values: property_value1
    }
  }
}
```

Ilustración 22. Response api/thing/1

ESQUEMA BASE DE DATOS SAC

En la Ilustración 23 mostramos el esquema usado en SAC. En ella se ven las tablas y campos usados. Vamos a hacer un repaso de los datos que almacenamos. También explicaremos varios los campos

- fb_delegated de tabla owner y tabla friend.
- El campo root de tabla thing.
- El campo name de la tabla Action.

También la existencia de relación N:M entre tablas owners y things. El usuario para acceder a esta base de datos debe tener esto permisos: DDL ALTER, DML SELECT, INSERT, UPDATE, DELETE.

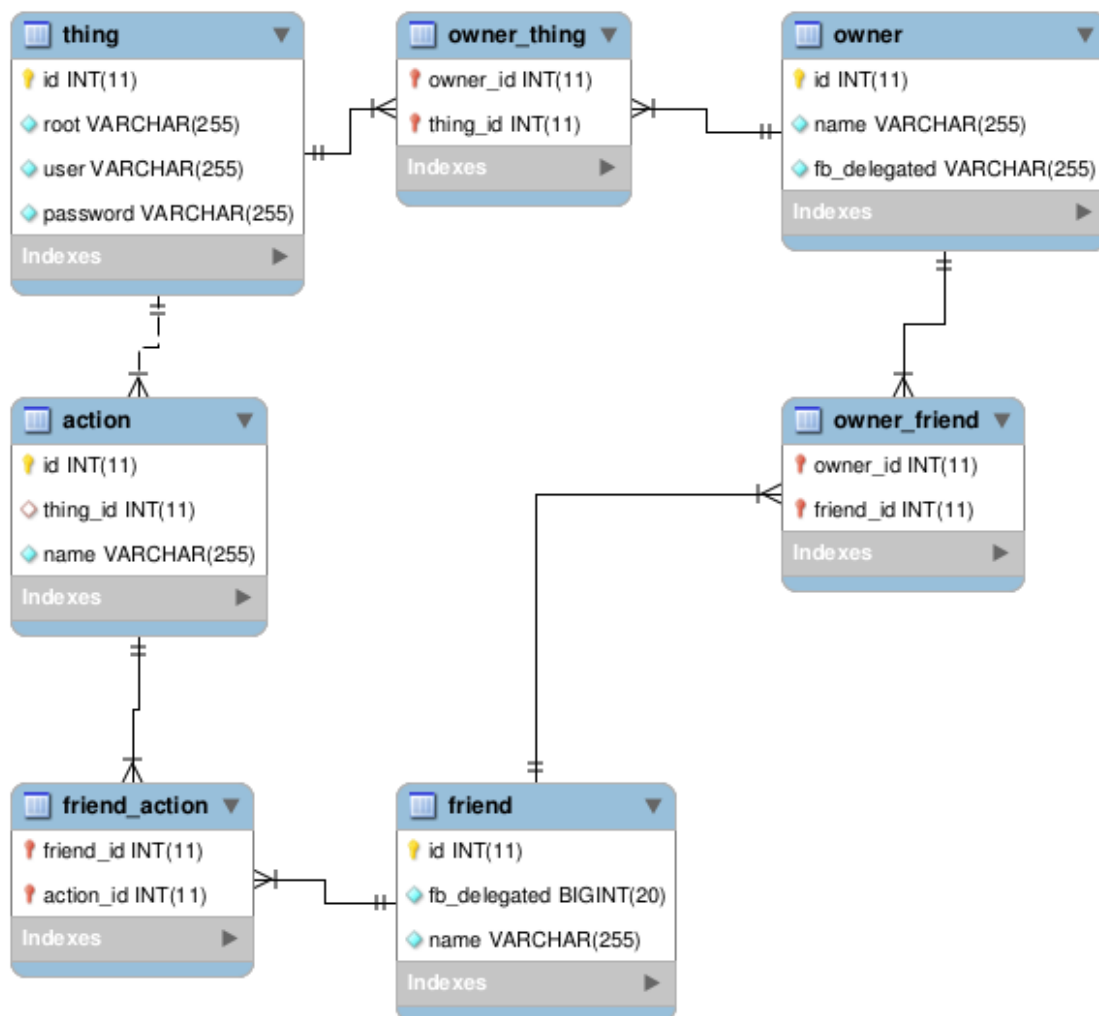


Ilustración 23. Esquema de SAC.

Datos almacenados en Base de Datos de SAC

Los campos almacenados en SAC obedecen al cumplimiento del requisito de almacenar la mínima cantidad de información por parte de SAC.

Para Owner almacenamos el token de Facebook y su nombre. Para Friend lo mismo, el token de Facebook y su nombre. Para Thing el root que lo explicamos en 39 sus credenciales; usuario y contraseña. En Action guardamos el nombre en siguiente apartado vemos por qué.

Explicación Tabla Action: campo Name

Usamos el valor de la tabla action y campo name como nexo de unión para Action entre la información del SAC e lot_emulator. Resolvemos de esta manera esta unión: los valores almacenados SAC deben concordar con el valor de respuesta del lot_emulator al nivel marcado en siguiente json.

```

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

```

Explicación fb_delegated en tablas owner y friend

Estos campos tienen su utilidad durante el proceso de logeo del usuario y en concreto con la Autenticación Delegada 29.

Explicación Tabla thing: campo root

Este campo es usado por API SAC para traer los datos actualizados de los WTs. El campo root es el endpoint al que dispara API SAC para obtener los datos actualizados.

Explicación Relación N:M entre owners y things

Mientras que la base de datos está preparada para que SAC pueda soportar múltiples Owners, relación N:M entre Owner y Thing El caso de uso diseñado para la creación del Owner provoca hace inviable múltiples Owners por lo que una relación 1:N entre Owner y Thing hubiera sido suficiente. Hemos preferido estabilizar las diferentes funcionalidades ya implementadas y de cara a una facilitar una ampliación del proyecto dejar los esquemas con estas relaciones mas grandes. En el apartado de mejoras aparece esta (60).

ARQUITECTURA REST SAC DEL API SAC

Vamos a explicar los endpoints relacionados con el sac.

Verbo HTTP	Endpoint	Descripción
GET	/api/owner	
GET	/api/thing/{thingId}	
GET	/api/url/provider/thing	
GET	/api/url/provider/api/thing	
GET	/api/url/provider/api/share/action	

Endpoints

GET /api/owner

Devuelve un el valor de Owner serializado.

GET /api/thing/{thingId}

Desencadena la llamada al lot_emulator y serializa la respuesta. Relacionado con los explicado en 49

GET/api/url/provider/thing

Genera una url.

GET /api/url/provider/api/thing

Genera una url.

GET /api/url/provider/api/share/action

Genera una url.

ARQUITECTURA HEXAGONAL SAC

Vamos a explicar de qué manera se ha hecho una arquitectura hexagonal. Para ello, vamos a hacer un repaso a las capas Dominio, Aplicación e Infraestructura vista en Diseño de Software. Arquitectura Hexagonal.

Dominio SAC

Explicación de esta capa en pág 6.

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

// Repositorios
src/Domain/Repository/OwnerRepository.php
src/Domain/Repository/ThingConnectedRepository.php
src/Domain/Repository/ThingRepository.php
src/Domain/Repository/ActionRepository.php
src/Domain/Repository/FriendRepository.php
```

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

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

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

Aplicación SAC

Explicación de esta capa en pág 6.

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

Commands SAC

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

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

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

// Comandos para Thing
src/Application/Command/Thing/CreateThingCommand.php
src/Application/Command/Thing/GetActionsByThingIdCommand.php
src/Application/Command/Thing/GetThingConnectedInfoCommand.php
src/Application/Command/Thing/MergeThingWithThingConnectedByIdCommand.php
src/Application/Command/Thing/SearchThingByIdCommand.php

// Comandos para ThingConnected
src/Application/Command/Thing/ThingConnected/GetThingConnectedCompleteById
Command.php
```

CommandHandlers SAC

Vamos a centrarnos en los CommandHandlers que requieren más atención, que son: GetFbSharingStatusByOwnerCommandHandler, SearchThingConnectedActionsHandler, SearchThingConnectedNameHandler, SearchThingConnectedBrandHandler, IsActualUserAnOwnerHandler. Posteriormente en el ejemplo hexagonal veremos GetThingConnectedCompleteHandler

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

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

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

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

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

GetFbSharingStatusByOwnerCommandHandler

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

```
// estructura de shareStatus
[
  1 => array:2 [
    "thingId" => 1
    "actions" => array:2 [
      1 => "103671587486416"
      2 => "104003390786397"
    ]
  ]
]
```

IsActualUserAnOwnerHandler

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

SearchThingConnectedActionsHandler, SearchThingConnectedBrandHandler y SearchThingConnectedNameHandler

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

```
src/Application/Command/Thing/ThingConnected/GetThingConnectedCompleteById  
Command.php
```

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

GetThingConnectedInfoCommandHandler

La explicación para este CommandHandler esta explicada en el ejemplo general hexagonal en 50.

Infraestructura SAC

Explicación de esta capa en (7). Esta capa tiene mucha información pero es información que si bien siendo imprescindible puede ser substituida por otra sin que eso afectase a la funcionalidad del Dominio de la aplicación.

Empezaremos viendo la carpeta `Resources` y las configuraciones contenidas. Explicaremos cómo hemos dividido los controladores. En la parte de Repositorios del SAC veremos como la capa de Infraestructura implementa los contratos (interfaces) ofrecidos por el Dominio. En los Repositorios también terminaremos con el ejemplo iniciado en la parte de Dominio relacionado con `ThingConnected`. En serializadores vemos como Infraestructura se encarga de dar forma al Dominio. Hemos generado mucha cantidad de Comandos SAC tanto para el desarrollo como para pruebas. También veremos la manera en la que hemos restringido el acceso a Controladores enteros usando `Event Subscribers`. Finalmente, aunque estrictamente no esté en capa de Infraestructura al ser un tema muy relacionado con `Symfony` lo comentamos aquí acabaremos hablando de los servicios.

Resources SAC

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

Archivos .yaml con definiciones de rutas

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

```
// 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 (10) en SAC existen fichero .yaml donde cada entidad define tanto las relaciones entre ellas como sus propiedades. Es en estos directorios donde quedan definidas

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

Controladores SAC

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 ya explicado. OwnerApiController es donde SAC consulta información del Owner un ejemplo de uso de endpoint está en 34. Al igual un ejemplo para ThingApiController lo tenemos en 36.

```
// 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 27.

```
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 24 así como endpoints pedidos por Facebook como son /privacy y /conditions. OwnerController se encarga de la parte vista en “Mapa web para Owner” de la 25 del endpoint /owner. FriendController la parte vista en la 27 “Mapa web para Friend” del endpoint /friend. Por ultimo ThingController muestra “Admin de un Wt” 26 en endpoint thing/info también procesa thing/create.

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

Respositorios SAC

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

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

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

Serializadores SAC

```
// Serializador de Action
src/Infrastructure/Action/Serializer/ActionArraySerializer.php

// Serializador de Owner
src/Infrastructure/Owner/Serializer/OwnerArraySerializer.php

// Serializador de Thing
src/Infrastructure/Thing/Serializer/ThingArraySerializer.php
src/Infrastructure/Thing/Serializer/ThingWithThingConnectedArraySerializer
.php

// Serializador de ThingConnected
src/Infrastructure/ThingConnected/Serializer/ThingConnectedSerializer.php
```

Comandos Symfony SAC

Namespace

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

```
src/Infrastructure/Action/Command  
src/Infrastructure/Friend/Command  
src/Infrastructure/Owner/Command  
src/Infrastructure/Thing/Command  
src/Infrastructure/ThingConnected/Command
```

Listado completo

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

```

$ php bin/console
app
  app:Action:Create           Creates an Action
  app:Action:SearchActionById Searches an action By
id
  app:Friend:Create          Creates a Friend
  app:Friend:SearchFriendByFbDelegated given fbDelegated
returns friend
  app:Friend:SearchFriendById given id returns
friend
  app:Owner:AddFriendToOwner Given fbDelegated of
Owner and fbDelegated of Friend will create a relationshing in
owner_friend table
  app:Owner:AddThingToOwner   Given fbDelegated of
Owner and Thing id will create a relationship in owner_thing table
  app:Owner:Create            Add a short
description for your command
  app:Owner:GetFbSharingStatus gets All relationships
between Owner-Friends-Actions
  app:Owner:GetListThingsByOwner Given an fb_delegated
returns list of things
  app:Owner:SearchByfbDelegated Given a fbDelegated
gets Owner
  app:Owner:ShareActionToFriend Given an owner
fbDelegated, a Friend ID and ActionId. Shares Given Action to Friend
  app:Thing:Create            Add a short
description for your command
  app:Thing:GetActionsByThingId Given an thing.id
returns Actions
  app:Thing:MergeThingWithThingConnectedById Given and (int) id
merges sac Thing with ThingConnected
  app:Thing:SearchByThingId   Given a (int) id
searches Thing
  app:ThingConnected:GetThingActionsByThingId Connects to thing and
retrieves name
  app:ThingConnected:GetThingBrandyThingId Connects to thing and
retrieves name
  app:ThingConnected:GetThingConnectedCompleteById Connects to thing and
retrieves All ThingConnected Info
  app:ThingConnected:GetThingNameByThingId Connects to thing and
retrieves name

```

A modo de ejemplo ponemos el resultado de lanzar este comando
app:ThingConnected:GetThingConnectedCompleteByld

```

$ php bin/console app:ThingConnected:GetThingConnectedCompleteById 3
array:3 [
    "message" => ""
    "status" => true
    "data" => {#352
        +"id": 4
        +"name": "thing_name4"
        +"brand": "thing_brand4"
        +"links": {#342
            +"actions": {#360
                +"link": "/actions"
            }
            +"resources": {#327
                +"action_name1": {#358
                    +"values": "property_value1"
                }
                +"action_name2": {#349
                    +"values": "property_value2"
                }
                +"action_name3": {#346
                    +"values": "property_value3"
                }
                +"action_name4": {#347
                    +"values": "property_value4"
                }
            }
        }
    }
}
]

```

Event Subscribers SAC

Para poder proteger zonas que necesiten de autenticación Facebook hemos usado los Eventos de Kernel de Symfony. Esta sección es única para SAC. Es una manera elegante de controlar el acceso a distintas zonas.

```

// EventSubscriber
src/Infrastructure/EventSubscriber/HasFbSessionSubscriber.php

```

Aquellos controladores que lo implementen como por ejemplo

```

class ThingApiController extends AbstractController implements
HasFbSessionController

```

Hace que antes que el controlador genere response se ejecute este código

```

if ($controller[0] instanceof HasFbSessionController) {
    $request = $event->getRequest();
    $session = $request->getSession();
    if ($session->get('ownerFbDelegated') === null) {
        $event->setController(
            function(){
                return new Response('Resource Not Found', 404);
            }
        );
    }
}
}

```


Donde se comprueba si existe en sesión la variable 'ownerFbDelegated' en caso contrario se muestra hn "Resource Not Found"

Services Symfony

Symfony tiene un Dependency Injector Container (DIC) que permite inyectar clases al constructor sin tener que instanciarlas. Mostramos la definición del servicio sacado del fichero configuración de servicios:

```
Config/service.yaml
```

Vemos como creamos un servicio llamado `app.command_handler.thing.merge_thing_with_thing_connected` que a su vez recibe dos servicios como argumento: `app.command_handler.thing.search_by_id` y `app.command_handler.thing_connected.get_complete`

```
app.command_handler.thing.merge_thing_with_thing_connected:
  class:
    App\Application\CommandHandler\Thing\MergeThingWithThingConnectedByIdHandler
  public: true
  arguments:
    - "@app.command_handler.thing.search_by_id"
    - "@app.command_handler.thing_connected.get_complete"
```

Ahora vemos el ejemplo del constructor del Controlador para API SAC `ThingApiController` y como Symfony es capaz de construirle el `CommandHandler MergeThingWithThingConnectedByIdHandler`.

```
public function __construct(MergeThingWithThingConnectedByIdHandler
$mergeThingWithThingConnectedByIdHandler)
```

Ejemplo hexagonal: ejemplo de ThingConnected.

Al centrarnos en este ejemplo vemos que la Entidad de Dominio `Thing` tiene una propiedad llamada `thingConnected` que se rellena con los datos del `lot_emulator`. También en Dominio existe el Repositorio `ThingConnectedRepository`. En la capa de Aplicación existen varios `CommandHandlers` que hacen consultas al `lot_emulator` usando la implementación concreta en los Repositorios de Infraestructura que cumplen el contrato establecido en Dominio.

En esta sección vamos a ver una manera hexagonal de resolver como traer datos del `lot_emulator`.

ThingConnected en capa Dominio

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

```
src/Domain/Repository/ThingConnectedRepository.php
```

`ThingConnectedRepository` define una interfaz de los métodos a implementar para obtener los datos del WT del `lot_emulator`.

ThingConnected en capa Aplicación

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

```
// Command
src/Application/Command/Thing/ThingConnected/GetThingConnectedInfoCommand

//CommandHandler
src/Application/CommandHandler/Thing/ThingConnected/GetThingConnectedCompleteHandler
```

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

```
public function __construct(ThingConnectedRepository
$thingConnectedRepository,
```

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

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

El `CommandHandler` está orquestando en cuanto que provee a métodos de la información que precisan para ejecutarlos. Está poniendo en marcha el Repositorio de Dominio nutriendolo con el Repositorio de Infraestructura.

ThingConnected en capa Infraestructura

En la esta sección vemos de qué manera el Repositorio de Dominio implementa el Repositorio de Dominio Acabando.

```
src/Infrastructure/ThingConnected/CurlThingConnectedRepository.php.
```

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

```
private function sendCurlOrException($thingRoot, $thingUserName,
$thingPassword)
{
    $ch = curl_init($this->iotEmulatorHost . '/' . $thingRoot);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");
    curl_setopt($ch, CURLOPT_PORT, $this->iotEmulatorPort);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array(
        'user: ' . $thingUserName,
        'password: ' . $thingPassword
```

```
    )  
    );  
$json = curl_exec($ch);
```

SEGURIDAD SAC

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

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

TECNOLOGÍAS USADAS

PHPSTORM

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

- PHP
- HTML
- MySQL
- Doctrine
- Javascript
- Twig
- Symfony
- Cliente HTTP

Así como ayudas que ofrece mejorar el código:

- PSR
- Creación de servicios symfony
- Búsqueda inteligente de:
 - Definiciones de métodos
 - Implementaciones de interfaces

FACEBOOK

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

Facebook API

- Autenticación delegada.
- Consulta de lista de amigos.

Facebook developers

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

SISTEMA OPERATIVO

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

variable de entorno

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

sshfs

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

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

alias

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

httpie

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

```
/tests/requests
```

jq

Procesador json por terminal, usado para mostrar respuestas curl o buscar ciertos claves o valores en respuestas. Muy útil durante el desarrollo de APIs.

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/lot_emulator

AWS EC2

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

NGINX

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

Configuración lot_emulator

```
# Virtual Host configuration for iot.socialaccesscontroller.tk
#
server {
    listen 80;
    listen [::]:80;

    server_name iot.socialaccesscontroller.tk;

    root /var/www/Iot_emulator/public;
    index index.php;

    error_log /var/log/nginx/iot.socialaccesscontroller_error.tk debug
    access_log /var/log/nginx/iot.socialaccesscontroller_access

    location / {
        try_files $uri /index.php$is_args$args;
```

```
}  
  
location ~ /\.php {  
    try_files $uri =404;  
    fastcgi_split_path_info ^(.+\.php)(/.+)$;  
    include fastcgi_params;  
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;  
    fastcgi_param SCRIPT_NAME $fastcgi_script_name;  
    fastcgi_index index.php;  
    fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;  
}  
}
```

Configuración SAC

```
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    include snippets/snakeoil.conf;
    index index.php;
    error_log /var/log/nginx/socialaccesscontroller.error.log;
    access_log /var/log/nginx/socialaccesscontroller.access.log;

    root /var/www/sac/public;
    server_name socialaccesscontroller.tk;
    location / {
        try_files $uri /index.php$is_args$args;
    }

    location ~ /\.php {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        fastcgi_param SCRIPT_NAME $fastcgi_script_name;
        fastcgi_index index.php;
        fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
    }
}
```

HTML5, CSS, BOOTSTRAP Y MOUSTACHE

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

JAVASCRIPT, JQUERY Y JQUERY-UI, TWIG

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

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

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

Twig es un sistema de template integrado en Symfony. Las vistas del SAC se generan con Twig.

SYMFONY 4

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

NPM

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

MYSQL Y DOCTRINE

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

Doctrine es un ORM muy integrado en Symfony

MySQLWorkbench es como hemos construido los esquemas en esta aplicación y hecho ingeniería inversa para mostrar las ilustraciones.

COMPOSER

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

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

ANÁLISIS RESULTADOS

Se ha conseguido crear un entorno seguro donde un Owner pueda de manera segura y sencilla elegir qué Actions compartir y a qué Friend compartirlas. Esto lo conseguimos almacenando de manera segura en SAC las credenciales para los permisos de acceso a los WTs y obteniendo la estructura de relaciones sociales directamente de Facebook.

Usando SAC el Owner no tiene por qué compartir sus credenciales a Friends tampoco necesita buscar WTs multiusuarios. SAC funciona como intermediario entre la información de un WT compartido con Friend de Facebook. Pudiendo hacerse con WTs monousuarios y no cediendo las credenciales. Además, algo único del SAC es la capacidad de compartir a nivel de Action y no a nivel superior de WT.

RELACIÓN CON ASIGNATURAS DEL MÁSTER

Entorno Web

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

```
CI/amazon-provision.sh
```

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

```
CI/deploy/playbook.yml
CI/deploy/Vagrantfile
```

Por otro lado, el uso de git es básico aunque sea un desarrollo hecho por una persona. Nos ha servido para tener un repositorio de código y poder montar los proyectos en distinto sitios. Sobre todo, al principio de los proyectos hemos hecho mucho usos de lo aprendido con ramas.

Bases de Datos

Hemos usado los conocimientos aprendidos en asignatura Bases de Datos a la hora de construir los esquemas relacionales de tal manera que sean correctos con lo que buscamos. Estos conocimientos también son necesarios al trabajar con Doctrine para anticipar que datos deben devolverte los distintos métodos.

Maquetación web

Este Trabajo Fin de Máster no tiene una alta carga de trabajo en frontend. Aún así nos hemos esforzado por usar bibliotecas de mucho uso en la web para dar un aspecto de modernidad. Tal como vimos en la asignatura hemos maquetado usando un grid de doce columnas.

PHP

Toda la lógica de negocio está programada en PHP. Hemos usado muchas cosas aprendidas en la asignatura de PHP. Como los parámetros y returns tipados de los métodos o funciones. El uso de PSR y su integración con namespace para usar autoloader. Esto último ha sido muy útil para el código no solo del Trabajo Fin de máster.

Frameworks

La asignatura de Framework y la de PHP son seguro las asignaturas que más hemos reforzado haciendo los proyectos. Si bien aún se puede avanzar e implementar cosas vistas como i18n hemos aplicado otros muchos conceptos vistos en asignatura de Frameworks como el uso de servicios, los Comandos, Event Subscribers, enrutamiento. Incluso algunas no vistas como Fixtures.

Desarrollo Eficiente

Hemos usado mucho tiempo en revisar el código tanto a nivel funcional como semántico. Hemos aplicado dos Arquitecturas hexagonales, haciendo hincapié en el SAC frente al lot_emulator.

Centrándonos en la asignatura hemos hecho uso de patrones de diseño como DTO o Inyección de dependencias. También hemos aplicado los principios SOLID a la hora de programar y el lenguaje ubicuo poniendo nombres semánticos que describan lo que hacen.

Al principio del desarrollo hasta que el código tuvo cierta solidez hicimos varias iteraciones de Refactoring, por ejemplo al decidir usar un DTO específico en lot_emultao para las credenciales.

Aunque la cobertura de tests es poca sí que hemos usado PHPUnit en lot_emulator donde también hemos diseñado tests sin este framework. En ambos proyectos existen tests, no automáticos, diseñados unos para funcionar y otros para fallar mediante requests ya construidas.

Rendimiento

Hemos procurado hacer un front liviano que sea capaz de cargar varios WTs simultáneamente.

SEO

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

Seguridad

En la asignatura de Seguridad aprendimos a no fiarnos del input del usuario. Antes de guardar en base de datos la arquitectura hexagonal lanza una Excepción si algún dato no es correcto.

Emprenduría

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

CONCLUSIÓN

Internet of Things (IoT) es la agrupación e interconexión de dispositivos y objetos a través de una red. Estos objetos físicos conectados a la red tienen una representación virtual llamada Web Things (WT).

En este trabajo hemos implementado un Social Access Controller (SAC) poniendo en práctica una manera segura de compartir las contrapartes virtuales de los dispositivos físicos WT entre amigos. Hemos podido usar Facebook para manejar el acceso a WT basado en Facebook. La idea del modelado de WT la hemos de W3Consortium y la idea de cómo compartir y crear el SAC de [61].

Este Trabajo Fin de Máster es, sin lugar a dudas, la pieza de código más grande emprendida por mí jamás. He trabajado en código muy grande y complejo, pero siempre lo he heredado de otros este lo he hecho desde cero. Todo el diseño de esquema de base de datos, diseño de API, diseño del frontend, diseño de las clases, documentación, etc... poder sacar esto teniendo que trabajar y tener una familia ha supuesto un esfuerzo descomunal. He confirmado cosas que ya sabía, me gusta el backend y me gusta el trabajo de sistemas.

TRABAJO FUTURO

Multi-owner

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

Cacheado inteligente de datos de cada WT

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

Proponemos que aquellos que 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.

ANALISIS ECONOMICO DEL PROYECTO

Diseño

70 horas

Implementación

200 horas

Pruebas

40 horas

Memoría

80 horas

GLOSARIO

- **Web Thing (WT)**. Objeto con conexión a internet que ofrece información interna vía http(s) y arquitectura REST. En el presente proyecto WT devuelven formato de datos JSON. Posee dos zonas:
 1. Zona Pública
 - Nombre
 - Brand
 2. Zona Privada
 - Action
 - Property
- **WT Brand (Brand)**: Es la compañía que ha construido el objeto físico
- **WT Name (Nombre)**: Es el nombre que recibe el objeto físico, sirve para que diferenciarlo entre ellos.
- **Property**: Es una propiedad de un WT representa un estado interno.
- **Action**: Es una interacción 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.

BIBLIOGRAFÍA

1. W3Consortium WOT Model <https://www.w3.org/Submission/2015/SUBM-wot-model-20150824/>
2. "Sharing Using Social Networks in a Composable Web of Things" <https://vs.inf.ethz.ch/publ/papers/dguinard-sharin-2010.pdf>