

Índice

- Glosario
- Resumen

Glosario

- **Web Thing (WT)** objeto físico conectados a la red y disponibles vía web llamados Web Things
- **Internet of Things (IoT)** agrupación e interconexión de dispositivos y objetos a través de una red
- **Owner** (Dueño) persona que posee las credenciales a un WT
- **Friend** (Amigo) persona de la confianza del Owner

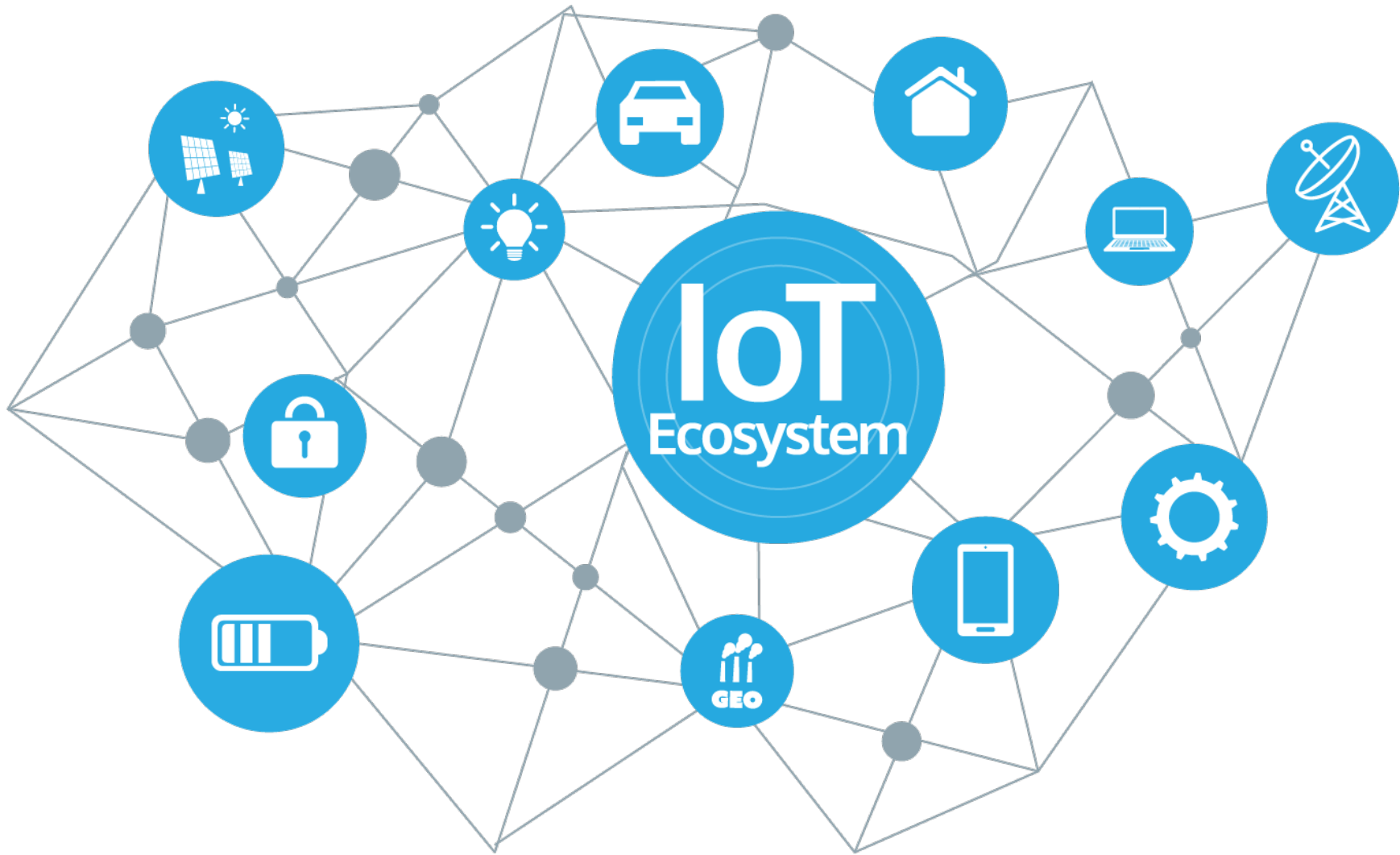
Web Thing (WT)

- Objeto Físico
- Representación Virtual



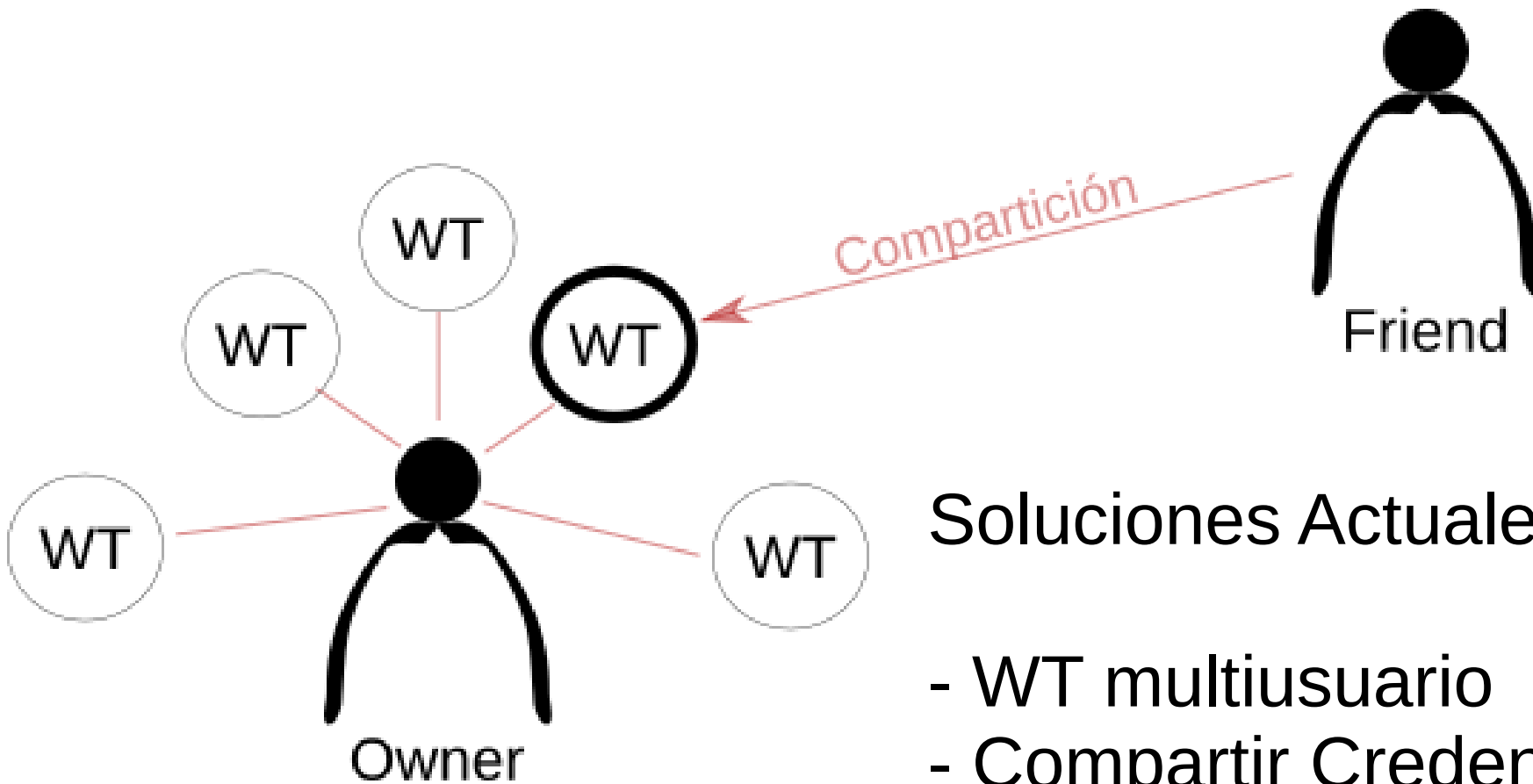
```
{  
  "id": 1,  
  "name": "cafetera plus R10",  
  "brand": "Acme",  
  "links": {  
    "actions": {  
      "link": "/actions",  
      "resources": {  
        "calentar_agua": {  
          "values": "0"  
        }  
      }  
    }  
  }  
}
```

Internet of Things (IoT)



Motivación

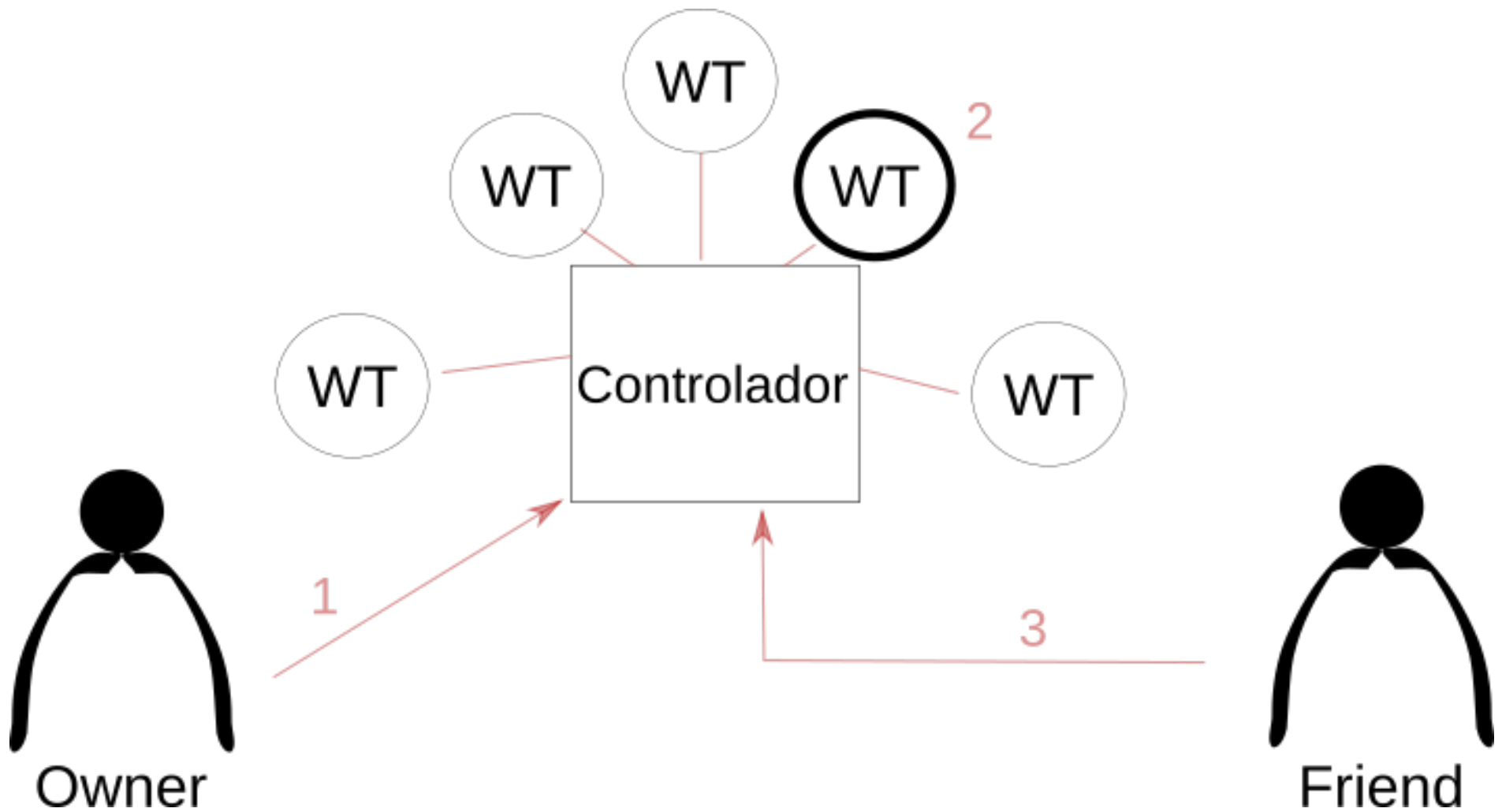
Owner quiere compartir un WT con un Friend



Soluciones Actuales:

- WT multiusuario
- Compartir Credenciales

Solución



Modelos Usados

1) W3Consortium

Como deben ser los WT

W3Consortium



Web Thing Model

W3C Member Submission 24 August 2015

2) Dominique Guinard

Como debe ser el Controlador

Guinard

Sharing Using Social Networks
in a Composable Web of Things

Dominique Guinard, Mathias Fischer, Vlad Trifa
Institute for Pervasive Computing, ETH Zurich
and SAP Research CEC Zurich
8092 Zurich, Switzerland
Contact Email: dguinard@ethz.ch

Solución

- Controlador para compartir WT

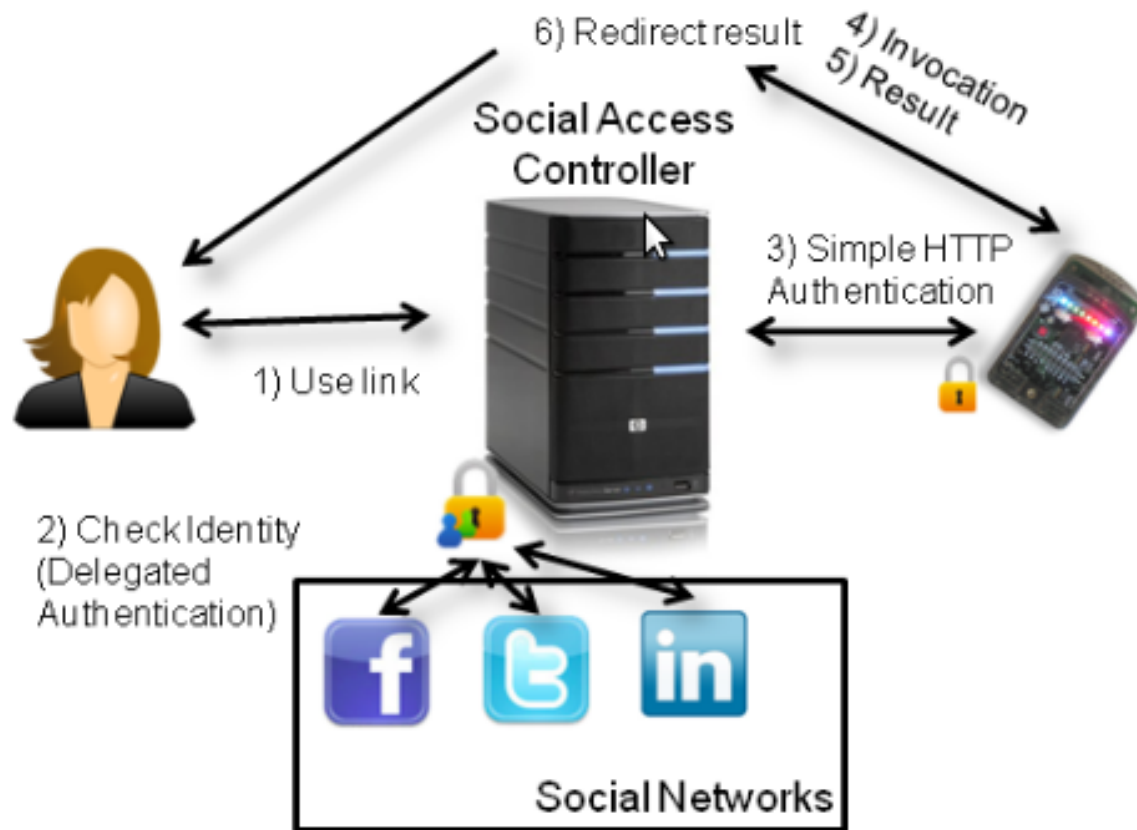


Fig. 3. Accessing shared objects using delegated authentication through the Social Access Controller.

Ajuste con modelo 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

Ajustamos a un 60% a Nivel DEBERÍA
Ajustamos al 0% en Nivel PODRÍA

Ajuste modelo Guinard

Los requisitos que no hemos implementado

- Independencia de Red social usada
- Descubrimiento de Wts automática
- Discernir verbo HTTP compartido

Simplificaciones

Un único Owner

Action

- Relación Action y Property
- Action name como nexa unión entre Sac e
lot_Emulator

Pk id = identificador de WT

Tecnologías Software Backend



Symfony 4

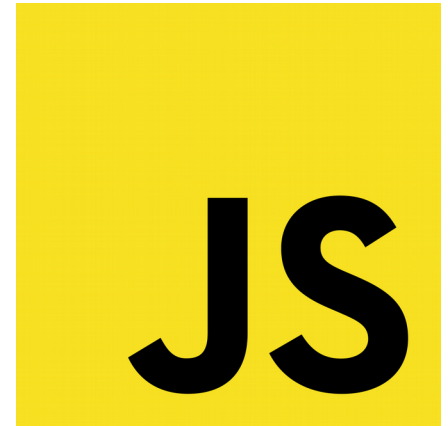


Tecnologías Software Frontend

HTML



CSS



Servidores Producción y Red

Dominio y SubDominio configurados

Proyecto	URL
SAC	https://socialaccesscontroller.tk
lot_emulator	http://iot.socialaccesscontroller.tk



Amazon EC2



Tecnologías dependencias y Entorno Desarrollo

Entorno Desarrollo



ANSIBLE



Tecnologías dependencias

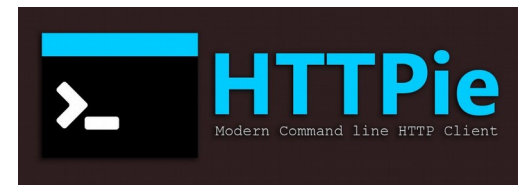


Herramientas desarrollo



curl://

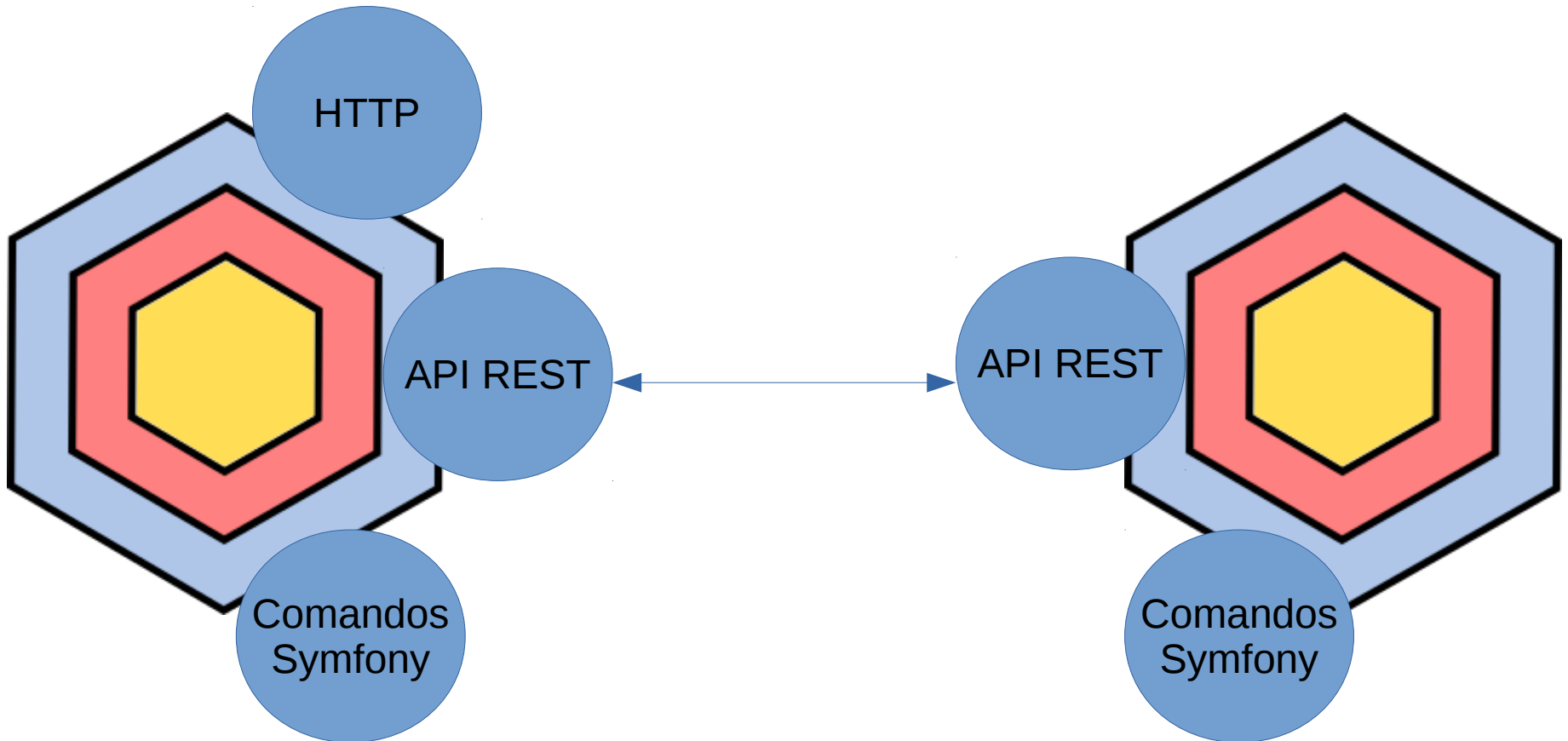
./jq



Arquitectura Software

SAC

lot_emulator



Dominio SAC

// 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

Aplicación SAC

```
// Comandos 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

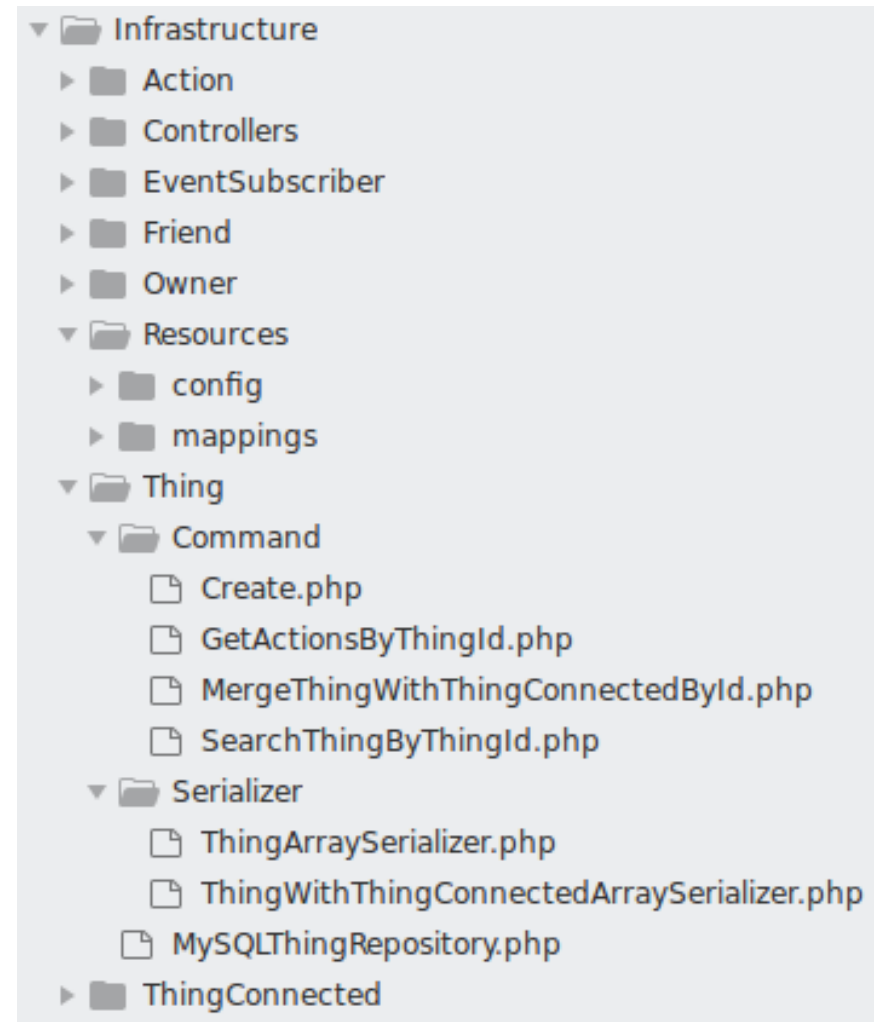
// Comandos para Thing
src/Application/Command/Thing/CreateThingCommand.php
src/Application/Command/Thing/GetActionsByThingIdCommand.php
src/Application/Command/Thing/GetThingConnectedInfoCommand.php
src/Application/Command/Thing/MergeThingWithThingConnectedByIdCommand.php
src/Application/Command/Thing/SearchThingByIdCommand.php

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

CommandHandlers son muy similares

Infraestructura SAC

- Resources
 - config/routes
 - mappings
- Controladores
- Repositorios
- Serializadores
- Comandos Symfony
- Event Subscriber

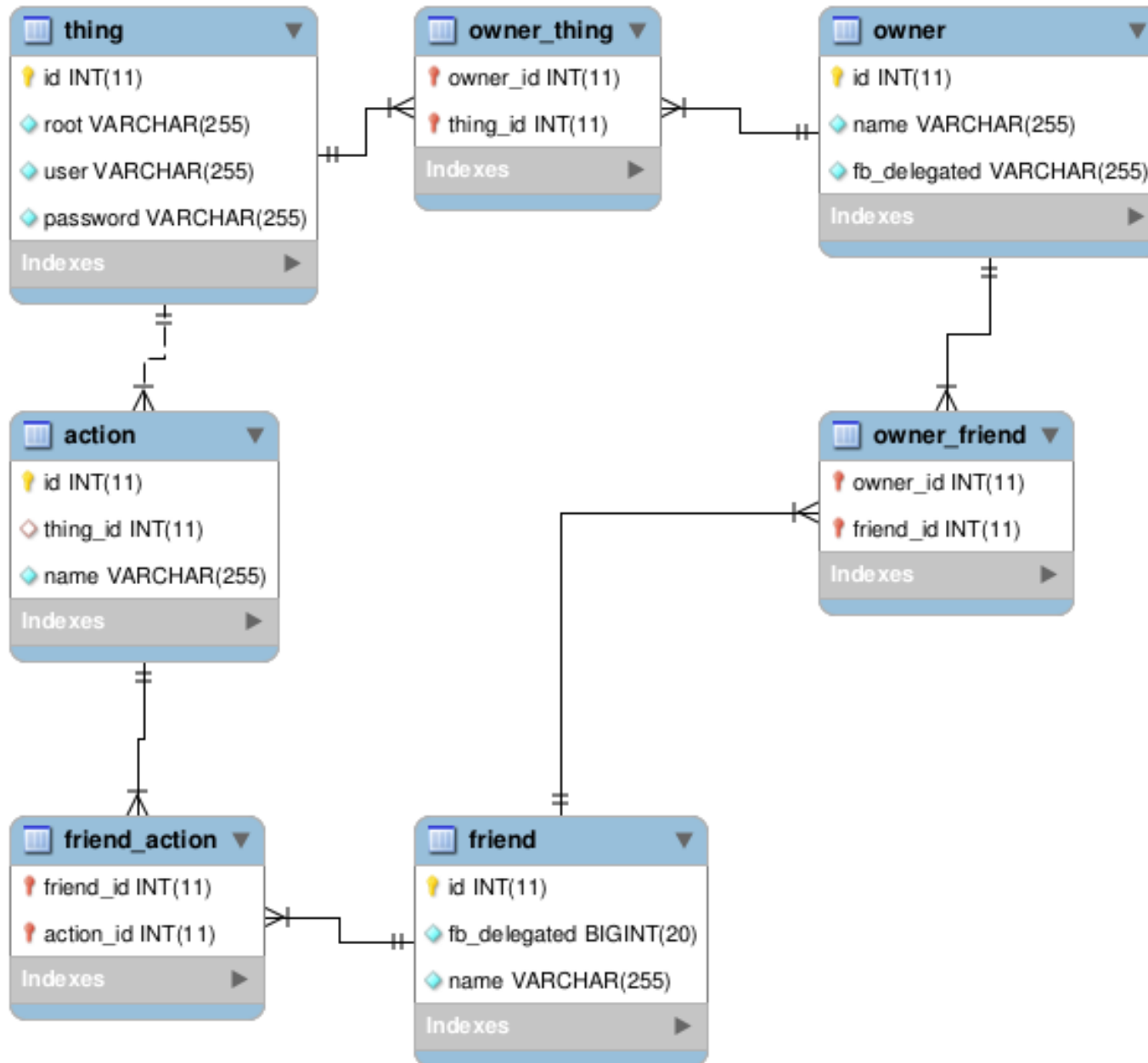


Thing desplegada. Action, Friend, Owner tienen estructura similar

REST API SAC

Verbo HTTP	Endpoint
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

Esquema Base de Datos SAC



Dominio lot_emulator

```
// 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
```

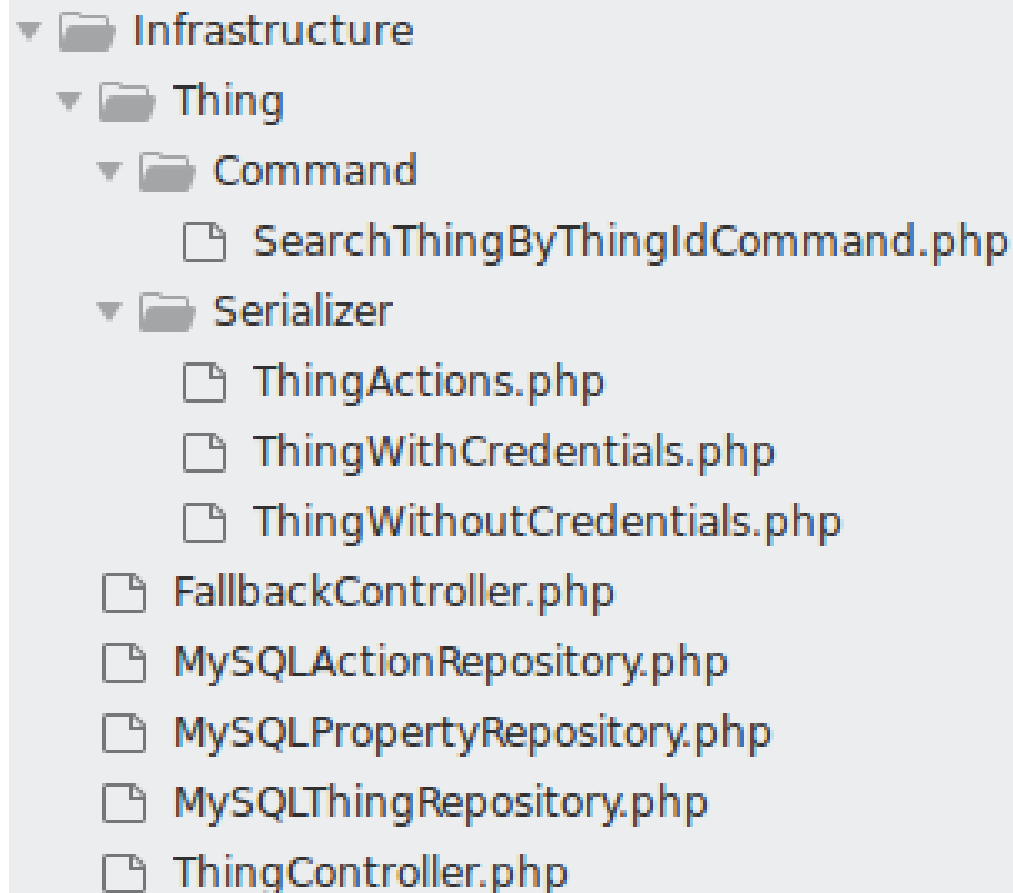
```
src/Domain/Repository/UserRepository.php
```

Aplicación lot_emulator

```
// Commands  
src/Application/Command/Thing/CreateThingCommand.php  
src/Application/Command/Thing/ExecuteActionCommand.php  
src/Application/Command/Thing/SearchThingByIdCommmand.php  
  
// CommandHandlers  
src/Application/CommandHandler/Thing/CreateThingHandler.php  
src/Application/CommandHandler/Thing/ExecuteActionHandler.php  
src/Application/CommandHandler/Thing/SearchThingByIdHandler.php
```

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


Infraestructura lot_emulator



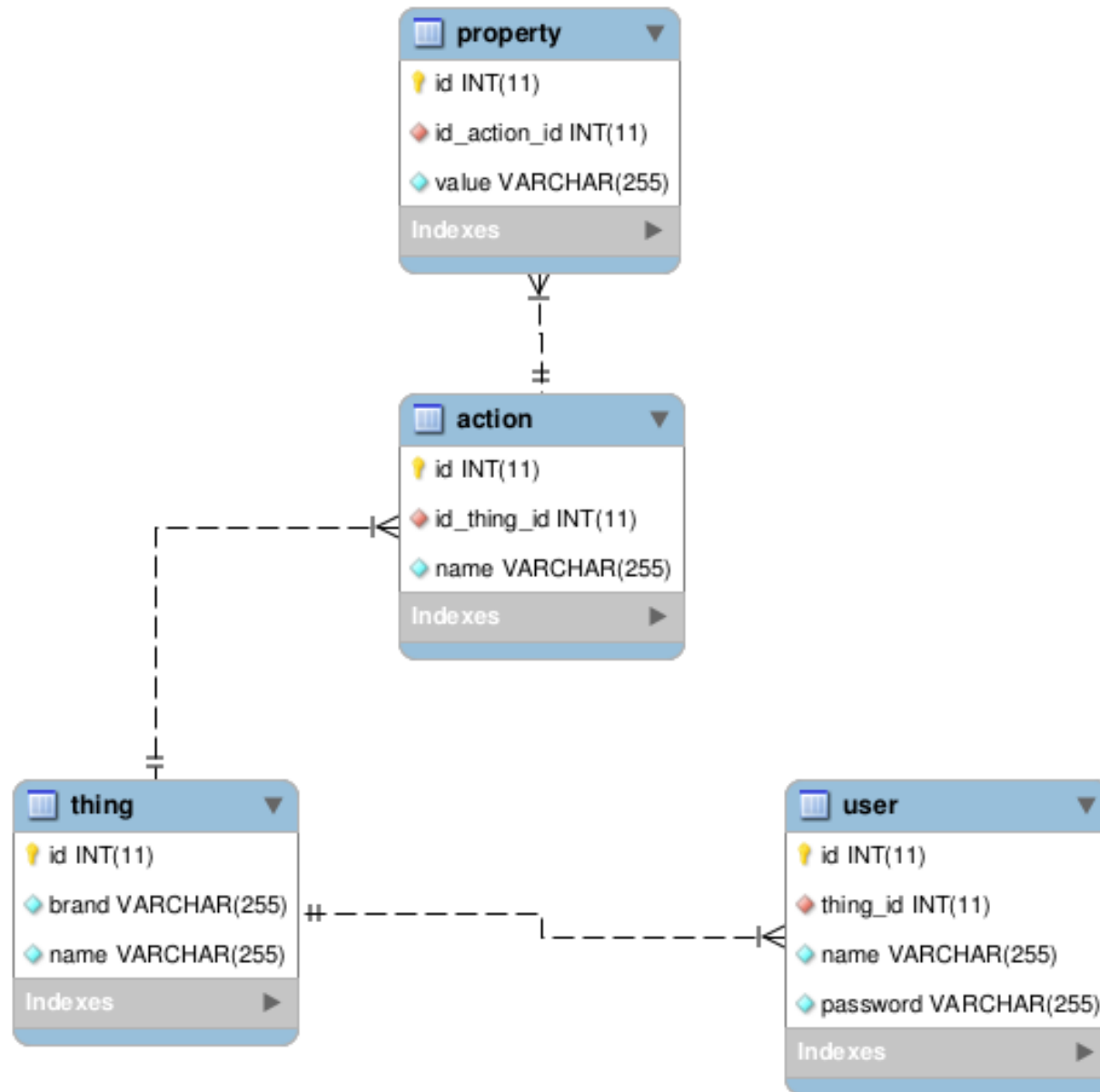
```
▼ Infrastructure
  ▼ Thing
    ▼ Command
      SearchThingByThingIdCommand.php
    ▼ Serializer
      ThingActions.php
      ThingWithCredentials.php
      ThingWithoutCredentials.php
  FallbackController.php
  MySQLActionRepository.php
  MySQLPropertyRepository.php
  MySQLThingRepository.php
  ThingController.php
```

The image shows a file explorer view of a project structure. The root directory is 'Infraestructura'. Inside it, there is a 'Thing' directory. The 'Thing' directory contains two sub-directories: 'Command' and 'Serializer'. The 'Command' directory contains a single file 'SearchThingByThingIdCommand.php'. The 'Serializer' directory contains three files: 'ThingActions.php', 'ThingWithCredentials.php', and 'ThingWithoutCredentials.php'. Outside the 'Thing' directory, there are five more files: 'FallbackController.php', 'MySQLActionRepository.php', 'MySQLPropertyRepository.php', 'MySQLThingRepository.php', and 'ThingController.php'.

Rest lot_emulator

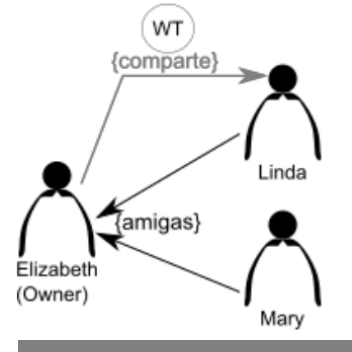
Verbo HTTP	Endpoint
GET	/
GET	/ {id} (Sin credenciales)
GET	/ {id} (Con credenciales)
POST	/create
POST	/ {id}/actions/{action_name}
GET	/ {id}/properties/{property_name}
GET	/ {id}/actions

Esquema Base de datos lot_emulator



Datos de Pruebas (fixtures)

Facebook



lot_emulator

```
{ "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" } ] } }
```

SAC



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

WHERE IS?

- Cache/Redis