UNIVERSIDAD CAECE - DEPARTAMENTO DE SISTEMAS

Práctica profesional supervisada

Carrera:	Ingeniería en Sistemas		
Responsable:	Grippo, Marcelo		
Integrantes:	Di Maggio, Mariano Quintana, Gabriel		
Año:	2017	**Cuatrimestre:**	2

Título del proyecto:	Cerradura inteligente
Fecha de entrega:	

Firma del Profesor:	

Contenido

Introducción

El proyecto consiste en la implementación de una cerradura electrónica activada mediante una etiqueta o tag RFID.
Dicha cerradura inteligente permitirá que un usuario con un llavero admitido pueda ingresar con el solo hecho de apoyar el llavero sobre el lector y solo usuarios admitidos puedan entrar. De esta forma podremos asignar más de un usuario a una misma cerradura (cada cerradura contará con una lista de llaveros admitidos). Y podremos asignar la misma llave a diferentes cerraduras. Por ejemplo, un profesor podría tener acceso al laboratorio y al aula 42 si así se determina que tiene que ser. Si la clave es correcta se habilita el acceso mostrando un mensaje de bienvenida y se activa el mecanismo de apertura, en caso contrario se lanza el mensaje error.

Página web del proyecto: https://github.com/gaq07ar/CI-AMega2560

Objetivos

A continuación enumeran algunos de los objetivos principales de la solución

- Desarrollar un sistema que sea capaz de habilitar o no la correspondiente Autorización de acceso a un conjunto determinado de recursos protegidos, mediante la tecnología arduino.
- Se debe poder trabajar con repositorios y credenciales en Cloud.
- Se debe poder instalar de forma sencilla en cualquier edificio.
- El acceso a internet es obligatorio.
- Se debe contar con la posibilidad de trabajar con múltiples llaves y cerraduras.

Alcance y su justificación

Con respecto al alcance del proyecto, se va a desarrollar lo siguiente:

- Permitir el ingreso de un usuario autorizado.
- Rechazar el ingreso de personal no autorizado.
- Definir el almacenamiento de usuarios autorizados.
- Permitir el ingreso de un usuario asignado a múltiples cerraduras.
- Registrar fecha y hora de ingreso de un usuario X.
- Proveer de una API rest para la integración entre el dispositivo arduino y el servicio en cloud. Dicha API debe contar con mecanismos para crear/eliminar puertas, usuarios y llaves, asimismo se deben poder realizar las asociaciones entre usuarios y llaves y llaves y puertas.

- Diseñar el sistema mediante el uso de una base de datos descentralizada, en Cloud, con las consideraciones de seguridad informática que sean requeridas para evitar el robo o la pérdida de información.

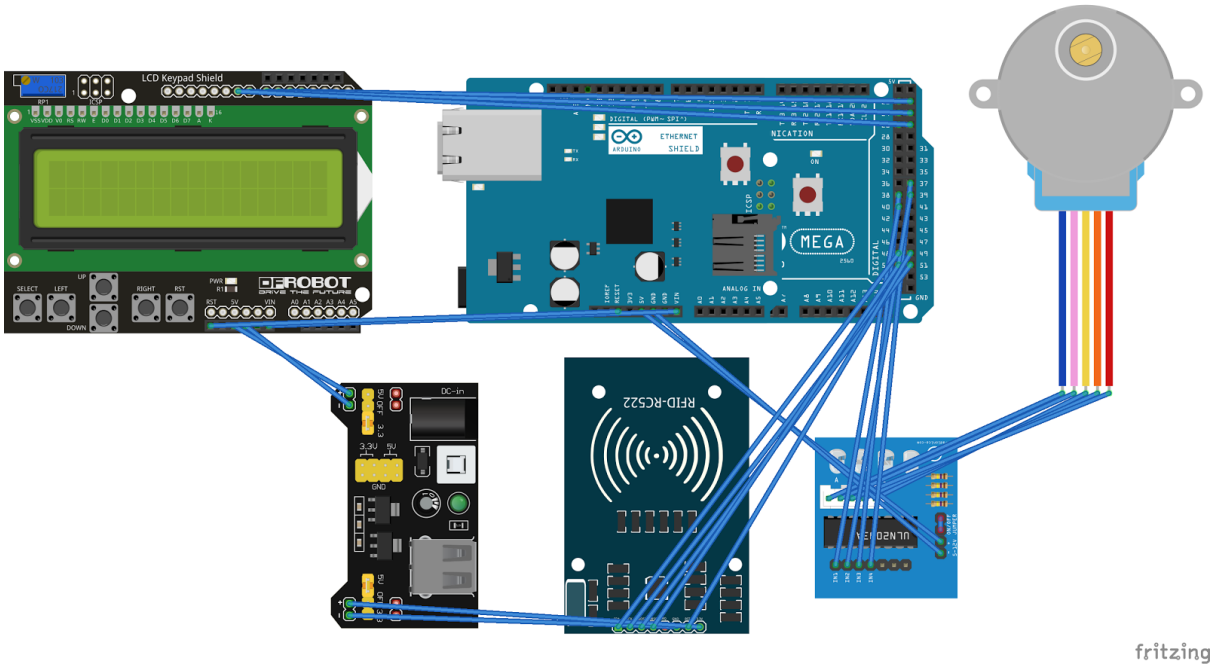
Materiales utilizados

Descripción	**Nombre técnico**	**Precio**
Arduino Mega 2560	Arduino Mega 2560 (Rev3)	$290
Display	LCD Keypad Shield	$199
Driver Motor Stepper	ULN2003A	$80 **
Ethernet Shield	Arduino Ethernet Shield (Rev3)	$229
Motor Stepper	28BYJ-48 Stepper Motor	**
RFID	RFID-RC522-v3	$87 ***
YwRobot Power Supply	YwRobot Breadboard Power Supply	$76
Kit Cables	Kit 120 Cables 20cm	$156
Mantención servidor	Server en Digital Ocean	$170
		$1287

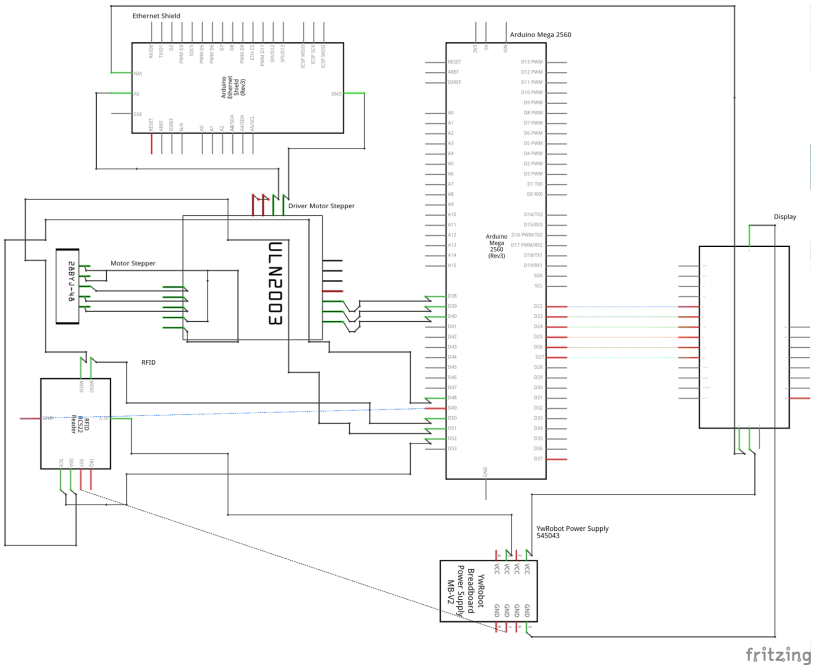
* Precios obtenidos en Mercado Libre.
** El motor stepper y su correspondiente driver se suele vender en combo.
*** Viene el módulo RFID más una tarjeta y un llavero tag.

Esquema de conexionado

Representación física:



Esquema lógico:



* Ambas imágenes pueden verse en la web del proyecto.
** Para diagramar el conexionado utilizamos una herramienta llamada Fritzing (http://fritzing.org/).

Actividades realizadas

1. Seteo entorno de trabajo

   Para poder trabajar con Arduino Mega tuvimos que instalar el IDE oficial de Arduino que se puede encontrar en la siguiente página:
   https://www.arduino.cc/en/Main/Software
   Donde hay una versión para cada sistema operativo.

   En orden de trabajar más cómodos utilizamos además otra herramienta que se llama Visual Studio Code, que puede ser encontrada aquí:
   https://code.visualstudio.com/

Para poder trabajar con este IDE sin embargo hay que configurar un par de cosas:

   a. Instalar la extensión de Arduino desarrollada por Microsoft (esto automáticamente instalará la extensión C/C++ necesaria para trabajar con Arduino).
   b. Entrar en el archivo de configuración mediante “Ctrl+,”.
   c. Setear tanto el directorio donde tenemos instalado el IDE de Arduino (en nuestro caso “/opt/arduino”)y cambiar la configuración del intellisense de “default” por “Tag Parser” porque el complemento de C/C++ todavía no soporta el intellisense que viene por defecto como sigue:

```
{
        "arduino.path": "/opt/arduino/",
        "C_Cpp.intelliSenseEngine": "Tag Parser"
}
```

   d. Con esto ya tenemos el IDE funcionando que proporciona características extras sobre el IDE oficial de Arduino como autocompletado, búsqueda de palabras claves, etc. Documentación sobre cómo utilizar la extensión de Arduino se puede encontrar en el siguiente link.

2. Conexionado.

   a. Abrir el archivo en el siguiente link con el programa Fritzing para poder apreciar las conexiones en detalle: https://goo.gl/QaWtLv.

3. Desarrollo e implementación de código
    a. Para la implementación del código tuvimos que instalar las siguientes librerías:
        i. LiquidCrystal.
        ii. MFRC522.
        iii. SPI.
        iv. Ethernet.
    b. A fines estéticos y de continuidad del proyecto, el código del programa del proyecto se encuentra en un repositorio público de GitHub del proyecto en el siguiente link: https://github.com/gaq07ar/CI-AMega2560/blob/master/Codigo/ProgramaPrincipal.ino.
    c. Tener en cuenta que este código funciona para el esquema de conexionado mencionado en el punto 2. De lo contrario no funcionara debido a que los pines se configuran tanto en la parte física (conexionado) como en la parte lógica (en el software del programa).

Funcionamiento

Por cada Arduino que configuremos se deberá setear el número de puerta tanto en la página web como en el código que será subido a Arduino para esa puerta en específico. De modo que quedaría como sigue:

1. En ProgramaPrincipal.ino:

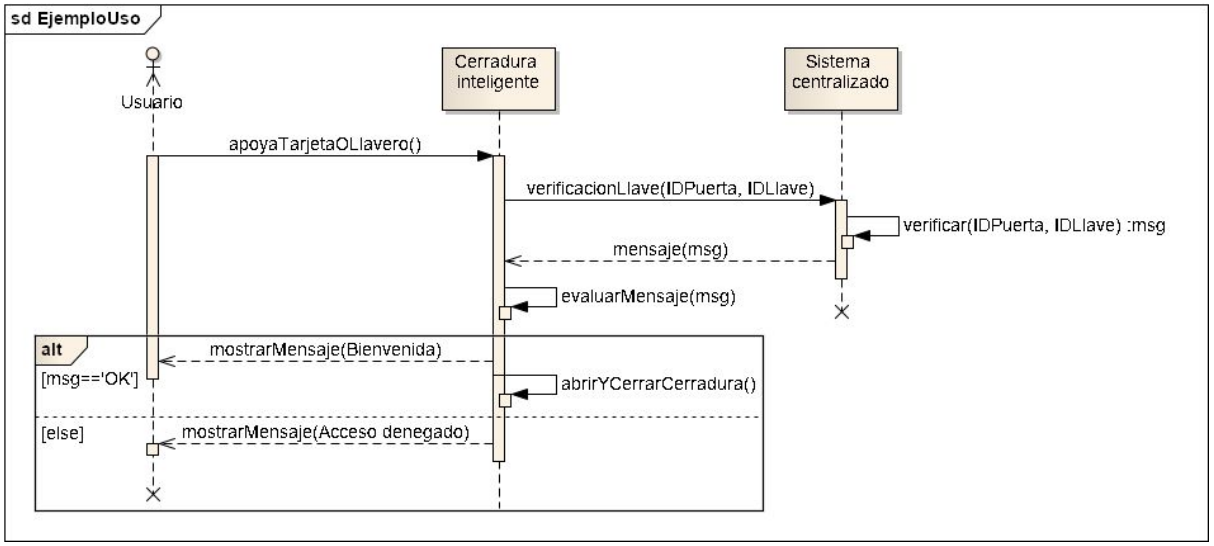
```
char idPuerta[] = "1";
```

2. En la web del sistema centralizado en la siguiente URL:

```
{
  "creationDate": "2017-11-29T23:03:22.979Z",
  "enabled": true,
  "id": "string",
  "publicIdentification": "1"
}
```

Donde se especificará en “publicIdentification” el ID de la puerta en cuestión.

Ejemplo de funcionamiento:



Vista completa en:
https://raw.githubusercontent.com/gaq07ar/CI-AMega2560/master/EjemploUso.png

Funcionamiento servicio en cloud

Servicios Business

Características principales de los servicios

Describimos las características principales del servicio Business PPS:

- Se ejecuta en su propio proceso.
- Se despliega independientemente.
- El servicio posee su propio modelo de datos interno. Este por ser interno no se expone al mundo exterior y lo que se provee externamente es una definición de API.
- El servicio posee su repositorio en Gitlab.
- El servicio es stateless, de esta forma puede escalar horizontalmente. Es decir no mantenemos el estado global de ningún atributo de la solución en un único servicio, como por ejemplo sesiones o contadores globales.

- El servicio no comparte su bases de datos con ningún otro componente, esto es para evitar cualquier tipo de acoplamiento que impidan evolucionar a la solución.
- El servicio es Testeable con Test de caja Negra. Se puede utilizar JMETER o similar.
- Orientado a DevOps, posee una API que permite determinar si el servicio esta disponible o no.
- El manejo de versiones se resuelve a través del Header HTTP de los mensajes Rest y la versión de un determinado Servicio está ligada a la versión de la API. Semantic versioning.

Dependencias y utilidades utilizadas para el backend

Java

Java SDK 8, para Sistemas Operativos linux, ya que hemos elegido ubuntu como imagen de nuestro SO en Digital Ocean.
Ver anexo.

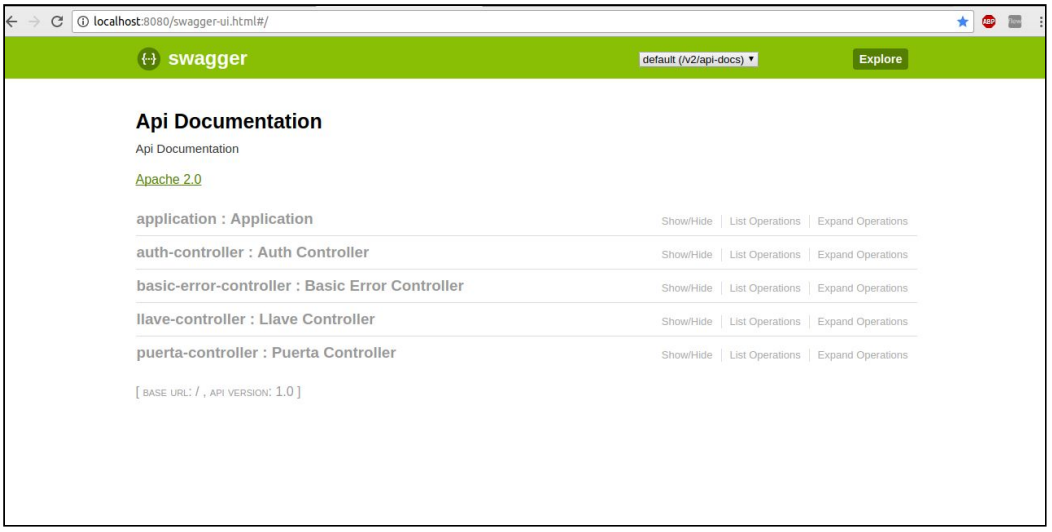
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

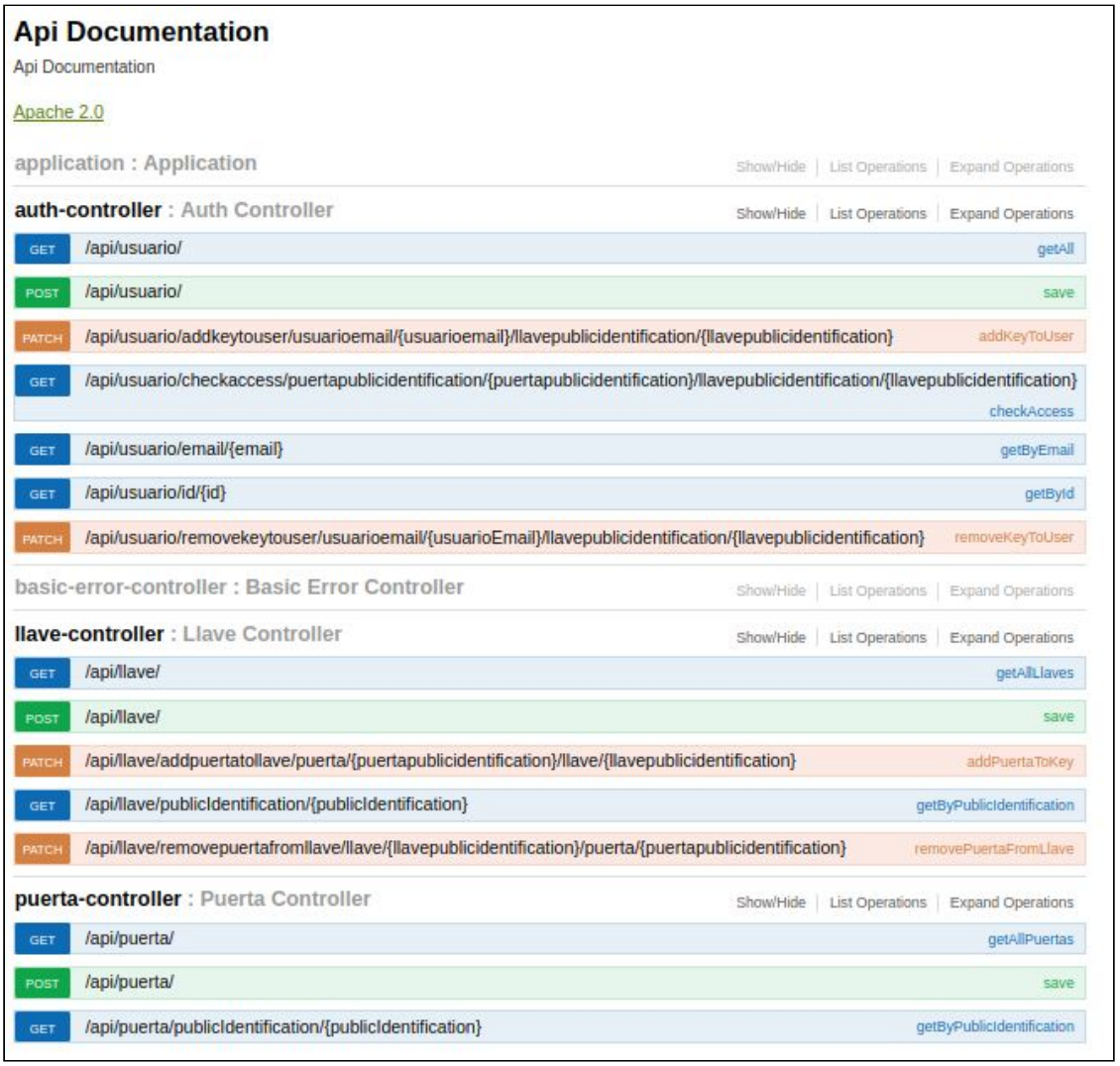
SpringBoot

Framework Java, ver anexo

Swagger/Swagger UI

Framework Java para el desarrollo de servicios
Posee una UI que permite acceder a todos los endpoints de la solución a través del puerto 8080, en el host correspondiente.
Todos los endpoint que desarrollamos para administrar la solución, se encuentran documentados en la UI de swagger.





Github

Para la gestión de la configuración (scm) utilizaremos cuentas públicas de github.

La URL del repositorio cloud es git@github.com:mariano-dim/pps.git

Cloud Service

El servidor cloud que utilizaremos será Digitalocean, debido a su excelente documentación, su accesibilidad económica y a la experiencia que contamos con dicha plataforma.
En dicho servidor, residirá un servidor web rest, desarrollado en java (ver anexo), que persiste los datos en un motor de base de datos MySQL.
Toda la información de usuarios, llaves y permisos se encuentra centralizada en dicho server.

Droplet basico en cloud DigitalOcean

Especificaciones de HW

- Server x86/ 2 CPU
- 4 GB de ram
- 30 GB SSD Disk
- 4 TB de transferencia de datos (entrante y saliente)

Especificaciones de SW

- Ubuntu server 16.04
- MongoDB

https://www.digitalocean.com/?refcode=46854a426040

MongoDB

Con respecto al motor de base de datos los backups los realizaremos utilizando **mysqldump**. Este "backup" realiza un export de todas las tablas, pero estas no conservan el mismo punto en el tiempo. o sea, en cuanto arranca a leer la tabla, ejecuta el backup completo de la misma y sigue por la siguiente. si esta tiene datos nuevos, los va a salvar. pero en la anterior se perdieron.
https://www.mongodb.com/

Docker

Toda solución cloud debe permitir el crecimiento horizontal de la misma de forma nativa, por esta razón hemos utilizado mongoDb, además de que trabajamos con formato JSON en nuestra API rest, lo cual se adapta perfectamente al formato estándar del motor de base de datos.
Cuando decimos que nuestro servicio escala horizontalmente, lo mismo que el motor MongoDb, queremos indicar que si hubiera que soportar una carga grande de requerimientos, por ejemplo 500 trx por segundo o mas, la arquitectura planteada lo podria

soportar. Por ejemplo si instalamos el sistema en una gran complejo con cientos de puertas y miles de usuarios, se dara el caso de que en algunos momentos haya picos en los accesos, lo cual requiere que nuestro sistema pueda responder rápidamente a los requerimientos. En la solución actual hay una sola instancia de MongoDB, pero se ser necesario se puede utilizar replica set o sharding para soportar mayor carga. Lo mismo aplica para nuestro servicio, al ser stateless, puede haber cientos de instancias ejecutándose en paralelo, lo único que se requiere en ese escenario es instalar un API Gateway o un Load balancer para distribuir la carga entre todas las instancias.
El motor de base de datos lo hemos instalado en un contenedor docker para facilitar la gestión e instalación del mismo.
Docker es un contenedor, algo similar a una VM pero que utiliza menos recursos y comparte algunos servicios y librerías del SO host, lo que los hace más livianos e idóneos para el despliegue de soluciones en distintos ecosistemas.

https://www.docker.com/

Docker-Compose

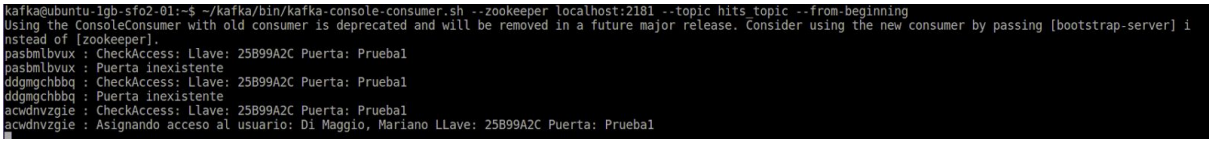
Un orquestados local de docker.

Kafka Broker

Hemos instalado un Kafka Broker, que es un sistema de mensajería persistente para la gestión de los Hits o eventos que se producen en el sistema, cuando un usuario intenta ingresar a través de una puerta.
Kafka es una base de datos noSQL, orientada a eventos.
https://kafka.apache.org/

Para observar los eventos que se van generando el sistema, se pueden utilizar los mecanismos provistos por kafka, como por ejemplo los siguientes comandos:

~/kafka/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic hits_topic --from-beginning
Muestran los eventos o mensajes que se van generando

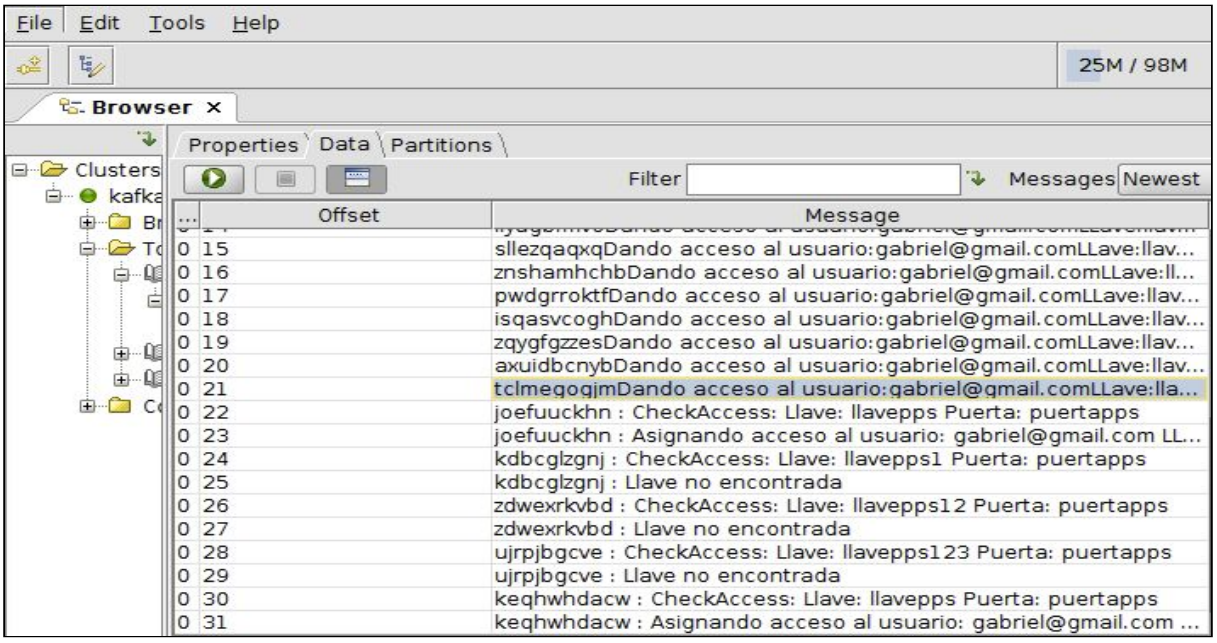


La utilizamos para generar eventos cuando un usuario intenta ingresar al sistema, estos eventos están relacionados con las validaciones que realiza el sistema cuando es invocado el endpoint **checkAccess** desde la aplicación arduino (cliente).

Siempre se genera un evento por cada invocación al servicio con los datos que se pasaron por parámetro, como lo son la **Puerta** y la **Llave**.
Debido a que no encontramos una imagen de docker madura como trabajar con Kafka, decidimos realizar la instalación en el servidor.

Kafka Tool

Una herramienta UI que permite de una forma amena ver los datos de kafka.
http://www.kafkatool.com/
El comando **kafka-console-consumer.sh**, hace lo mismo, pero esta herramienta provee una interfaz amigable. La herramienta solo fue utilizada para desarrollo.



Mongo-Express

Herramienta utilizada para gestionar MongoDB

La hemos instalado en docker de la misma forma que el motor para facilitar la gestión e instalación.

Provee una interfaz web que se puede encontrar en el puerto 8081 del servidor donde se encuentre instalado.

Endpoint disponibles en la API Rest

Si bien la aplicación auto-documenta los endpoints (a través de Swagger), definimos también aquí los mismos.

Auth-controller

(GET) /api/usuario

Obtiene todos los usuarios registrados en el sistema en formato json.

(POST) /api/usuario

Crea un nuevo usuario en el sistema.

Ejemplo del json que se le pasa como parámetro

```
{
  "creationDate": "2017-12-04T02:31:01.146Z",
  "email": "pps@gmail.com",
  "nombre": "pps"
}
```

(PATCH) /api/usuario/addkeytouser/usuarioemail/{usuarioemail}/llavepublicidentification/{llavepublicidentification}

Le agrega una llave a un determinado usuario.

(GET)

/api/usuario/checkaccess/puertapublicidentification/{puertapublicidentification}/llavepublicidentification/{llavepublicidentification}

Chequea si una llave (de un usuario) puede abrir una puerta

En caso de satisfacer las validaciones, retorna OK concatenado con el símbolo numeral, concatenado con el email del usuario propietario de la llave. Además registra un evento en Kafka para el analisis posterior.

En otro caso retorna ERROR y registra un evento en Kafka.

(GET) /api/usuario/email/{email}

Retorna los datos del usuario que tiene el email enviado como parámetro.

(GET) /api/usuario/id/{id}

Retorna los datos del usuario que tiene el id enviado como parámetro.

(PATCH) /api/usuario/removekeytouser/usuarioemail/{usuarioEmail}/llavepublicidentification/{llavepublicidentification}

Elimina una llave del usuario, dado el usuario y la llave pasados como parámetros.

Llave-controller

(GET) /api/llave

Retorna los datos de todas las llaves en formato json

(POST) /api/llave

Crea una nueva llave el sistema, se pasa como parámetro un parámetro json con los datos de la llave.

Una llave puede tener un usuario y puede tener 0..n puertas. En el alta se le pueden pasar estos parámetros o asignarlos con posterioridad.

Ejemplo del json que se le pasa como parámetro

```
{
  "creationDate": "2017-12-04T02:31:01.273Z",
  "enabled": true,
  "publicIdentification": "llavepps",
  "puertas": [
    "puertaprincipal"
  ],
  "usuario": "pps@gmail.com”
}
```

(PATCH) /api/llave/addpuertatollave/puerta/{puertapublicidentification}/llave/{llavepublicidentification}

Agrega una puerta a una llave.

(GET) /api/llave/publicIdentification/{publicIdentification}

Obtiene los datos de una llave a través de su identificados público.

(PATCH) /api/llave/removepuertafromllave/llave/{llavepublicidentification}/puerta/{puertapublicidentification}

Desasocia una puerta de una llave

Puerta-controller

(GET) /api/puerta/

Obtiene los datos de todas las puertas

(POST) /api/puerta/

Crea una puerta con los datos que se le pasan el un json.

Ejemplo

```
{
  "creationDate": "2017-12-04T02:31:01.348Z",
  "enabled": true,
  "publicIdentification": "puertaprincipal"
}
```

(GET) /api/puerta/publicIdentification/{publicIdentification}

Obtiene los datos de una puerta mediante el identificar publico de la misma.

Compilación del código fuente backend o servicio

El proceso de compilación del código fuente del backend (servicio Java PPS) se realiza utilizando una herramienta denominada maven del proyecto Apache.
https://maven.apache.org/

Para realizar la compilación del proyecto se puede trabajar con una IDE como Eclipse u alguna otra (o también maven mediante la línea de comandos), a la cual se le define que el proyecto es uno de tipo maven y se le indica el archivo principal de maven, que se denomina.

Pom.xml y se encuentra en la raíz del proyecto.

Con fines de realizar la documentación del proyecto realizamos las indicaciones que corresponden al proceso completo desde línea de comandos.

El siguiente comando compila el código fuente y genera el artefacto (jar) con las clases compiladas que es ejecutable.
Hacemos notar que no todos los archivos jar son ejecutables, pero en nuestro caso por trabajar con el framework SpringBoot, el mismo se encarga de hacer ejecutable nuestro jar.

mvn clean install

Una vez generado el JAR (Java Archive), el mismo por ser un ejecutable se puede invocar desde la línea de comandos.
Con el siguiente comando se despliega la aplicación

java -jar target/PPS-0.0.1.jar

https://es.wikipedia.org/wiki/Java_Archive

Despliegue del servicio backend en el servidor cloud

Teniendo disponible el usuario y la clave del servidor cloud, mediante el comando scp se puede realizar la copia segura de los artefactos al servidor, mediante el siguiente comando:

scp target/PPS-0.0.1.jar root@138.197.206.194:/root

```
marianodim@ubuntu:~/Desktop/proyectos/pps$ scp target/PPS-0.0.1.jar root@138.197.206.194:/root
root@138.197.206.194's password:
PPS-0.0.1.jar
marianodim@ubuntu:~/Desktop/proyectos/pps$
```

Una vez copiado se ingresa al servidor mediante ssh, como aqui:

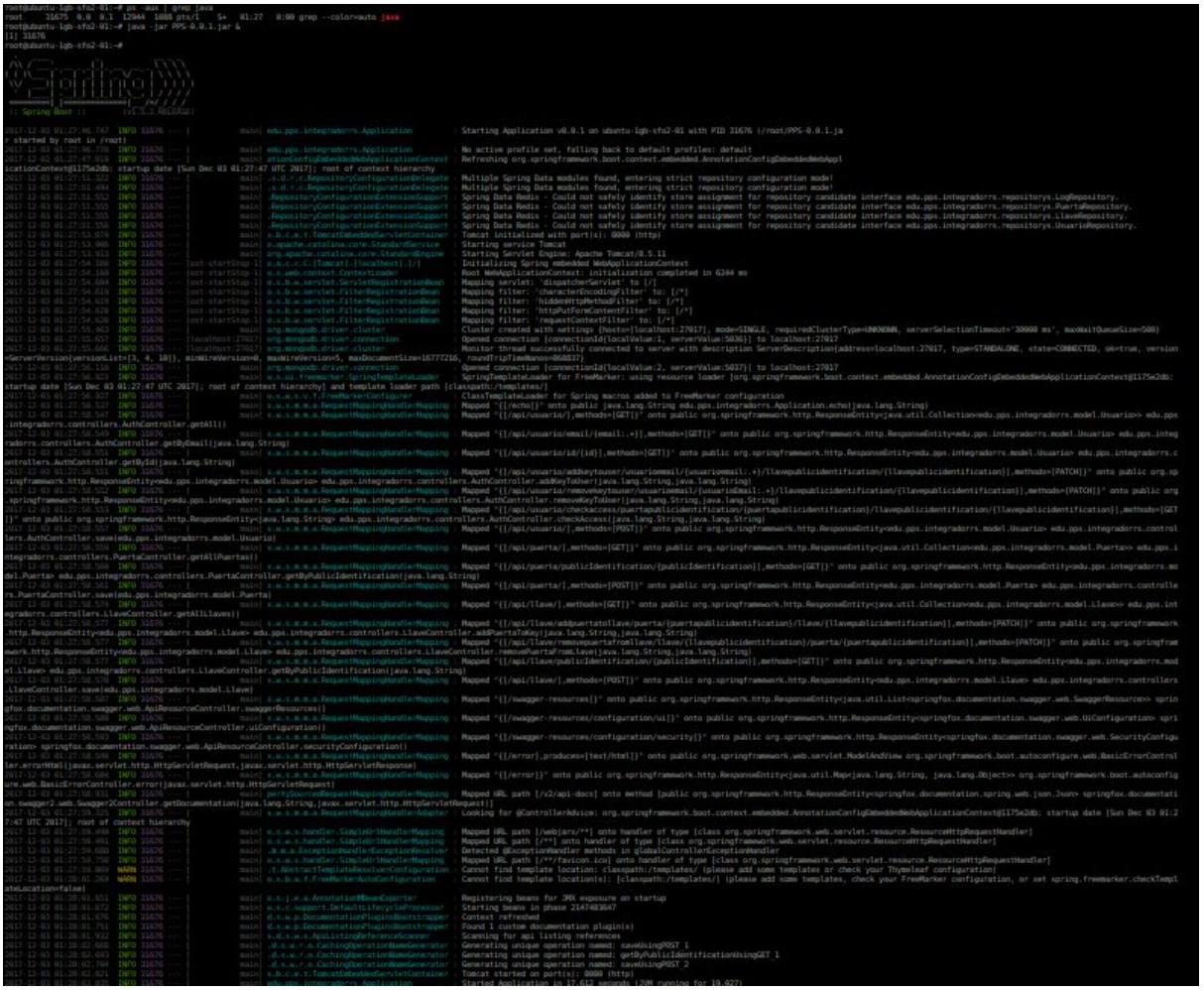
ssh root@138.197.206.194
El servidor requerida que ingresemos nuestra clave de acceso.

```
marianodim@ubuntu:~/Desktop/proyectos/pps$ ssh root@138.197.206.194
root@138.197.206.194's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-97-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

48 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Sat Dec  2 22:49:18 2017 from 190.190.193.126
```

Una vez logueados en el servidor, directamente ejecutamos el archivo JAR

java -jar PPS-0.0.1.jar &

El ampersand es para que cuando salimos de la consola el proceso continúe en ejecución.



Anexo

Docker-Compose

Archivo docker-compose utilizado para el despliegue de MongoDb y Mongo-Express

```
version: '2'

services:

 mongodb:
   image: mongo:latest
   environment:
     - MONGO_DATA_DIR=/data/db
     - MONGO_LOG_DIR=/dev/null
   volumes:
     - ~/data:/data/db
     - ~/databasedata:/databasedata
   ports:
     - 27017:27017
   command: mongod --smallfiles --logpath=/dev/null # --quieto


 mongo-express:
   image: mongo-express
   restart: always
   mem_limit: "512m"
   environment:
     - ME_CONFIG_OPTIONS_EDITORTHEME=ambiance
     - ME_CONFIG_MONGODB_SERVER=mongodb
   ports:
     - 8081:8081
```

```
links:
  - mongodb
```

Java como lenguaje de programación

Java es uno de los lenguajes de programación más utilizados y esa fama tiene más de una justificación. Haremos foco en las principales características de Java relacionadas con el modelado de la capa de negocio.

Brevemente resumimos algunas de las principales cuestiones que justifican dicha decisión.

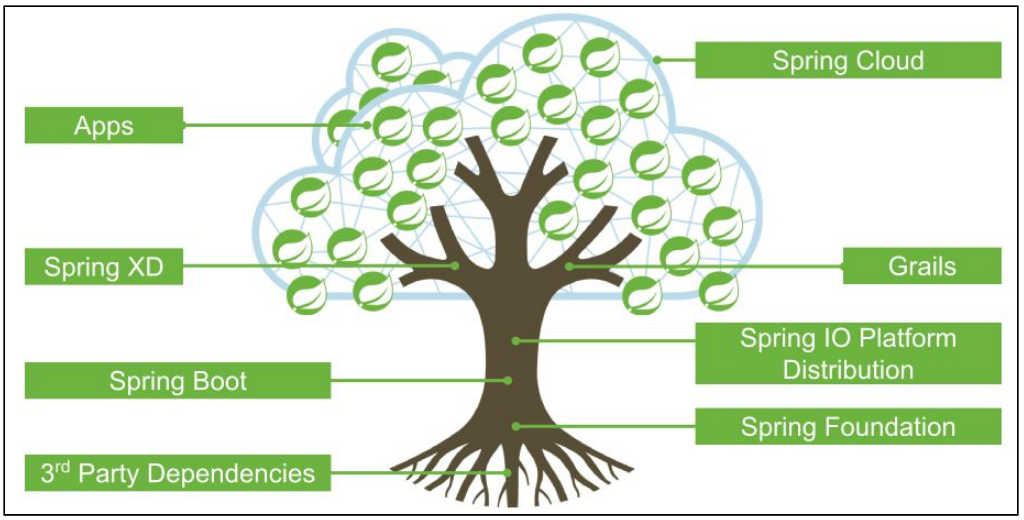
1. Java es soportado por la gran mayoría de los Sistemas Operativos y dispositivos. Es Multi-Platform.
2. Java es un lenguaje maduro y con el paso de los años ha ganado cada vez más terreno; hoy día existen librerías, frameworks y componentes Java que brindan soporte a casi cualquier aspecto relacionado con el desarrollo de software.
3. Existe una gran oferta de desarrolladores Java en el mercado.
4. Java es Multi-thread y Multi-process. Se ha ido convirtiendo en un lenguaje de programación robusto. El core del lenguaje está muy optimizado. Es compilado lo que lo hace muy eficiente y veloz para la mayoría de las tareas.
5. Es seguro. Parches de seguridad se liberan permanentemente.
6. Java es un lenguaje de vanguardia, es decir que está presente en los desarrollos relacionados con tecnologías de última generación.
7. Java es compatible Backward, lo que significa que podemos utilizar las últimas versiones del lenguaje para ejecutar código fuente de versiones muy anteriores.
8. Posee muy buena documentación.
9. Es un lenguaje orientado a objetos.

Spring Boot como Framework

Dentro del mundo Java existen cientos (o miles) de frameworks y librerías, que resuelven casi cualquier aspecto del desarrollo de diversas formas. De ese conjunto utilizamos el framework Spring Boot (aunque Spring Boot es mucho más que un framework lo denominaremos así para facilitar su comprensión).

Spring Boot es un framework Java soportado por una comunidad internacional gigantesca.

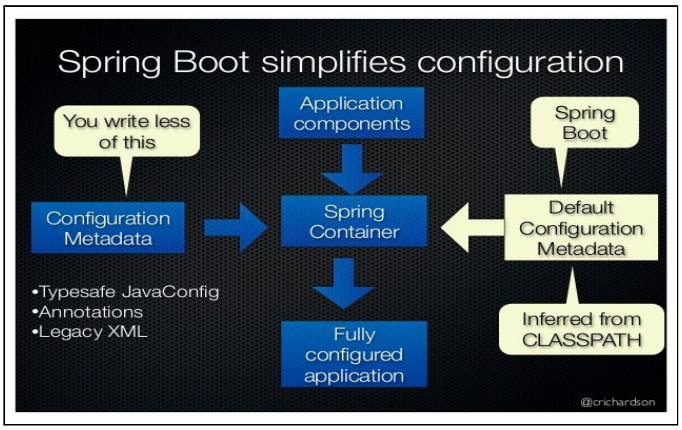
Spring Boot a su vez se apoya sobre el core de proyectos de Spring y librerías de terceros (Spring Foundation y Third Party Dependencies).



Características principales

Dentro del amplio conjunto de características que acompañan el desarrollo sobre este framework, podemos indicar las siguientes características principales:

1. Soporta el desarrollo ágil out of the box, es decir que el framework está pensado con el objetivo (misión) de facilitar el desarrollo netamente productivo desde el primer momento, para que el Developer pueda pensar en el problema del negocio que intenta resolver y no en los aspectos propios del framework.
2. Provee Convention over Configuration, esto significa que no nos tenemos que preocupar por la configuración, ya que todo componente de Spring incluye su configuración por defecto). Esto acelera el desarrollo. Obviamente habrá que verificar que la configuración por defecto cumple con los requerimientos no funcionales del proyecto particular en estudio.
3. Provee mecanismos out of the box para resolver cualquier aspecto no funcional del desarrollo de software moderno (Alta Disponibilidad, Integración, Seguridad, Orientado a Devops, Servidores Embebidos, métricas).
4. Proveer integración nativa con Flyway, para realizar las migraciones de datos versionadas.
5. Provee integración nativa para Internacionalización I18N.
6. Gestion efficiente de contextos (develop, production, test).



Conceptos clave

- Simplifica el desarrollo al establecer convenciones que se deben seguir para el correcto desarrollo de la aplicación.
- El framework provee la posibilidad de que realizar aplicaciones standalone al embeber un servidor web (Tomcat, Jetty u otros) en la propia aplicación. Esto permite la creación de aplicaciones autocontenidas, facilitando el despliegue de las mismas al no requerir componentes adicionales para su funcionamiento.
- Las herramientas que provee el framework, así como sus características, permiten un rápido desarrollo de aplicaciones. El hecho de proveer una aplicación encapsulada lo hace ideal, por su rápido despliegue, para el desarrollo de servicios que provean una API REST.