Universidad
Rey Juan Carlos

ESCUELA TÉCNICA Y SUPERIOR DE

INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Curso académico: 2012/2013

Proyecto de Fin de Carrera

CPDPocket:

Gestión de un CPD

desde tu bolsillo

Autor: Marcos Lara Torres

Tutor: Gregorio Robles Martínez

Proyecto Fin de Carrera

CPDPocket,
gestión de un CPD desde tu bolsillo

Autor

Marcos Lara Torres

Tutor

Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el
día de de 2013,
siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Móstoles, a de de 2013

A mis padres y a Sara

Agradecimientos

Para mi este proyecto significa mucho ya que ha marcado una notable evolución personal a lo largo de la gran cantidad de tiempo que, sin quererlo, ha supuesto terminarlo.

Siempre supe que quería estudiar informática, y en cierto modo encumbré esta idea. A lo largo de mi vida académica, los hitos que de alguna manera marcaban la posibilidad o no de llegar a conseguirlo siempre me supusieron más nervios de lo normal. Y este, no podía ser menos.

Para llegar hasta aquí he pasado por muchas dudas, muchos cambios de idea a la hora de elegir un tema, y sobre todo un cambio en la percepción de lo que para mi era el Proyecto de Fin de Carrera.

Sin duda este proyecto marca el final de una etapa muy importante en mi vida, pero sobre todo marca el comienzo de una nueva, y espero que sea al menos tan gratificante como esta. Atrás quedan muchos sacrificios para conseguir llegar hasta aquí, pero ha merecido la pena.

Quiero agradecerles:

A mis padres, que me hayan moldeado desde pequeño para darme una buena educación. Que me hayan dejado la libertad suficiente para desarrollar mi creatividad en las cosas más cotidianas de la vida.

A mis hermanos mayores, Carlos y Raúl, a los que tengo una gran admiración. Sin duda toda mi idea de la informática empezó por querer imitarlos cuando ellos sabían manejar aquel Amstrad y yo copiaba texto solamente para pasar tiempo delante del ordenador (una pena que no les picara el gusanillo de la informática como a mi)

A mi novia Sara, por aguantarme tantos años, y apoyarme en tantos momentos difíciles confiando en mí siempre más de lo que lo he hecho yo mismo. Y enseñarme a ver los problemas de una manera más humana.

A mis cuñadas Pi y Clara.

A mis sobrinos Noelia, Héctor, Hugo, Myriam, y en especial a Iván por ayudarme a recordar como se ven las cosas desde los ojos de un niño.

A mi tía favorita, Carmen.

A mis amigos de siempre Héctor, José y Álvaro, con los que tantas veces “he salvado el mundo”, por servirme de válvula de escape.

A mis compañeros de carrera, y en especial a Ismael Sánchez, con los que tantos problemas hemos sacado adelante.

A esos profesores que muestran un interés especial por enseñar y muestran cada día su notable preparación.

A mi primer compañero de curro, que sin duda se ha convertido en un gran amigo. Paco Lain, por ayudarme a ver la vida laboral de una manera más eficiente y menos formal. Sin olvidarme tampoco de Alberto Felguera.

Como no, a los integrantes del grupo GsyC/LibreSoft en especial a Gregorio Robles y al equipo de sysadmins, por la oportunidad que me ofrecieron en su día, y por la colaboración en la elaboración de este proyecto.

Por último, tampoco me puedo olvidar de todas aquellas personas que no han confiado en mi, no les guardo rencor, ya que en su momento me sirvieron de aliciente para superarme.

Gracias a todos ellos, soy el informático, y sobre todo la persona que soy hoy, tanto para bien como para mal.

Me gustaría dedicar este Proyecto de Fin de Carrera a todas las personas que me han apoyado a lo largo de toda mi vida, en especial a mis padres, y a mi novia.

Resumen

En la actualidad, muchas son las empresas que a pesar de no dedicarse a nada relacionado con la informática, necesitan tener un entorno de máquinas relativamente grande para el cálculo, almacenamiento de datos o simplemente autogestionar sus servicios. Esta necesidad empuja a las empresas a crear una infraestructura mínima de máquinas capaces de cubrir todas sus necesidades. De esta manera, surgen los Centros de Procesado de Datos y alrededor de éstos se crea un grupo de personas capaces de administrar y mantenerlas. Este Proyecto de Fin de Carrera está orientado a facilitar el trabajo de estas personas.

Dentro de este contexto, y dando por hecho la situación social en la que nos encontramos, en el que la mayoría de las personas posee un teléfono inteligente capaz de realizar operaciones tan complejas que hace no mucho sólo estaban pensadas para máquinas mucho más grandes.

Se ha diseñado e implementado un sistema capaz de almacenar información relativa a las máquinas de un Centro de Procesamiento de Datos para su posterior consulta. Además de esta funcionalidad, se le ha añadido la posibilidad de consultar en tiempo real algunos datos relativos a su disponibilidad. Ejemplos de estos datos son la predisposición de ciertos puertos previamente registrados, la dirección IP de la máquina asociada al nombre que está registrado, y comprobar que la máquina responde a una llamada de ping. Todos esto nos permite crear perfiles personalizados que se ajusten a las necesidades de los distintos tipos de máquinas.

Este servicio se ofrece a través de una web que se puede consultar desde cualquier dispositivo con soporte para navegación. De esta manera, un administrador de sistemas podrá acceder a esta información desde su móvil sin necesidad de tener que acudir a un puesto de trabajo más potente para ello, y gracias a esto evitar tener que desplazarse hasta él.

Por último, y para facilitar aún más la consulta por parte del trabajador, se ha investigado a cerca de la reutilización de software de terceros para poder identificar las máquinas de una manera relativamente automática. Un ejemplo de esto se ha conseguido gracias a la aplicación “Barcode Scanner”, que nos permite leer un código de barras vinculado a la máquina y redirigirnos a la parte de nuestro sistema correspondiente en un navegador.

Por todo esto, y por la idea de poder administrar un Centro de Procesamiento de Datos desde el bolsillo, este Proyecto de Fin de Carrera recibe el nombre de **CPDPocket**.

Índice general

1 Introducción

1.1 Estructura de la memoria

Para tratar de ayudar a comprender un poco la estructura de la memoria, y que de esta manera se obtenga una mayor compresión de los temas expuestos en ella, a continuación se hace una breve descripción de cada uno de los puntos principales que forman este documento.

- **Introducción.**

  En este apartado tratamos de poner en contexto al lector para que pueda entender todo el proceso de diseño y elaboración de este proyecto. Se tratan temas importantes como son la motivación que llevó a la generación de la idea, y los objetivos de la misma.

- **Estado del arte.**

  Aquí se describirán las diferentes tecnologías que se usan directa, o indirectamente en el proyecto para tratar de ilustrar brevemente a los lectores con menos conocimientos en estos campos de la informática, y a modo de resumen para lo más avanzados.

  Aquellos que lo crean necesario deberán completar esta información según su grado de inquietud con los distintos temas.

- **Diseño del sistema.**

  En este apartado se describe el diseño del sistema completo. Diferenciando los diferentes bloques que generan el sistema completo de la aplicación.

- **Implementación.**

  En este apartado se describe la implementación definida por el análisis descrito en el apartado anterior, junto con las decisiones finales.

- **Uso del sistema en un entorno real.**

  En este apartado describimos las características de una implantación controlada en un entorno real, el CPD del grupo GsyC/LibreSoft, para complementar el desarrollo y este documento.

1.1 Estructura de la memoria

- **Conclusiones y futuras lineas de trabajo.**

  En este último apartado se detallan las conclusiones de este proyecto, así como las justificaciones de algunas de las decisiones más importantes tomadas durante su implementación.

- **Glosario**

  Por último se añade un glosario de términos para tratar de facilitar la compresión del lector, en caso de ser necesario.

- **Apéndice**

  Se presenta como anexo, un apéndice que muestras los modelos de caso de uso de este desarrollo, para su consulta y mejora de la comprensión del sistema. Se proporcionan como anexo para tener concentrado todo este tipo de material, evitando así que se entremezclen con el resto de la memoria.

  Además también son competencia de este punto el manual de instalación y el manual de usuario.

1.2 Contexto: Centro de Procesamiento de Datos

Para entender realmente cómo se pueden explotar todas las características de CPDPocket antes, debemos entender cómo surgió la idea.

1.2.1 ¿Qué es un CPD?

Un CPD, Centro de Procesamiento de Datos o Data Center, no es más que una agrupación de máquinas, en una infraestructura en la que se ha diseñado un ambiente controlado expresamente para ellas.

Su uso se remonta a los comienzos de la informática. Las primeras máquinas eran muy sensibles, requerían de mucho mantenimiento y ocupaban físicamente un espacio considerable.

Esto hacía que fuera necesario tener los sistemas separados, y centralizados geográficamente lo más próximos posibles, reduciendo la longitud de los cables a utilizar. No conviene olvidar que nos estamos refiriendo al tiempo en el que no se utilizaba el circuito integrado.

En la actualidad, los CPDs han cambiado considerablemente, pero la idea de base sigue siendo la misma.

Sin duda depende del tamaño de los mismos, pero generalmente un CPD que se precie tiene:

1.2.2 Elementos básicos de un CPD

- **Sistema de alimentación independiente**. Se trata de aislar, en la medida de lo posible, los fallos ajenos a las máquinas, para que nada detenga el servicio. Normalmente se crean varias redes de alimentación para que un único fallo no interrumpa el funcionamiento de todo el CPD. Además es muy común añadir generadores de corriente autónomos que se activen en caso de fallo eléctrico, y/o baterías que puedan enmascarar cortes puntuales.
- Estructuras metálicas parecidas a una estantería, diseñadas para apilar máquinas. Estas estructuras se denominan **RACKS**.

De esta manera podemos reducir la cantidad de metros cuadrados que ocupa un CPD sin tener que renunciar a una correcta refrigeración producida por máquinas demasiado próximas.

- En estas salas se genera mucho calor, ya que al fin y al cabo se trata de una agrupación de máquinas que trabajan las 24 horas del día. Para evitar esto, están dotadas de **climatizadores** que mantienen la temperatura ambiente suficientemente fría como para evitar averías.
- Al tener las máquinas concentradas en una misma zona, es más fácil diseñar una **topología de red mucho más eficiente**, y en ocasiones reducir el número de nodos por los que debe pasar la información desde el origen hasta el destino.
- Se aprovecha la proximidad de las máquinas para **reducir el número de terminales de interacción humana** necesarios para la administración del mismo. Es decir, un mismo "monitor + teclado + ratón" se utiliza para que un ser humano pueda acceder a varias máquinas.

  Esto se consigue:

  - KVMs (Keyboard-Video-Mouse). Estos dispositivos permiten conectar múltiples máquinas a un mismo conjunto de teclado, ratón y monitor, y conmutar las entradas para manipular en cada caso la que sea necesaria.
  - Carro de mantenimiento. Se trata de una estructura móvil equipada con un teclado, un ratón y un monitor, que permiten llevar y conectar estos periféricos a las máquinas de una manera relativamente cómoda, para operar con ellos.

1.2.3 Herramientas típicas de un CPD

En este apartado se pretende hacer especial mención a las herramientas que se utilizan dentro de un CPD, ya sean hardware o software, que tienen cierto interés para el proyecto.

- Software de monitorización. Existen distintas aplicaciones con este fin, ya sean específicas de un servicio, o genéricas de la máquina. Un ejemplo muy común de estos últimos es "Nagios".

- Nodo especial en la red. Se trata de una o varias máquinas que por su carácter administrativo tienen privilegios especiales, bien por tener acceso a varias redes que están separadas entre sí para aumentar la seguridad, o bien porque tienen acceso o información especial para el acceso a gran parte de los demás nodos.
- Los ya mencionados elementos hardware que permiten reducir el número de pantallas, teclados y ratones necesarios para la administración de las máquinas de un CPD.

1.2.4 Personal de un CPD

- Técnicos en general para el correcto funcionamiento de los elementos ajenos a las máquinas. Aquí se pretende englobar a los operarios que se encargan del sistema eléctrico, el aire acondicionado, etc. No porque tengan menos importancia, sino para tratar de enmarcar mejor el contexto de CPDpocket.
- Administradores de sistemas, que se encargan de la correcta configuración y funcionamiento de los servidores
- Expertos en comunicaciones, encargados de los diseños e implementaciones del sistema de red que permitirá la conectividad entre máquinas.

Estos dos últimos grupos utilizan a diarios las “herramientas típicas de un CPD” y poseen el perfil de usuario hacia el que está orientado CPDpocket.

1.3 Motivación

A lo largo de mi formación académica he realizado trabajos como becario en distintas empresas. En una de esas empresas trabajé como Administrador de Sistemas de un CPD con cientos de máquinas.

A continuación trataré de explicar brevemente algunos casos que sin duda fueron de inspiración para la elaboración de este Proyecto de Fin de Carrera.

1.3.1 Un día de trabajo normal

En el día a día era bastante habitual tener que ir físicamente al CPD para solucionar incidencias, bien porque algún usuario notaba algún comportamiento anómalo, o bien porque se recibía algún tipo de alerta automática de una herramienta de monitorización.

Antes de acudir a resolver la incidencia, se debe revisar la información registrada de la máquina para poder ir con el suficiente contexto. Cosas como un usuario y un password con suficientes privilegios son sin duda la clave para poder solucionar un problema.

Una vez allí, podemos encontrarnos con alguna máquina, diferente a la que vamos revisar, con algún tipo de indicador luminoso que nos indique que es necesario realizarle algún tipo de inspección. Esto puede obligarnos en la mayoría de las situaciones a dar más de un viaje para consultar más información. Este hecho, dependiendo de la lejanía del CPD con respecto a un terminal en el que podamos consultar los datos, causará mayor o menor impacto en nuestra productividad, pero siempre es un hecho valorable.

1.3.2 Una tarea excepcional

Además del trabajo normal de un administrador, tuvimos que enfrentarnos a la nada desdeñable tarea de hacer una migración completa del CPD de un edificio a otro.

Para poder lograr hacer esto ofreciendo una parada de servicio lo más corta posible, tuvimos que realizar tareas de recopilación de información, comprobaciones de configuración, y todo tipo de tareas previas al apagado que el lector pueda imaginarse para una vez parado el servicio, poder llevar la máquina a su nuevo emplazamiento, volver a encender la máquina y volver a ofrecer el servicio tratando de evitar cualquier tipo de error que dilatara la espera.

Una vez definido el plan de apagado, y diseñados los planos para la nueva ubicación de cada máquina, se puede comenzar la tarea. Se apagan las máquinas, se embalan y se envían al nuevo edificio. Una vez allí se desembalan, se identifican nuevamente y se procede a llevarlas a su nuevo emplazamiento.

Tras la reconexión de la máquina, se procede a su encendido. Además es necesario comprobar que el servicio se está ofreciendo, bien conectando un terminal directamente a la máquina o utilizando un nodo especial para poder acceder a la misma. Esta tarea se vuelve bastante monótona y laboriosa.

1.4 Objetivos

Se pretende realizar una aplicación que sirva para simplificar el día a día de un técnico de CPD.

La idea inicial trata de ampliar los servicios ofrecidos por un nodo especial de administración de un CPD. Se pretende que mediante un navegador, se puedan consultar datos específicos sobre las máquinas, además de ampliar en la medida de lo posible la información.

Gracias a la gran versatilidad de este tipo de aplicaciones, sumado al boom de las "nuevas tecnologías" en las que, por suerte, un alto porcentaje de la población es propietario de un dispositivo móvil compatible con servicios web a través de una gran cantidad de navegadores. Se parte de la idea de poder realizar bastantes tareas básicas a través de un nodo especial controlado por un dispositivo tan común como un móvil. Por esto, se hace el símil de manejar un CPD con el bolsillo, y de esta manera este proyecto recibe el nombre de **CPDPocket**.

Además aprovechando la gran versatilidad de estos dispositivos se ve posible explotar más potencial de los terminales, a parte del uso del navegador.

De este modo, se pretende crear una aplicación que:

- Almacene información importante relativa a las máquinas de un CPD, y que además permita servirla de una manera cómoda y segura.
- Aumente esta información estática en la medida de lo posible. Se pretenden añadir comprobaciones automáticas lanzadas a modo de script desde el servidor para aportar valores dinámicos que complementen la información estática que se pueda almacenar.
- Permita consultar esa información desde la mayor cantidad de dispositivos posible.
- Permita identificar de manera rápida una máquina y que esta identificación pueda desencadenar consultas directas en este Proyecto de Fin de Carrera

1.4 Objetivos

En este proyecto se pretende, además de estudiar en profundidad las tecnologías disponibles para poder ofrecer este servicio contando con las necesidades normales de un entorno de estas características, elaborar una aplicación que tenga un uso concreto y que cubra una necesidad. Es decir, se pretende realizar un proyecto más allá de la valoración como Proyecto de Fin de Carrera, pueda ser usado en la vida real.

Para ello, se quiere implantar en un entorno real y realizar un estudio de satisfacción de los usuarios, con el fin de evaluar el estado de la misma, y poder aumentar y mejorar su funcionalidad para el futuro.

2 Estado del arte

En este apartado pasarán a describirse la totalidad de las tecnologías usadas en este proyecto.

Trataremos de no profundizar demasiado, ya que en todos los demás subapartados de este bloque, podemos encontrar bastante literatura detallada sobre ellos, lo que sin duda nos obligaría a extendernos en exceso para dar una descripción precisa.

2.1 Modelo Cliente-Servidor

Para la elaboración de este proyecto, necesitamos una aplicación distribuida. Para facilitar su control, se ha elegido una estructura centralizada, obteniendo así un control de los datos almacenados. Y para facilitar su difusión, se utilizarán aplicaciones distribuidas que se comunicarán con el servidor central.

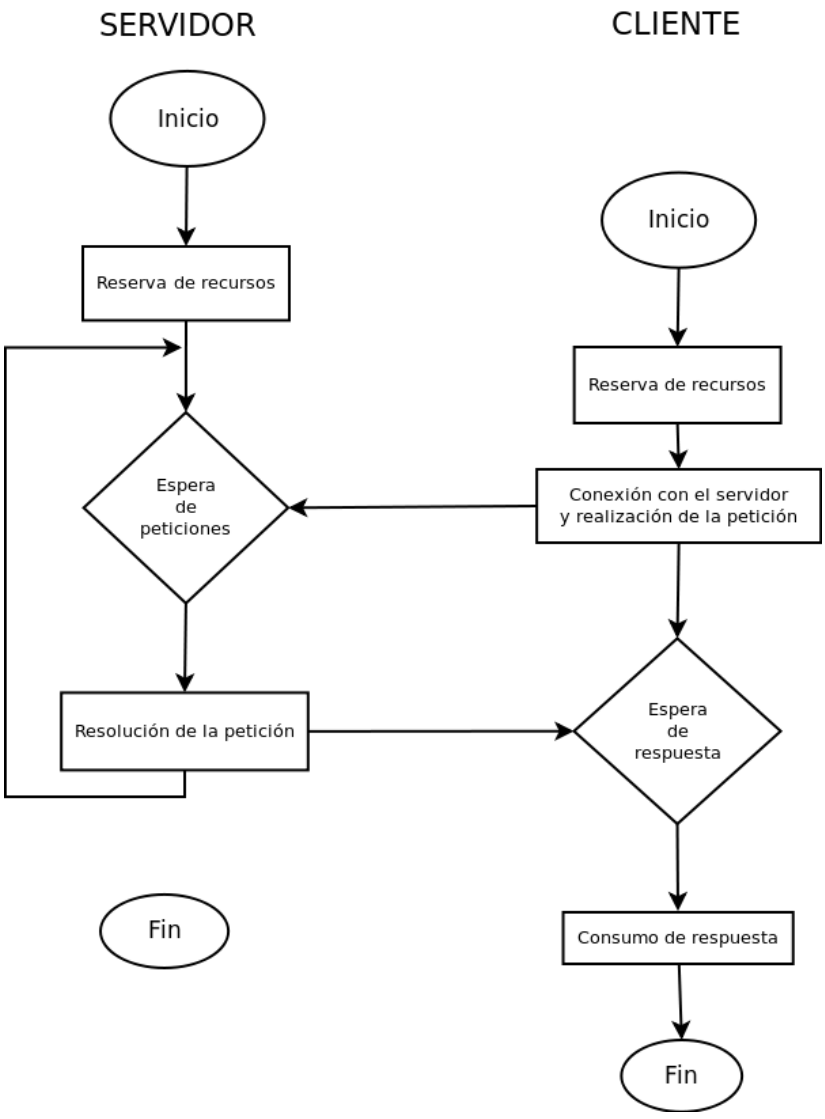
Este diseño de arquitectura se conoce como modelo Cliente-Servidor, y es sin duda el modelo más extendido y utilizado en el contexto de aplicaciones distribuidas.

En él, diferenciamos dos tipos de roles específicos, definidos según su papel en la comunicación.

Por una parte, tenemos un proceso encargado de escuchar peticiones para realizar una determinada acción. Este proceso recibe el nombre de **servidor**.

Por otra parte, el consumidor del servicio, el proceso que se encarga de enviar peticiones al servidor, se denomina **cliente**.

2.1 Modelo Cliente-Servidor



Como podemos apreciar en la figura, el proceso servidor pasa por los siguiente hitos:

1. Proceso de arranque.
    - Se inicia el proceso servidor, reservando lo que sea necesario.
2. Proceso de escucha de peticiones.
    - En el proceso de inicialización, el servidor reserva un puerto de la red del sistema para poder utilizarlo. Una vez listo, comienza la etapa de escucha de peticiones en él.

3. Resolución de la petición
4. Vuelta al paso 2

Mientras que el proceso cliente no está diseñado para ser iterativo:

1. Proceso de arranque
    1. Al igual que el servidor, el proceso cliente pasa por una serie de reservas e inicializaciones, pero en este caso no se necesita solicitar un puerto para la escucha.
2. Conexión con el servidor.
3. Realización de la petición.
4. Recepción de la petición.
5. Consumo de la petición.
6. FIN.

2.2 Python y Django

A la hora de tener que implementar un servicio web, disponemos de muchas opciones clasificadas principalmente en tres grupos.

Los Sistemas de gestión de contenidos o CMS ofrecen un contexto de alto nivel que permite obtener entornos muy elaborados rápidamente gracias a la extensa oferta de módulos de todo tipo. Con ellos el desarrollo es muy rápido, y para muchas aplicaciones no es necesario tener conocimientos de programación para utilizarlos. Pero su principal potencia es a la vez su mayor debilidad, al reutilizar módulos, y reconfigurarlos, se pierde bastante capacidad de personalización, que en ocasiones, nos obliga a tener que elaborar nuestros propios módulos o modificar en exceso los ya existentes.

Existen de muy diversos tipos, tanto por su funcionalidad específica orientada a un sector, como el lenguaje de programación en el que están escritos.

Zope, Wordpress, Joomla y Drupal son ejemplos de CMS.

En el contexto de este Proyecto de Fin de Carrera, se ha descartado esta opción.

En el extremo opuesto, tenemos la elaboración desde cero de nuestro servicio web. En esta opción tenemos el control absoluto de todo lo que hagamos. Se puede decidir el comportamiento de nuestro servicio en absolutamente todos los estados que queramos. Pero al igual que en el caso descrito anteriormente, toda esta potencia se convierte en un desarrollo mucho más largo y costoso. Debemos implementar la totalidad de la funcionalidad, aumentando considerablemente el tiempo de desarrollo, además de los posibles puntos de fallo, aumentando también el periodo de depuración del código.

En el contexto de este Proyecto de Fin de Carrera, también se ha descartado esta opción.

Por último, el grupo que queda por definir en esta descripción, y que como ya se habrá podido observar, se ha reservado deliberadamente para el final, son los frameworks de desarrollo.

Un framework de desarrollo se encuentra en el termino medio de esta enumeración de tipos, es un entorno de programación preparado para aportar cierta funcionalidad de base. De esta manera, ahorramos en tiempo de desarrollo, y a la vez tenemos potencia suficiente como para personalizar bastante nuestro nuevo servicio. Esa personalización está algo más limitada que en un desarrollo desde cero, pero sin duda llegamos al compromiso ideal para la elaboración de este proyecto. Por tanto, esta opción si es la que se ha elegido para este desarrollo.

Ejemplos de frameworks de desarrollo web son Ruby on Rails, Zend Framework y Django.

2.2.1 Python

Python es un lenguaje de programación interpretado. Está diseñado para ser bastante legible por parte de un humano, lo que facilita su rápida asimilación por los usuarios menos duchos en este lenguaje.

Es un lenguaje de alto nivel, lo que le aporta mucha potencia y flexibilidad, además, al estar enmarcado dentro de un paradigma orientado a objetos, y su popularidad, hace que existan innumerables librerías y demás APIs que aumentan en la medida de lo posible su rápido desarrollo y su rápida implantación para soluciones de todo tipo.

A pesar de ser un lenguaje orientado a objetos, en algunos libros se le considera un lenguaje multiparadigma, ya que es posible utilizarlo para hacer programación imperativa, y funcional, entre otras.

La declaración de las variables es implícita y dinámica. Tiene un tipado fuerte y es sensible a mayúsculas y minúsculas.

Python fue creado por Guido Van Rossum a finales de los ochenta como sucesor del lenguaje de programación ABC para el Centro de investigación para las matemáticas y la informática Centrum voor Wiskunde en Holanda. Su tipo de licencia ha cambiado a lo largo de los años para mantenerse compatible con las licencias GPL, actualmente, se distribuye bajo una licencia “Python Software Foundation License”. Por estos motivos, se considera a la “filosofía Python” muy similiar a la “filosofía Unix”.

2.2.2 Django

Django es un framework de desarrollo web de código libre. Está escrito en Python, y usa este lenguaje en su totalidad, incluso los ficheros de configuración de Django están escritos para ser interpretados en él.

Su idea principal es la de servir de base para que los desarrollos sean lo más rápidos posible, y de esta manera permite elaborar entornos bastante complejos en poco tiempo. Se centra en la filosofía de la programación orientada a objetos, promoviendo la reutilización de código y la conectividad de componentes.

Para continuar con esta filosofía, utiliza el paradigma Modelo-Vista-Controlador separando el desarrollo en estas tres divisiones.

Por un lado tenemos el Modelo, la información de nuestra aplicación tal cual se guarda y utiliza.

Por otro tenemos la Vista, la información enmaquetada para ser mostrada.

Por último, el Controlador se encarga de manejar los eventos y manipular la información.

Django es un framework muy completo, ya que además de tener una funcionalidad básica, que permite al programador abstraerse de temas no tan básicos como la gestión de sockets, la seguridad frente a temas generales como el SQL Inyection, etc, posee un pequeño servidor. Este servidor nos será muy útil durante el tiempo de desarrollo, pero no está recomendado para su posterior paso a producción ya que como los propios desarrolladores de Django dicen: “ellos no se dedican a hacer servidores web” y una vez más, haciendo gala de su filosofía de reutilización de módulos, delegan esta tarea a alguno de los grandes servidores que todos conocemos, como Apache, entre otros.

Django, inicialmente fue concebido para ser un gestor de noticias en la World Company, en Kansas, pero fue liberada bajo una licencia BSD en julio de 2005.

2.3 Bases de datos

Una base de datos es una agrupación de información con algún tipo de relación entre sí, que se almacenan siguiendo una estructura prediseñada para tratar de mejorar su manejo. Dentro del contexto de la informática, en el que nos estamos moviendo a lo largo de este documento, un Sistema Gestor de Base de Datos es un software diseñado para almacenar, y posteriormente consultar información estructurada para mejorar lo máximo posible su accesibilidad.

Aunque existen diversos modos de clasificarlas, su diferencia más importante suele residir en el tipo de descripción del almacén de la información y de los procedimientos de almacenamiento y recuperación de la misma, es decir, el modelo de datos que utiliza.

Existen varios tipos de modelos de datos, como por ejemplo los “jerárquicos”, “de red”, “transaccionales”, etc. Pero sin duda, el que más uso tiene en la actualidad es el modelo de datos relacional. Tanto es así que en mucha documentación podemos encontrar ejemplos de SGBBDD divididos en “relacionales” y “no relacionales”.

Debe su fama a que, al no contemplar la forma de almacenar los datos en sí, resulta ser uno de los modelos de datos más fáciles de entender, especialmente para los usuarios esporádicos. Ya que se definen los datos en tablas, especificando la relación entre las mismas.

Fue definido por Edgar Frank Codd para IBM en 1970.

El lenguaje convenido, y más extendido para la consulta en Bases de Datos Relacionales es el SQL (Structured Query Language o Lenguaje Estructurado de consultas).

2.3.1 MySQL

Es un Sistema de Gestión de Bases de Datos relacional, multiusuario y multihilo.

Desarrollado en C y C++, es una solución generada como software libre.

2.3.1 MySQL

Durante mucho tiempo ha sido la principal alternativa al SQL Server de Oracle. La empresa desarrolladora de este software fue inicialmente comprada por Sun Microsystems y posteriormente por Oracle, su dueño actual. Por este motivo, hacemos referencia a principal alternativa en pasado, ya que en la actualidad no supone una competencia directa entre empresas.

Posee una gran variedad de soluciones que se adaptan a los entornos en los que sea necesario su uso. Bien sea teniendo una base de datos local, una remota, una solución en clúster, etc. Además tiene sus propias herramientas de monitorización.

2.3.2 SQLite

Es un Sistema de Gestión de Bases de Datos relacional creado por Richard Hipp.

Al igual que en el caso anterior, está escrita en C y es un proyecto libre.

La principal característica que en la SQLite se diferencia de otros Sistemas de Gestión de Bases de Datos es que en este caso la funcionalidad de acceso a los datos no se ofrece como un servicio independiente, sino que se "incrusta" dentro del software que estemos desarrollando, utilizando las librerías oportunas.

Otro dato a tener en cuenta de este sistema es que tanto los datos como la definición de modelos están diseñados para almacenarse en un mismo fichero. Esta cualidad hace que se utilice internamente en muchos programas sin que seamos conscientes de ello. Como, por ejemplo, las cookies en el navegador Mozilla Firefox.

2.4 Servidores Web

En el mundo del Web, es bastante común que utilicemos una arquitectura de tipo Cliente-Servidor. Como ya se ha hablado de este tipo de arquitecturas en este mismo apartado, aprovecharemos ese conocimiento para introducir el concepto de servidor Web.

Un servidor Web es un software que se encarga de estar permanentemente a la espera de algún tipo de petición Web para ofrecer un servicio a los posibles consumidores de los recursos que éste ofrece.

Denominamos petición Web a toda aquella solicitud que sigue el protocolo HTTP, y por extensión su versión “segura” la HTTPs.

Por último HTTP, Hypertext Transfer Protocol o protocolo de transferencia de hipertexto. Es un convenio por el cual se define una sintaxis y una semántica especiales para ser utilizadas dentro de las comunicaciones consideradas como “Web”.

Actualmente existen diversas soluciones software que cubren este tipo de necesidades. Algunos ejemplos de los más populares son: Apache, Cherokee y Lighttpd.

Dentro del contexto de este proyecto, queremos hacer especial mención a **Apache**, uno de los más extendidos, y que de alguna manera ha servido de ejemplo a seguir para los desarrollos posteriores de otras soluciones similares.

Es un servidor Web de código abierto desarrollado por la Apache Software Foundation.

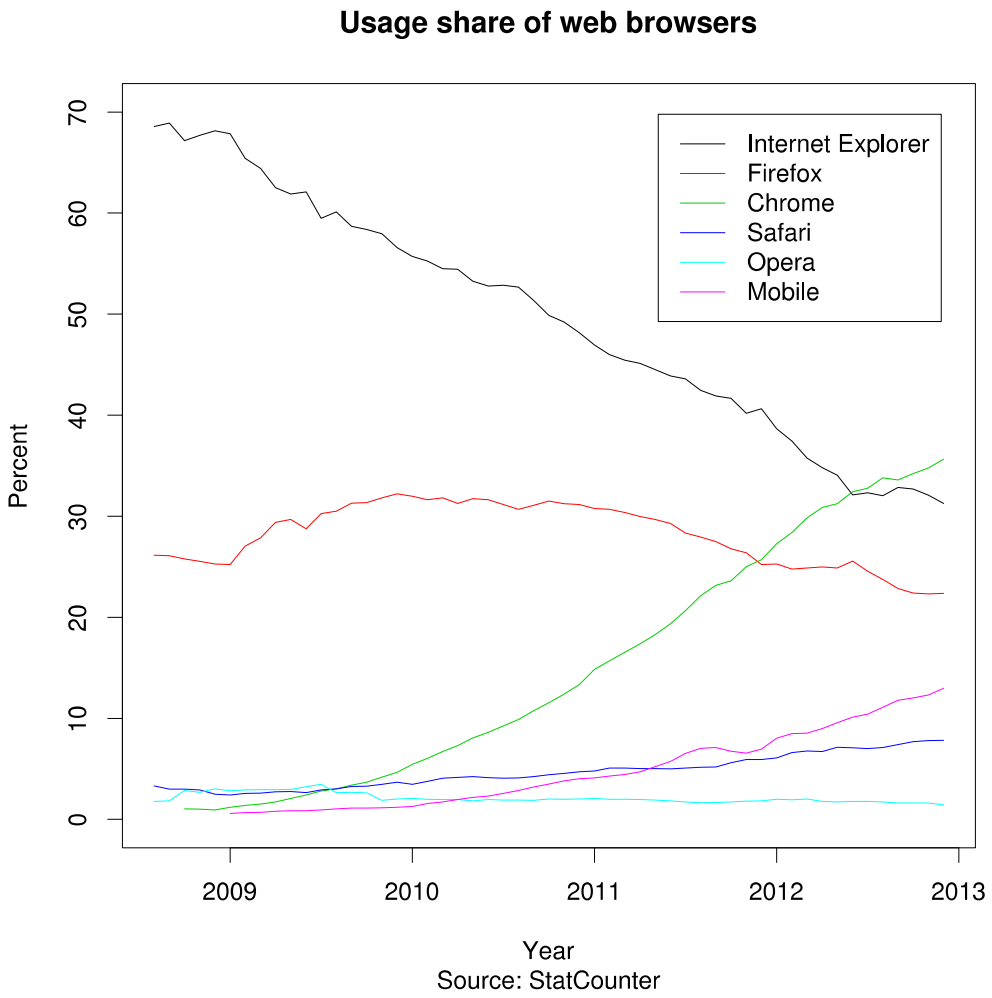
Posee un desarrollo muy modular que permite crear multitud de personalizaciones capaces de cubrir una gran cantidad de funcionalidades. Un claro ejemplo de las bondades de este desarrollo modular es el modulo de WSGI. Gracias a este módulo podemos ejecutar un proyecto Django a través del servidor Apache.

2.5 Navegadores Web

Un navegador Web es una aplicación que se encarga de servirnos de interfaz a la hora de visualizar contenido a través del protocolo HTTP o HTTPS. A lo largo de los años, los navegadores han ido avanzando y cada vez soportan más extensiones, y son capaces de reproducir una gran cantidad de contenidos, pero inicialmente fueron diseñados para visualizar texto.

Uno de los más conocidos y antiguos es Netscape Navigator, a pesar de no ser el primero, generó un gran impacto en el mercado, y despertó la curiosidad de Microsoft en la elaboración de navegadores.

2.5 Navegadores Web



En la actualidad existe una gran variedad de navegadores web, lo que hace que exista una gran oferta en la que elegir el que más se asemeje a nuestras necesidades, o simplemente elegir el que más nos guste. Esta guerra de navegadores, que compiten por un hueco en el mercado, sin duda favorece que cada vez sean más robustos y estén mejor diseñados. Y nos permite poder disfrutar de ellos en una gran variedad de plataformas.

Para este Proyecto de Fin de Carrera, nos interesa especialmente este punto, y aprovecharemos que existen navegadores en una gran variedad de dispositivos, principalmente en este contexto en ordenadores personales, y en dispositivos móviles, ya sean smartphones o tablets.

2.6 Identificación de objetos

En la actualidad, existen diversas formas de etiquetar objetos con algún tipo de identificador único. Sin duda, el más extendido es el código de barras, pero existen otros sistemas distintos. A continuación pasamos a hablar un poco más sobre el código de barras y los RFID

2.6.1 Códigos de barras

Los códigos de barras están presentes en nuestro día a día, pero ¿qué son realmente?.

Un código de barras es una representación gráfica de cierta información, utilizando una convención de líneas de diferente grosor y normalmente en dos colores diferentes, buscando un alto contraste.

Su primera patente es de 1952, pero no empezó a utilizarse comercialmente hasta 1980.

Un código de barras nos permite asignar un identificador único a un objeto. En la actualidad se utilizan también códigos de barras de dos dimensiones. Un ejemplo de esto son los códigos QR, que se representan en una estructura cuadrada, y pueden almacenar mucha más información.

En el contexto de este proyecto, nos centraremos en los códigos de barras como identificadores únicos.

Una parte, que indiscutiblemente es casi más importante que el código de barras, es el dispositivo capaz de operar con él, y que puede traducir a un lenguaje legible por un ser humano lo que el código muestra.

Este dispositivo recibe el nombre de escáner de códigos. Se emite una linea con un láser, y se analiza el rebote de la luz contra el código, las células fotosensibles del dispositivo analizan las diferencias de grosor entre las lineas. Gracias al alto contraste de los códigos, se absorbe de forma diferente la luz emitida. Esto se traduce en una señal eléctrica, que posteriormente consigue convertirse en la información cifrada en dicho código.

2.6.1 Códigos de barras

En la actualidad, no es necesario disponer de ese dispositivo para identificar un código de barras. Con una cámara podemos capturar una imagen del código de barras, y con un software de tratamiento de imágenes podremos descifrar el código.

Existen numerosas aplicaciones capaces de hacer esto desde un smartphone.

En este proyecto nos aprovecharemos de este hecho, y utilizaremos como parte de la aplicación.

2.6.2 RFID

Radio Frequency IDentification, o en español identificación por radiofrecuencia, es un sistema de almacenamiento y recuperación de datos por radiofrecuencia.

Existen principalmente dos tipos de tarjetas RFID:

- Pasivas: se fabrican en tarjetas, pegatinas, o cubiertas por una pequeña capa de plástico. Tienen aproximadamente el tamaño de una moneda de 2 euros, y poseen un alcance muy reducido ya que funcionan con una pequeña corriente enviada por el lector que hace reaccionar el circuito de la tarjeta y emite la información.
- Activas: al contrario que las pasivas, éstas tienen su propia fuente de alimentación, pudiendo emitir su información sin necesidad de recibir una reacción del lector. Esto produce una señal más intensa que puede ser captada a mayor distancia. El principal inconveniente viene dado por el mantenimiento de las mismas, que si bien tienen una larga autonomía, al llevar una batería siempre llegará el momento de tener que sustituirlas. Su tamaño es algo mayor que las pasivas, ya que necesitan más elementos.

En la actualidad, cada vez más smartphones llevan incorporada una tecnología compatible con este tipo de dispositivos. Es lo que conocemos como el Near Field Communication, o en español Comunicación de Campo Cercano, que es una tecnología inalámbrica, que nos permite tener una pequeña comunicación a una pequeña distancia.

2.7 Android

Android es un sistema operativo basado en Linux. Principalmente pensado para dispositivos móviles como smartphones y tablets, aunque cada vez está más presente en otros soportes como portátiles, netbooks, smartTVs, etc.

Al igual que en las versiones de los distintos sistemas operativos Linux, las versiones de Android, siguen un patrón de letras, siguiendo un orden alfabético. En este caso, se fija la primera letra de la versión y se utilizan nombres de postres para aportarle un nombre más fácilmente recordable, de esta manera actualmente tenemos:

- A: Apple Pie (v1.0) Tarta de Manzana. Lanzado el 23 de septiembre de 2008.
- B: Banana Bread (v1.1) Pan de plátano. Lanzado el 9 de febrero de 2009.
- C: Cupcake (v1.5) Magdalena glaseada. Lanzado el 30 de abril de 2009.
- D: Donut (v1.6). Lanzado el 15 de septiembre de 2009.
- E: Éclair (v2.0-v2.1). Dulce francés parecido a un pepito o petisú. Lanzado el 26 de octubre de 2009.
- F: Froyo (v2.2) Abreviatura de Frozen Yogurt, yogurt helado. Lanzado el 20 de mayo de 2010.
- G: Gingerbread (v2.3) Pan de jengibre. Lanzado el 6 de diciembre de 2010.
- H: Honeycomb (v3.0-v3.1-v3.2) Panal de miel. Lanzado el 22 de febrero de 2011. Menos conocido porque fue exclusiva para tablets.
- I: Ice Cream Sandwich (v4.0) Sandwich de helado. Lanzado el 19 de octubre de 2011.
- J: Jelly Bean (v4.2) Judia de gominola. Lanzado el 13 de noviembre de 2012.
- K: aún por confirmar.

2.7 Android

El código fuente está accesible, y puede ser consultado ya que está disponible para su consulta. Ya que se distribuye bajo una licencia GPLv2.

Inicialmente desarrollado por la empresa Android Inc, posteriormente fue comprada por Google en 2005. Actualmente es desarrollado por el conjunto de empresas Open Hanset Alliance, entre las que se encuentra Google.

3 Diseño del sistema

Para la elaboración de este sistema, se ha pensado en una arquitectura del tipo cliente servidor. Se pretende enmarcar esta aplicación dentro del contexto en el que será usada, y se quiere tratar de aprovechar en la medida de lo posible todo tipo de desarrollo existente. De esta manera se puede aumentar la portabilidad de la aplicación y por tanto aumentar el uso y versatilidad de la misma, pudiendo llegar a más usuarios.

Ya que tenemos dos desarrollos, por así decirlo, utilizaremos dos metodologías de desarrollo diferentes.

Por un lado para la parte de CPDpocket más relacionada con el servidor, seguiremos un proceso de desarrollo iterativo e incremental, que se asemejará bastante a un desarrollo en espiral.

Por otro, para la parte del desarrollo orientada al cliente, seguiremos un proceso de desarrollo rápido basado en prototipos. Ésto nos permitirá tener mucha flexibilidad en esta parte, viendo in situ los avances que realicemos y estudiando las posibles mejoras.

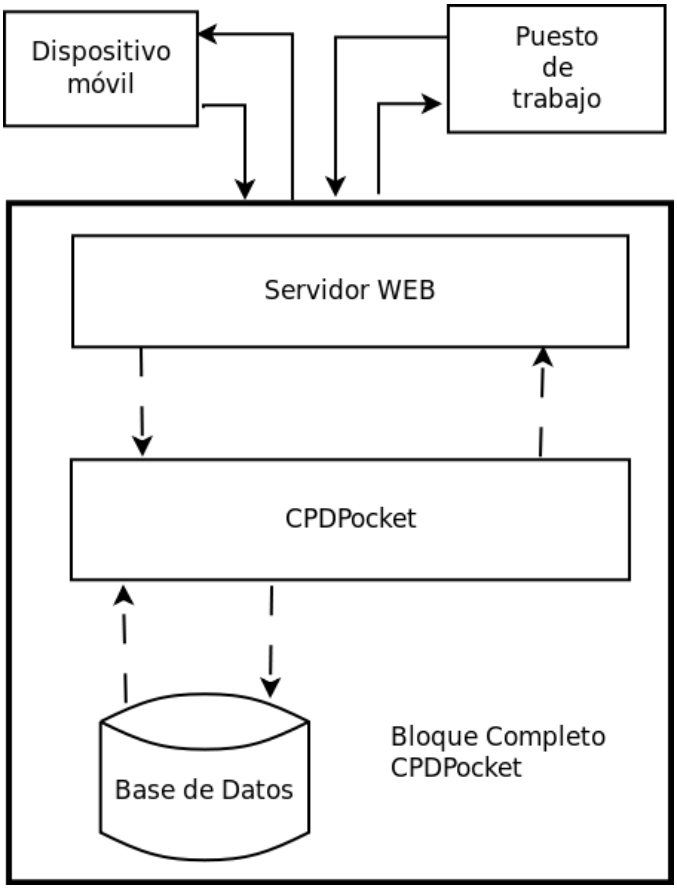
3.1 Servidor

En este apartado se centra todo el desarrollo puro de este Proyecto de Fin de Carrera, delegando la parte de cliente en el más que poblado mundo de los clientes http.

Cómo el lector puede sospechar, gracias al apartado de estado del arte, se ha elegido Django para elaborar esta parte. Además de por todas sus virtudes, se ha elegido pensando en la familiaridad que todo administrador de sistemas que se precie, debe tener con el mundo del scripting. Además Python cada vez es más usado para la elaboración de scripts comunes del sistema, lo que sin duda podría facilitar el camino a los usuarios potenciales, que quisieran añadir funcionalidad a CPDPocket y complementar la herramienta para su día a día.

3.1 Servidor

El servidor consta básicamente de tres apartados:



3.1.1 Base de datos

La base de datos albergará la información estática de la aplicación. Se requiere hacer un estudio de las necesidades más comunes que se tendrán en un CPD para tratar de elaborar un diseño lo más abierto posible.

Además necesitamos sentar las bases para poder utilizar la información estática almacenada para generar la información dinámica que hará más rica nuestra aplicación.

3.1.1 Base de datos

Se ha decidido utilizar tres tipos de tablas distintas que se corresponden con los siguientes objetos del mundo real:

- **Máquina virtual:** es la unidad más pequeña de la base de datos, ya que contiene solamente información estática.

  En ella se guardarán datos como:

  - Código: identificador de la máquina
  - Nombre: nombre de la máquina, referenciando el hostname que ofrece en la red de cara a la parte dinámica de la aplicación.
  - Notas: En este apartado se podrá anotar cualquier apreciación que quiera hacerse sobre la máquina, que pueda resultar útil para el técnico.
  - Credenciales: usuarios o conjunto de usuarios con sus correspondientes contraseñas que serán de gran ayuda para el técnico si necesita hacer una manipulación directa sobre la máquina
  - Proyecto: para poder vincular la máquina a un proyecto específico en el que está siendo utilizada, y poder así llevar un control, y facilitar las búsquedas.
  - Responsable: Persona que es responsable de la máquina, con la que se deberá tratar en caso de tener que hacer un cambio significativo.
  - Lista de puertos ofrecidos por la máquina, para poder tener controlada la funcionalidad.
  - Servidor físico en el que está alojada la máquina virtual.

3.1.1 Base de datos

- **Servidor:** el perfil de un servidor tiene pocas diferencias frente al de una máquina virtual, pero se considera más complejo. Esto es así porque además de tener las opciones de una máquina virtual, también tiene la posibilidad de ser un servidor de máquinas virtuales, es decir un contenedor de máquinas.

  La información que guardaremos de un servidor por tanto es muy similar a la que tenemos de una máquina virtual, salvo por la única característica física, sus dimensiones:

  - Código: identificador de la máquina
  - Nombre: nombre de la máquina, referenciando el hostname que ofrece en la red de cara a la parte dinámica de la aplicación.
  - Notas: En este apartado se podrá anotar cualquier apreciación que quiera hacerse sobre la máquina, que pueda resultar útil para el técnico.
  - Credenciales: usuarios o conjunto de usuarios con sus correspondientes contraseñas que serán de gran ayuda para el técnico si necesita hacer una manipulación directa sobre la máquina
  - Proyecto: para poder vincular la máquina a un proyecto específico en el que está siendo utilizada, y poder así llevar un control, y facilitar las búsquedas.
  - Responsable: Persona que es responsable de la máquina, con la que se deberá tratar en caso de tener que hacer un cambio significativo.
  - Lista de puertos ofrecidos por la máquina, para poder tener controlada la funcionalidad.
  - Unidades de rack que ocupa la máquina. Se da por hecho que la profundidad es estándar, aunque a priori no es cierto, es bastante común tener la gran mayoría de las máquinas de la misma profundidad, variando solamente la altura. Para esta altura se utilizará el convenio de “us” o “rack units” que equivalen a 44,45 mm.
  - Por último también se hará referencia al rack en el que está este servidor físico.

- **Rack:** se utilizará para guardar la información relativa a la estructura en si. Para el contexto inicial de la aplicación no será más que el almacén de servidores. Y servirá para ubicar geográficamente la máquina.

  Aquí guardaremos información del tipo:

  - Código: un identificador único del rack.
  - Nombre: un identificador que sea más fácil de recordar para un ser humano, si se necesita.
  - Notas: al igual que en las máquinas, quizá sea interesante guardar alguna apreciación del servidor, como por ejemplo que se trata de un servidor con doble circuito eléctrico, o con un sistema de alimentación continua en lugar de alterna.
  - Ciudad: de esta manera permitiremos tener en la misma aplicación registradas máquinas de distintas sedes, si fuera necesario.
  - Unidades totales de rack disponibles: esta información nos será útil a la hora de ver el hueco que hay disponible a la hora de ubicar una máquina nueva, o bien a la hora de diseñar una reestructuración.

3.1.2 CPDPocket

3.1.2.1 Interfaz

Se quiere hacer un sistema completo y polivalente. Para ello se diferenciarán dos interfaces principales.

Una de ellas será simple y de fácil lectura, y estará orientada a la visualización en dispositivos móviles. A pesar de que cada vez son más potentes, damos por hecho que cuando se está utilizando la aplicación desde el móvil es durante una situación en la que se tiene algo de prisa y cualquier cosa que nos evite que tener que hacer un zoom selectivo hacia lo que queremos ver, o tener que descartar información, será de agradecer por parte del usuario.

3.1.2 CPDPocket

Por el contrario, la otra, más orientada a ser vista desde dispositivos más potentes, será más atractiva y con un diseño más parecido al que poder encontrar en cualquier otra página web de Internet.

3.1.2.2 Tipos de URL

Para poder acceder al servidor, y a sus distintas funcionalidades necesitamos definir una serie de urls que nos facilitarán tanto los accesos como la comprensión de la aplicación:

- Mostrar información estática.
- Mostrar información dinámica.
- Añadir nuevo registro.
- Editar información estática.
- Búsquedas en la base de registros.
- Página de bienvenida.
- Página de error para ayuda, por un mal uso.

3.1.3 Servidor web

Para servir la aplicación, se utilizará un servidor web ajeno a Django, consiguiendo de esta manera una mejor gestión. Según la documentación de Django, ellos no se dedican a desarrollar servidores, por tanto recomiendan encarecidamente que se utilice su framework para desarrollar aplicaciones, pero que se configure adecuadamente un servidor web dedicado a tal efecto para ofrecer el servicio.

3.2 Cliente

Para la parte cliente de nuestra aplicación utilizaremos básicamente cualquier tipo de navegador común. Gracias a esto, podemos aprovechar la robustez de un desarrollo ajeno al nuestro, que ha sido probado en profundidad.

Ya se ha descrito detalladamente la gran variedad que tenemos de este tipo de software en el mercado, así que no añadiremos nada más.

Además se quiere tratar de aportar un paso más para facilitar la vida del usuario. Para ello intentaremos buscar también dentro de la oferta que se encuentra actualmente en el "play store" de Android, para tratar de adecuar algún desarrollo de identificación de etiquetas de los descritos en el "Estado del Arte" para tratar de aportar robustez y colaboración entre aplicaciones, para este desarrollo.

3.3 Funcionalidad

- **Máquina virtual**
  - **Parte estática**

    En la parte estática correspondiente a la máquina virtual se guardaran datos estáticos importantes para la identificación y manipulación de la máquina por un humano, además de algunos valores especiales necesarios para poder ampliar la información de manera dinámica. Los datos que se almacenarán de la máquina virtual se han descrito en profundidad en el apartado de diseño de la base de datos.

    Se permite:

    - Añadir una nueva máquina virtual.
    - Editar los datos de una máquina existente.
  - **Parte dinámica**

    Para completar la información estática, se utilizarán principalmente dos métodos:

    - Al ampliar la información se utilizará el "nombre" de la máquina, dando por hecho que se trata del hostname anunciado en la red por la misma para:
      - Hacer una resolución de nombres, para mostrar el valor de la IP asociada a ese hostname. Gracias a este añadido podremos detectar un posible fallo de IP incorrecta, o no resuelta, o quizá ofrecer al usuario la información necesaria para saber que esta máquina se encuentra en una red especial.

- Se realiza un diagnóstico de encendido, por medio de una respuesta a un ping.

- Además, existirá un campo de la información estática de la máquina, que nos permitirá poder ofrecer una información más personalizada. En este campo, mediante una sintaxis especial, mostrará los diferentes puertos de la máquina que se quiere comprobar que estén abiertos de cara a la red.

- **Servidor Físico**

  - **Parte estática**

    Al igual que en el caso de un máquina virtual, en la parte estática correspondiente a al servidor físico, se guardaran datos estáticos importantes para la identificación y manipulación de la máquina por un humano, además de algunos valores especiales necesarios para poder ampliar la información de manera dinámica. Los datos que se almacenarán del servidor físico se han descrito en profundidad en el apartado de diseño de la base de datos.

    Se permite:

    - Añadir un nuevo servidor físico.

    - Editar los datos de un servidor físico existente.

  - **Parte dinámica**

    Al igual que en la máquina virtual, para completar la información estática, se utilizarán principalmente dos métodos:

    - Al ampliar la información se utilizará el "nombre" de la máquina, dando por hecho que se trata del hostname anunciado en la red por la misma, para:

      - Hacer una resolución de nombres, para mostrar el valor de la IP asociada a ese hostname. Gracias a este añadido podremos detectar un posible fallo de IP incorrecta, o no resuelta, o quizá ofrecer al usuario la información necesaria para saber que esta máquina se encuentra en una red especial.

- Se realiza un diagnóstico de encendido, por medio de una respuesta a un ping.

- Además, existirá un campo de la información estática de la máquina, que nos permitirá poder ofrecer una información más personalizada. En este campo, mediante una sintaxis especial, mostrará los diferentes puertos de la máquina que se quiere comprobar que estén abiertos de cara a la red.

- Por último, al tratarse de un servidor físico, en el caso de que sea un servidor dedicado de máquinas virtuales, también tendremos esa información adicional, listando la totalidad de las máquinas registradas para este anfitrión.

Como ya se ha podido apreciar, la funcionalidad dinámica de un servidor físico, es idéntica a la funcionalidad dinámica de una máquina virtual, pero con el añadido de que un servidor físico puede albergar máquinas virtuales.

- **Rack**
  - **Parte estática**

    En la parte estática se guardaran los datos básicos relativos a esta estructura. Datos que se han descrito con mayor detalle en el apartado anterior.

    Se permite:

    - Añadir un nuevo rack
    - Editar los datos de un rack existente

  - **Parte dinámica**

    La parte dinámica que esta aplicación puede aportar como valor añadido a la información estática para un rack es sin duda, la posibilidad de mostrar una lista completa de todas las máquinas que están alojadas en él. Dado que se trata de una estructura inerte totalmente pasiva, no se encuentran a priori más posibilidades de complementación de esta información.

3.3 Funcionalidad

En resumen:

- Gestión incremental y de modificación de los registros de la base de datos.
- Visualización de los registros.
- Aumento de los datos estáticos mediante consultas en tiempo real que complementen la información.

4 Implementación

Una vez que el desarrollo está en un punto bastante maduro, paso a cumplimentar este apartado de implementación para así poder dar un punto de vista más constructivo sobre él.

Finalmente el desarrollo ha tenido dos partes bien definidas, una en la parte cliente, que ha sido más investigación que desarrollo en si mismo, y otra dedicada a la parte servidora, que ha tenido tanto desarrollo como investigación.

4.1 Servidor

Sin duda esta es la parte más elaborada de este Proyecto de Fin de Carrera, sin menospreciar por ello la parte de investigación, ni la de las demás partes.

4.1.1 Base de datos

Se ha implantado exactamente el modelo de base de datos que se ha descrito en el apartado de diseño.

Para complementar algo la información que se ha descrito en ese apartado, en éste pasaremos a hablar sobre la solución de base de datos que se ha utilizado durante el desarrollo de la aplicación.

Para seguir con la ideología de este proyecto, y por la proximidad a esta solución que tiene el ideador de CPDPocket, se ha elegido MySQL.

Se ha tomado esta decisión, entre otras cosas, por la posibilidad de extraer este servicio a un nodo ajeno al del servidor de CPDPocket. De esta manera se facilita la escalabilidad de la solución, tanto por la parte de la replicación de nodos para la base de datos, como por la parte de la replicación de nodos para el servidor de la aplicación. Manteniendo así el esquema mostrado anteriormente que representa la arquitectura del sistema.

En cualquier caso, al tratarse de un proyecto Django, no supone demasiado problema utilizar cualquiera de las otras soluciones ofrecidas por él. Si por algún motivo, no se quiere, o no se puede utilizar MySQL, se recomienda utilizar una solución que simplifique todo esto, aunque con ella se renuncia a parte de la escalabilidad descrita, un ejemplo de esto es SQLite.

4.1.2 CPDPocket

Para describir la implementación de este apartado, y ya que se trata de un servicio ofrecido a través de HTTP, se realizará una explicación siguiendo el esquema de URLs definidas en el apartado de Diseño y descrito más en profundidad en el anexo de Modelo de Casos de Uso.

- **Mostrar información estática**.

  La URL escogida para este fin es "/show/code/<IDENTIFICADOR>".

  Si utilizamos esta URL, generamos una consulta en la base de datos que nos mostrará los datos vinculados al <IDENTIFICADOR>. Se realiza una consulta y se muestran los datos formateados. Para facilitar el uso del sistema, en caso de error, se ha habilitado la posibilidad de registrar este nuevo <IDENTIFICADOR>, mediante un enlace autogenerado apuntando a otra URL del sistema, descrito más adelante. De esta manera la siguiente consulta de este <IDENTIFICADOR>, si que aparecerá.

- **Mostrar información dinámica**.

  La URL escogida para este fin es "/showComplet/code/<IDENTIFICADOR>".

  En este caso, el funcionamiento es algo más complejo que en el caso anterior, aunque en la parte inicial es idéntico.

  Inicialmente partimos de la misma base, según se recibe una petición, se van a buscar los datos estáticos a la base de datos relativos a <IDENTIFICADOR>. Una vez que se tienen estructurados, se pasa a "calcular" la parte dinámica.

  En la actualidad, la parte dinámica consta de:

  - **Identificación de IP**: utiliza el campo "nombre de la máquina" para hacer una consulta en la red y hacer una resolución DNS, que nos traduzca el nombre de la máquina en la IP correspondiente. De esta manera, soportaremos tanto máquinas que tengan una IP estática, como una configuración de red mediante DHCP con IP dinámica ya que se hace la resolución del nombre en el momento de la consulta.

    Esta resolución se hace utilizando la librería de python "dns.resolver" incluida en el paquete "python-dnspython"

- **Respuesta a Ping**: Realiza un ping desde la máquina en la que está instalado CPDPocket, hacia la máquina de la que se está obteniendo la información. Esta vez se utiliza la shell del sistema, se lanza un único ping, y en caso de obtener respuesta, se pinta una bola verde, en caso contrario, una bola roja. Se utiliza la shell porque no se ha encontrado ningún modulo simple que resuelva esta necesidad.
  - Se debe valorar la no respuesta a un ping como:
    - Un posible fallo en la máquina, que la mantiene apagada
    - Un posible fallo de conectividad que la mantiene desconectada.
    - Que la máquina no esté configurada para responder a ping
    - Que la red no permita el tráfico de "ping" por alguna razón

  Por estas razones, en la aplicación en lugar de mostrar esta información como "encendida" se muestra como "respuesta a ping".

- **Comprobación de puerto**: Esta funcionalidad es sin duda la más completa del sistema. Ya que ofrece una información bastante completa, y es suficientemente flexible como para ser útil en cualquier sistema.

  Para utilizarlo, se debe seguir una sintaxis especial, diseñada para que el propio usuario pueda definir tantos puertos como quiera, además del nombre asociado a él. De esta manera, se puede crear un perfil personalizado para cada máquina, mostrando varios puertos, y lo que es más importante, mostrando el nombre del servicio que el usuario haya elegido como más significativo en su propio contexto. La sintaxis escogida es simple y consta de dos partes "Nombre que identifique el servicio"$"puerto"%. Esto se puede repetir tantas veces como sea necesario para tener controlados tantos puertos como se deseen.

  De nuevo, para identificar si el puerto está escuchando o no utilizamos imágenes de bolas rojas o verdes.

4.1.2 CPDPocket

En este caso, para implementar esta funcionalidad, se ha utilizado nuevamente la shell del sistema. Esta vez sí que se han encontrado módulos de Python que hicieran algo similar, incluso se ha desarrollado un pequeño módulo mediante sockets con los que conectarse al puerto destino y así comprobar que hay un servicio escuchando en el puerto. El motivo por el que se ha escogido utilizar el comando "nmap" de la shell, en lugar de las posibles soluciones que ya hemos descrito, es que en una vez implementado el sistema, tras probarlo, comprobamos que el funcionamiento no era el correcto. En máquinas que tenían configurado un sistema de seguridad, basado en restricciones del acceso mediante acumulación de peticiones incorrectas, las consultas de las demás soluciones eran detectadas como uso indebido del puerto y por tanto a la larga bloqueaban el acceso a la máquina.

Un ejemplo de esto es "DenyHost", un servicio que se configura principalmente para aumentar la seguridad de un servidor de SSH. En este servicio se configuran una serie de contadores. Se rastrean los ficheros de acceso a la máquina incrementando esos contadores según se van detectando intentos de acceso a la máquina. Se consideran intentos fallidos de acceso a la máquina los que se han producido bien por contraseña incorrecta, o bien por no aportación de la misma. Una vez que se ha alcanzado el número máximo, se procede a una restricción del acceso a la máquina desde la ip origen de estos intentos. La severidad de esta restricción depende de la configuración de este módulo.

En resumen, utilizando el comando "nmap" conseguimos tener acceso a esa información sin modificar la máquina, que es justo lo que necesitamos.

Por último, consideramos que hay un servicio escuchando en el puerto solicitado cuando se obtiene un mensaje de "open" al realizar el escaneo. Los lectores que conozcan el funcionamiento de "nmap" sabrán que podríamos encontrarnos con el estado "filtered" y también podríamos estar en el caso de que el servicio esté escuchando, pero se ha decidido considerar unicamente el caso de "open" como correcto.

Los usuarios del sistema deberán saber esto para interpretar la información de manera correcta.

- **Lista máquina contenidas**: Finalmente, la última característica dinámica que se ofrece en este sistema es la de mostrar en forma de tabla resumen los datos estáticos de todas las máquinas albergadas en esta estructura. Facilitando enlaces para poder acceder a más información.

Esta es la descripción de toda la funcionalidad dinámica añadida en el estado actual de CPDPocket. Como se puede suponer, toda la funcionalidad no está disponible para todas las estructuras. Es decir, "listar máquinas contenidas" sólo estará disponible en estructuras especiales como son un Rack, o un servidor de máquinas virtuales, las dos únicas capaces de almacenar máquinas en su interior. Por otra parte, esta característica será la única característica dinámica que podrá mostrar un Rack, ya que se trata de una estructura inerte.

- **Añadir nuevo registro**.

  La URL escogida para este fin es "/add/code/<IDENTIFICADOR>".

  Esta opción está accesible, si se escribe directamente esta URL en el navegador. En caso de no forzar de esta manera, que se nos brinde la posibilidad de añadir un nuevo registro al sistema, la única forma en la que la aplicación nos ofrece la opción es al tratar de mostrar un código que no esté registrado en la base de datos.

  Se considera el procedimiento natural, en el que al tratar de ver los datos relativos a una máquina, al detectar que ésta no está registrada, se puedan añadir. De esta manera, se muestra una página de error, informando de que el código buscado no existe, y posteriormente se ofrece un enlace en el que añadirlo.

  En la implementación de esta funcionalidad, lo que se hace es, una vez solicitada la adición del nuevo código, éste se añade automáticamente sin información alguna, y acto seguido el usuario es redirigido a la parte de edición de datos de un registro ya añadido.

4.1.2 CPDPocket

- **Editar información estática**.

  La URL escogida para este fin es "/edit/code/<IDENTIFICADOR>".

  Como el propio título indica, en esta URL podremos modificar la información estática relativa a un registro ya registrado.

  En este caso, se permiten modificar absolutamente todos los campos, salvo el de código de identificación. Es decir, se puede pasar de una estructura a otra, como, por ejemplo, de un registro de Rack a un registro de Máquina Virtual, pero el código de identificación del registro quedará fijado desde el principio.

  En este caso, esta opción se ofrece, de manera indirecta al hacer una inserción, como ya se ha comentado en el apartado anterior, y de manera directa mientras se está mostrando la información asociada a un registro.

- **Búsquedas en la base de registros**.

  La URL escogida para este fin es "/search".

  Esta funcionalidad está más orientada al uso desde un puesto de trabajo que aporte mejores condiciones que un dispositivo móvil, pero se puede usar en ambos.

  Permite hacer búsquedas en los registros que ya han sido almacenados, permitiendo buscar en diferentes campos, aunque sólo se permite uno por búsqueda. En la base de datos permitimos buscar por: código, nombre de máquina, ciudad, proyecto y responsable. Además se permite hacer dos tipos de búsqueda: una búsqueda por valor exacto, que sólo reconocerá los campos que coincidan exactamente con el valor facilitado en la búsqueda; y la búsqueda por defecto, que buscará que el valor facilitado esté contenido en el campo elegido sin importar que existan caracteres adicionales.

- **Página de bienvenida**.

  La URL escogida para este fin es "/".

  Esta página simplemente muestra un resumen de los posibles enlaces que se pueden utilizar, y ya que no aporta demasiado valor, no haremos más hincapié en ella.

- **Página de error para ayuda, por un mal uso**.

  La URL escogida para este fin es "/*" refiriéndose a que cualquier URL que no encaje en las expresiones regulares anteriores se considerará una petición de ayuda.

  En este caso, al igual que en el caso anterior, se trata de un recurso meramente informativo. Simplemente muestra un resumen de los enlaces que se pueden usar.

  Así que, al igual que en el caso anterior, no nos detendremos más en ella.

- Se han añadido además tres URLs adicionales:

  Para darle algo más de funcionalidad a la aplicación, se ha diseñado un sistema que detecte si el dispositivo desde el que se consulta es un dispositivo móvil o, en su defecto, se considera un puesto de trabajo.

  Con esta información adicional, podremos ofrecer dos interfaces de usuario diferentes según el dispositivo en el que estemos.

  Teniendo una interfaz más cargada de elementos para la parte de visión desde un ordenador, y una más simple para poder ver desde el móvil, ya que se sobrentiende que al estar visualizando la información desde un terminal móvil, no se dispone del mismo tiempo para filtrar la parte de información que nos es útil de la meramente decorativa.

  A pesar de esto, se ha añadido la posibilidad de forzar que detecte el dispositivo desde el que estamos de la manera que nosotros deseemos. Bien para que nos sirva a modo de consulta, o depuración obteniendo la interfaz que deseemos, o bien porque a pesar de ser detectado como un tipo, queramos formar parte del otro.

  - **Fijar nuestra visión como si fuéramos un PC**

    La URL escogida para este fin es "/setdevice/pc".

    Si usamos este método, en consultas sucesivas se nos considerará un puesto de trabajo, independientemente de lo que seamos en realidad.

- **Fijar nuestra visión como si fuéramos un móvil**

  La URL escogida para este fin es "/setdevice/phone".

  En este caso nos encontramos exactamente en el método antagónico del anterior. Con él forzaremos a que en futuras consultas seamos considerados un dispositivo móvil.

- **Eliminar los valores de fijado de visión** y permitir que la aplicación detecte la fuente desde la que hacemos las consultas.

  La URL escogida para este fin es "/unsetdevice".

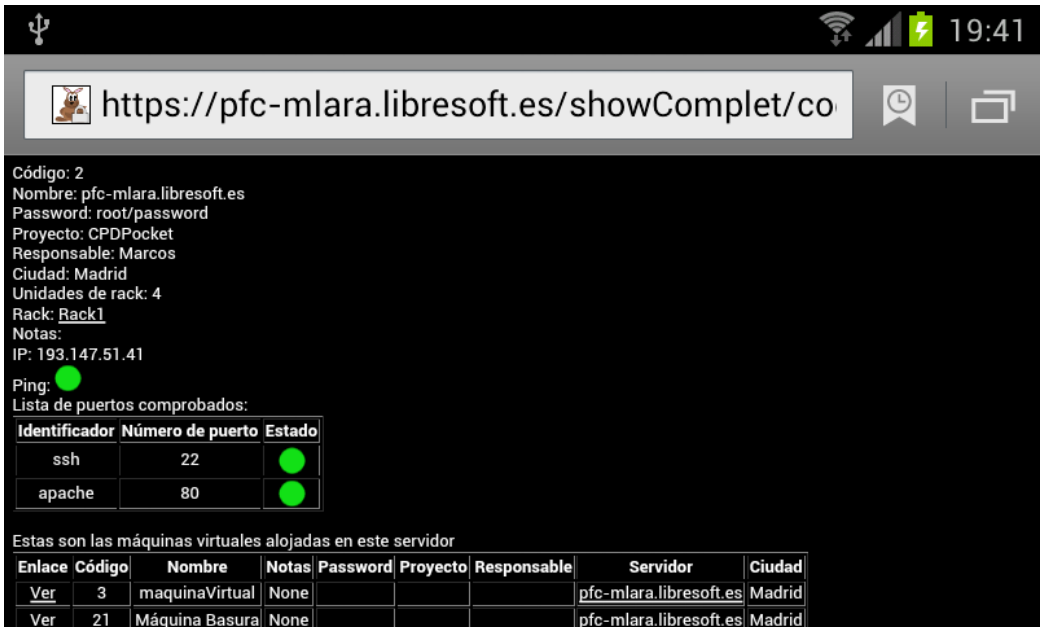
  Finalmente para eliminar los efectos que hayamos podido forzar con los enlaces anteriores, utilizando este último, volveremos a permitir a la aplicación que nos "detecte" y que nos ofrezca el material como crea conveniente.

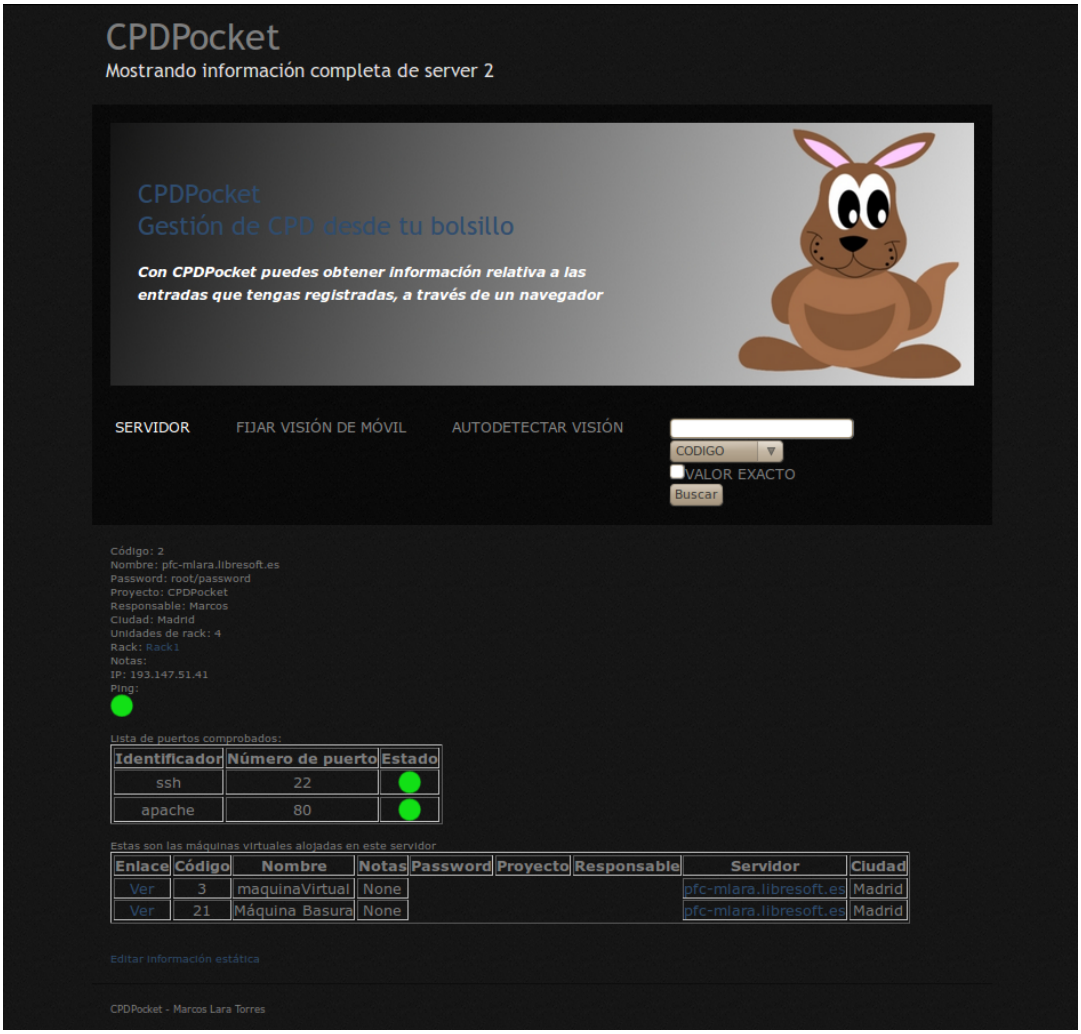
Esta funcionalidad está basada en el uso de plantillas que nos ofrece Django. Para utilizarlo de una manera simplificada y poder reutilizar todo lo posible, se ha diseñado una estructura modular de las diferentes partes necesarias en cada una de las páginas, y además se utiliza herencia para simplificar aún más.

Cada elemento, tanto PC como dispositivo móvil, posee una plantilla de base de la que hereda el formato visual común, y sobre él añade los diferentes módulos que completan la página.

De esta manera podemos apreciar en el siguiente ejemplo, una página relativa a una consulta avanzada sobre una máquina.

El ejemplo corresponde a la misma página consultada desde un PC y desde un dispositivo móvil. La página en sí muestra una consulta completa mostrando la información dinámica asociada a una máquina. Cómo podemos apreciar en las imágenes, en la parte de PC se muestra una página más atractiva, que incluye una cabecera con imagen y un pie.


19:41
https://pfc-mlara.libresoft.es/showComplet/co
Código: 2
Nombre: pfc-mlara.libresoft.es
Password: root/password
Proyecto: CPDPocket
Responsable: Marcos
Ciudad: Madrid
Unidades de rack: 4
Rack: Rack1
Notas:
IP: 193.147.51.41
Ping:
Lista de puertos comprobados:
Identificador | Número de puerto | Estado
ssh | 22
apache | 80
Estas son las máquinas virtuales alojadas en este servidor
Enlace | Código | Nombre | Notas | Password | Proyecto | Responsable | Servidor | Ciudad
Ver | 3 | maquinaVirtual | None | pfc-mlara.libresoft.es | Madrid
Ver | 21 | Máquina Basura | None | pfc-mlara.libresoft.es | Madrid


CPDPocket
Mostrando información completa de server 2
CPDPocket
Gestión de CPD desde tu bolsillo
Con CPDPocket puedes obtener información relativa a las entradas que tengas registradas, a través de un navegador
SERVIDOR
FIJAR VISIÓN DE MÓVIL
AUTODETECTAR VISIÓN
CODIGO
VALOR EXACTO
Buscar
Código: 2
Nombre: pfc-mlara.libresoft.es
Password: root/password
Proyecto: CPDPocket
Responsable: Marcos
Ciudad: Madrid
Unidades de rack: 4
Rack: Rack1
Notas:
IP: 193.147.51.41
Ping:
Lista de puertos comprobados:
Identificador | Número de puerto | Estado
ssh | 22
apache | 80
Estas son las máquinas virtuales alojadas en este servidor
Enlace | Código | Nombre | Notas | Password | Proyecto | Responsable | Servidor | Ciudad
Ver | 3 | maquinaVirtual | None | pfc-mlara.libresoft.es | Madrid
Ver | 21 | Máquina Basura | None | pfc-mlara.libresoft.es | Madrid
Editar información estática
CPDPocket - Marcos Lara Torres

4.1.3 Servidor Web

Dependiendo del entorno en el que nos encontremos deberemos tratar de proteger en mayor o menor medida la aplicación. Y como viene siendo habitual, y ya se habrá comprobado a lo largo de esta documentación, trataremos de utilizar las bondades de las funcionalidades ya implementadas en el software adicional que estamos utilizando.

Por la naturaleza de esta aplicación, tenemos información tan sensible como son nombres de usuario y contraseñas de usuarios privilegiados en las máquinas que estamos administrando.

Dos de estas medidas de seguridad que podremos configurar en la parte del servidor WEB son:

- Utilización de comunicaciones mediante SSL, que permitirá encriptar la comunicación entre el cliente y el servidor, impidiendo así que un tercero unicamente escuchando en la red adquiera información privilegiada. Esto lo conseguimos utilizando HTTPS en lugar de HTTP.
- Protección del uso, tanto la consulta, como la edición de la aplicación mediante un usuario.

Como ya se ha repetido en alguna ocasión, debemos elegir una solución de servidor web para no utilizar el pequeño servidor de desarrollo incluido en Django.

Además recomiendan ofrecer con un servidor la aplicación y con otro diferente la parte estática de ésta.

En este caso, hemos optado por servir ambos contenidos con la misma solución brevemente descrita en el apartado de “Estado del Arte”, Apache.

Para configurar el servidor Django a través de Apache, disponemos de varios módulos capaces de gestionarlo. En este caso, utilizamos el modulo WSGI, el más recomendado para estos fines.

En el código del proyecto se facilita un fichero necesario para que la configuración sea correcta, de esta manera facilitamos el trabajo de cualquier instalación de CPDPocket.

Para servir el material estático creamos un directorio en el que estén contenidos todos los recursos, y nos ocupamos de que la URL descrita en el proyecto, apunte a este directorio. De esto, de nuevo se encargará Apache. La URL escogida es “static” y para que todo funcione, debemos añadir al fichero de configuración de nuestro servidor una “Alias” previo a la definición del módulo WSGI.

Se recomienda utilizar el sistema de servidores virtuales de apache, para poder tener un fichero de configuración expresamente para CPDPocket. De esta manera, podremos tener un control más preciso sobre todo el sistema, podremos definir logs independientes, etc.

4.2 Cliente

En la parte cliente, partíamos de la base de que queríamos reutilizar lo máximo posible de otros desarrollos anteriores, para fomentar la portabilidad y la usabilidad de la plataforma.

Realmente el uso completo de CPDPocket, se hace a través de un navegador web así que en esta parte no se ha hecho desarrollo alguno, y al tratarse de un software tan de uso general, no ha sido necesario realizar ninguna investigación explícita para el uso, siendo posible utilizar cualquier navegador del mercado.

Por otra parte, se quería tratar de aumentar en cierta medida, la facilidad con la que un usuario interactuara con una máquina física, teniendo a su disposición un dispositivo móvil.

4.2.1 Lector de códigos de barras

Teniendo en mente la posibilidad que tenemos de etiquetar las máquinas con etiquetas que muestren un código de barras, y de esta manera identifiquen de manera inequívoca una máquina. Se ha probado gran cantidad de software desarrollado por otros, para valorar la posible incorporación de esta funcionalidad.

Por motivos de accesibilidad, sólo se han probado aplicaciones Android, dejando para futuros estudios el realizar la misma investigación para dispositivos móviles con otros sistemas operativos, ya que no sólo los terminales Android están dotados del hardware necesario para este fin.

4.2.1 Lector de códigos de barras

Cabe destacar la aplicación “Barcode Scanner” desarrollado por ZXing Team. Es una aplicación bastante difundida, y con sus más de 300.000 opiniones, obtiene actualmente una calificación media superior al 4 sobre 5. Nótese que, para valorar la difusión de este software, no se han enunciado sus descargas sino el número de votos que la aplicación ha obtenido.

Este software trae la posibilidad de configurar una URL que nosotros queramos, y mediante una sintaxis especial, generar peticiones personalizadas al escanear un código de barras.

Una vez dentro de la aplicación, pulsando la tecla menú configuramos el apartado “Búsqueda de productos con URL personalizada”, le asignamos el valor “direcciónServidorCPDPocket/show/code/%s”. Una vez hecho esto, cada vez que escaneemos un código de barras aparecerá la opción de “Búsqueda personalizada” que generará una consulta mediante un método GET a la URL configurada, sustituyendo el “%s” por el código capturado en cuestión.

Se recomienda utilizar la URL que únicamente muestra la información estática, ya que el otro apartado depende de la latencia de la red, y puede dar la sensación de que la aplicación no está trabajando. Pero si se quiere, se podría usar la URL de “showComplet” para mostrar tanto la información estática como la dinámica.

4.2.2 Lector de tarjetas RFID

Una vez más, teniendo en mente la posibilidad que tenemos de etiquetar cada máquina de manera inequívoca, esta vez pensamos en las etiquetas RFID.

Las ventajas y desventajas que éstas ofrecen frente a los códigos de barras, serán descritos más en profundidad, en el apartado de Conclusiones.

Al igual que en el apartado anterior, se han probado numerosos lectores de etiquetas RFID para tratar de reutilizar su desarrollo y aumentar la funcionalidad de este Proyecto de Fin de Carrera. Al igual que en el caso anterior, sólo se ha probado dentro del contexto de terminales con Sistema Operativo Android.

Esta vez sin éxito, ya que no se ha encontrado un software que permita leer tarjetas RFID y que además permita añadirle las modificaciones oportunas para que con dicha información se genere una consulta hacia nuestro servidor.

Por ello, se decidió realizar una pequeña aplicación que permita cubrir esta necesidad.

Ya que no aportaría ninguna funcionalidad adicional, para la elaboración de esta parte se decidió seguir un modelo de prototipos. De esta forma, de la manera más rápida tendríamos una aplicación lo más sencilla posible para probar su funcionamiento, y en caso de aportar alguna mejora significativa, podríamos dedicarle más tiempo.

Por tanto, se pretendía elaborar una aplicación que simplemente detectara una tarjeta RFID y que posteriormente abriera un navegador apuntando a la URL correspondiente.

Para este fin, se siguieron los pasos descritos en el apartado de “Estado del Arte” para llegar al primer prototipo.

En el momento del desarrollo de esta idea no se encontró demasiada documentación relativa a la utilización de NFC desde Android. Por esto se empezó a probar casi por ensayo y error para encontrar los métodos apropiados para el reconocimiento del identificador de la tarjeta.

Después de muchos intentos sin demasiado éxito, se empezó a tratar de analizar aplicaciones ya creadas que a pesar de tener más funcionalidad de la que en principio se necesitaba para este fin.

Finalmente, tras tratar de analizar la aplicación “NFC Reader”, desarrollada por Adam Nybäck, se contactó con él por correo electrónico, y tras explicarle la situación, él mismo indicó el método de la librería que debería servir para este fin tan simple.

Se desarrolló una aplicación que, una vez lanzada, simplemente esperaba la detección de una tarjeta RFID, y lanzaba un navegador. Estaba equipada con una tosca interfaz que no permitía personalizar absolutamente nada, la URL se definía en tiempo de compilación, y una vez detectada una etiqueta era necesario volver a lanzarla para empezar de nuevo.

4.2.2 Lector de tarjetas RFID

Este funcionamiento, a pesar de ser bastante rudimentario, fue necesario para explorar esta funcionalidad y desecharla. De esta manera, se abandonó el desarrollo de esta parte, debido a las conclusiones sacadas a partir de su uso, que se explicarán de una manera más detallada en el apartado de conclusiones.

5 Uso del sistema en un entorno real

Este proyecto ha sido probado en un entorno real con cientos de máquinas en un CPD distribuido en varias sedes. Las pruebas fueron realizadas de manera intensiva sólo en una de las sedes, y resultó ser de gran utilidad. Por motivos de seguridad y privacidad, no se pueden dar detalles precisos del mismo.

La instalación, utilización inicial, y formación de los miembros del equipo fueron dirigidos por el autor de este proyecto.

Para poder hacer una segunda prueba, teniendo el desarrollo un estado mucho más maduro, se ha elegido el equipo de administradores de sistemas del grupo GsyC/LibreSoft, de la Universidad Rey Juan Carlos. Este departamento se dedica a la investigación de software libre, entre otras labores.

El entorno de esta segunda prueba es bastante más modesto en cuanto a número de máquinas, ya que no se trata de un entorno orientado a una empresa grande.

Este nuevo experimento consta de distintos entornos, algunos en colaboración y otros autónomos. Posee a pequeña escala todos los elementos que han sido contemplados en el desarrollo, incluidos los entornos de virtualización con máquinas virtuales.

Además como esta prueba está orientada a su explicación en este documento, sólo se han añadido máquinas cuya descripción se pueda compartir, omitiendo en cualquiera de los casos los nombres de usuario y contraseña por seguridad.

Esta vez se ha hecho una instalación del servicio por parte del autor del proyecto, se les ha dado las urls necesarias para su completa utilización y libertad de uso en ella. De esta manera comprobamos además su usabilidad, y su compresión.

Tras un mes de pruebas del sistema, se les realizó una breve encuesta con cinco preguntas orientadas a conocer la opinión personal sobre el desarrollo, su utilidad, y sus posibles mejoras futuras.

5 Uso del sistema en un entorno real

Los usuarios coinciden en el potencial de este proyecto tanto en su idea inicial, como con su funcionalidad implícita que permite tener un inventario de máquinas. Pero destacan que el incremento de trabajo que supone el registro de las máquinas hace que la utilización sea más útil cuantas más máquinas se utilicen. Y en su caso el número total de máquinas quizá sea bastante reducido para un sistema como este.

Para tratar de solventar esto, y mejorar la tarea del registro de máquinas, recomiendan modificar la interfaz, para que no esté tan orientada a la consulta de códigos, y permita de una manera más natural, la interacción en el añadido de nuevos registros.

Por otra parte, también consideran que la aplicación tiene un propósito claro y bien definido, y que cumple con él. Creen que les sería de gran utilidad en sus labores diarias administrando un CPD más grande.

6 Conclusiones y futuras lineas de trabajo

6.1 Conclusiones

Para la realización de este proyecto he usado gran parte de los conocimientos aprendidos durante la carrera. Sin duda, por encima de todo este conocimiento aplicado, también se encuentran las metodologías y aptitudes asimiladas, que te dotan de un punto de vista crítico y de una visión ingenieril. Considero que la gran mayoría de las asignaturas cursadas durante mi formación me han servido, además de para adquirir conocimientos, para adquirir algo mucho más valioso y complejo. Evidentemente estamos hablando de adquirir una mentalidad, una forma de pensar, una forma de trabajar.

La elaboración de este proyecto ha pasado por muchas etapas, y me alegra saber que ha pasado por las fases que he estudiado a lo largo de la carrera, referentes al desarrollo del software.

Gracias a que cuenta con varios apartados, ha sido posible utilizar dos metodologías distintas.

Ésto nos ha ayudado bastante, sobre todo en la elección de una metodología ágil de desarrollo basada en el prototipado rápido para la parte frustrada del cliente desarrollado para Android que nos serviría para leer códigos RFID.

El haber utilizado un desarrollo rápido mediante prototipos, me sirvió para desechar la idea de utilizar tarjetas RFID en la aplicación.

Para poder leer un código de barras se necesita saber exactamente su ubicación, tener una iluminación adecuada, y por último, disponer del sitio suficiente que permita situar el lector de forma perpendicular al código. Si alguno de estos tres puntos no estaba presente, dificultaba notablemente la operación, llegando a casos en los que es imposible realizarla.

6.1 Conclusiones

Por ello se pensó en lecturas por proximidad, ofrecidas por la radiofrecuencia. A pesar del incremento del gasto que ello suponía. No se debe pasar por alto que para los códigos de barras, nos es suficiente con una impresora doméstica, mientras que para las tarjetas RFID, debemos comprar expresamente las tarjetas y grabarlas con los datos oportunos. En el contexto en el que estamos, se pensó utilizar simplemente el identificador de la tarjeta como clave única para abaratar en la medida de lo posible la opción.

La idea era suplir la necesidad de las tres situaciones descritas para el código de barras. Inicialmente se probó con tarjetas pasivas, algo más pequeñas y con menos mantenimiento. Al hacer las pruebas, comprobamos que era necesario situar el lector del móvil a unos dos centímetros de la tarjeta. Esto de nuevo volvía a traer el problema de "saber donde exactamente se encontraba la tarjeta".

Para solucionar este problema se pensó en tarjetas activas, pero al tener un radio de acción bastante mayor, y dado que las máquinas enrackadas están muy próximas entre sí, dificultaría mucho saber que máquina se estaba escaneando en cada momento, a pesar de estar frente a ellas.

Por estas razones, se descartó la radiofrecuencia en este proyecto, y se pasó a utilizar unicamente la identificación mediante código de barras.

Otra parte de desarrollo, que se ha realizado, pero no se ha incluido en esta versión de CPDPocket, es una adaptación de los datos de un monitor Nagios a los valores dinámicos de las máquinas. En este caso se trataba de recoger los datos que obtenía el servidor de Nagios del CPD, y formatearlos para ser mostrados en la vista extendida de una máquina registrada.

El problema que surgió aquí es que el desarrollo en este caso, era demasiado a medida sobre el entorno en el que se creó, evitando así la flexibilidad que un software que se pretenda distribuir necesita.

La idea inicial trataba de recoger los datos de las estructuras de datos internas de Nagios, y se traduciría en un problema a la hora de extrapolarlo a otros entornos, o incluso al cambiar de versión, o tipo de software de monitorización.

Por estas razones, y al contrario que en el caso anterior, esta vez no se ha desechado, sino que se ha aplazado como una posible mejora a aplicar en el futuro.

Al margen de estos detalles concretos, y revisando los objetivos fijados al comienzo de este proyecto, creo que se han cubierto de manera satisfactoria la totalidad de ellos.

En resumen, me alegro de haber realizado este Proyecto de Fin de Carrera, y me alegro también de haberlo hecho con la idea de liberarlo como software libre. Ésta será mi primera contribución directa a una ideología que tanto me ha dado y que por desgracia nunca había podido colaborar con una idea completa.

Espero que este proyecto no caiga en el olvido, y llegue más allá de un ejercicio para una calificación académica. Al menos por mi parte, pretendo continuar con el desarrollo completando funcionalidades y sobretodo, continuar aprendiendo durante ese desarrollo.

En cuanto al tema personal, este proyecto me ha permitido usar tecnologías hasta entonces desconocidas para mi, algunas mencionadas en este documento. Una de ellas que merece especial mención es “Git” como control de versiones. Sin duda, también me ha permitido mejorar en otras como por ejemplo Python y Django.

6.2 Futuras lineas de trabajo

La principal motivación de este proyecto ha sido el de cubrir una necesidad que se ha detectado en un entorno concreto, para facilitar la labor de un administrador de de sistemas de un CPD. Por ello, las futuras lineas de trabajo son muy amplias, pudiendo ampliar este sistema con tantas funcionalidades como se nos ocurran.

Como ya se ha mencionado en las conclusiones, se pretende avanzar en el desarrollo de CPDPocket, una vez entregado en este estado como Proyecto de Fin de Carrera. Por lo tanto, a continuación se comentarán algunas de las ideas que han ido surgiendo a lo largo del desarrollo y que no han sido añadidas por cuestión de tiempo.

- Añadir gestión de usuarios. En el estado actual, sólo se contempla un único usuario, contando con que existe un proceso de registro inicial en el que se almacenen los datos relativos al grueso del CPD. El resto de consultas deberían ser de sólo lectura, y se ha dado por hecho que al ser una herramienta de administradores para administradores, no se iban a realizar tareas maliciosas. En el futuro, se podría utilizar el modulo de usuarios de Django para generar distintos tipos de usuario, para de esta manera poder tener perfiles que sólo lean, otros que puedan escribir, y sobre todo, cierta información sensible sólo mostrada para unos pocos. Todo esto contando con que CPDPocket pueda ser usado para más fines.

- Desarrollar un cliente completo para dispositivos móviles. En lugar de utilizar el lector de códigos “Barcode Scanner”, se podría hacer una aplicación completa, inicialmente para Android, y después migrarla según necesidad. Esta aplicación podría consistir en un lector de códigos de barras, que al estar diseñada expresamente para CPDPocket, permita utilizar directamente ciertas opciones tras decodificar un código. Evitando así tener que seguir enlaces por el navegador web.

  Para esto se podría dejar el servidor tal cual está ahora mismo, o también se podría añadir soporte para servir el contenido mediante una estructura de datos que interprete y formatee la aplicación.

- Dar soporte a equipos de red. Existen varios equipos de red que podríamos contemplar en este desarrollo. En el estado actual podríamos registrarlos como un servidor físico cualquiera, pero se podría añadir la funcionalidad necesaria para que de manera dinámica, se muestre más información de estos dispositivos. Por ejemplo mostrando una configuración de rutas configurada en la máquina.
- Aumentar funcionalidad directa de CPDPocket sobre las máquinas registradas en su base de datos. Se podrían dotar el sistema de los suficientes privilegios como para poder apagar las máquinas de manera ordenada ante algún tipo de petición especial. Se podría añadir la información oportuna para realizar un encendido remoto de la máquina, etc.
- Integración de informes de monitorización dentro de la información dinámica de la máquina. De esta manera, al consultarla, podríamos ver también los detalles de los informes generados por el software de monitorización que se haya escogido para controlar el CPD.

6.2 Futuras lineas de trabajo

7 Bibliografía

- [DjangoWebSite] Información técnica. Sitio oficial de *Django, https://www.djangoproject.com/*. Última fecha de consulta: mayo 2013.
- [PythonWebSite] Información técnica. Sitio oficial de *Python*, *http://www.python.org/* . Última fecha de consulta: abril 2013.
- [AndroidDeveloper] Información técnica. Sitio oficial de *Android* para desarrolladores, *http://developer.android.com/*. Última fecha de consulta: mayo 2013.
- [Godinez:2009] L. M. Godinez Gonzalez, *RFID: oportunidades y riesgos, su aplicación práctica*, 2009, ISBN ISBN 9789701513118
- [RFIDLibro] Telectronia, *Introducción a la identificación por Radio Frecuencia*, *http://www.telectronica.com/rfidtelectronica.pdf*. Última fecha de consulta: enero 2013.
- [RFC2616] R.Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, y T. Berners-Lee, *RFC2616 – Hypertext Transfer Protocol* – HTTP/1.1 Información técnica, Network Working Group, IETF (Internet Engineering Task Force), Junio 1999.
- [Holovaty:2009] A. Holovaty y J. Kaplan-Moss, *The Django Book. Web Framework for Python*, *http://djangobook.com*
- [Lutz:2006] M. Lutz, *Programming Python*, 3rd Ed. O'Reilly, 2006 ISBN 9780596009250
- [Gramlich:2008] N. Gramlich, *Android Programming*, *http://andbook.anddev.org*, 2008

7 Bibliografía

8 Glosario de términos

Android Sistema Operativo basado en Linux que se ha extendido rápidamente en dispositivos móviles. Cada vez se usa en una variedad más grande de ellos. En la actualidad podemos encontrar dispositivos tan diversos como smart phones, tablets, y ordenadores portátiles.

API Acrónimo de "Application Programming Interface" o en español Interfaz de Programación para Aplicación. Es un conjunto de llamadas acordadas, que se ofrecen al programador para facilitar el uso de un módulo, librería, etc sin tener que adaptar el código o dedicar excesivo tiempo a su comprensión.

Arquitectura Cliente-Servidor Modelo de diseño de estructura basado en dos roles fuertemente diferenciados por su funcionalidad en el sistema. Por un lado el servidor genera recursos y está atento a la recepción de peticiones. Por otro el cliente se encarga de consumir esos recursos mediante solicitudes.

Django Framework de desarrollo web escrito en Python y de código abierto. Basa su uso en el patrón de Modelo-Vista-Controlador.

Framework en el contexto que nos concierne. Conjunto de librerías, compiladores, y demás software necesario para facilitar el desarrollo en un entorno concreto.

HTML Acrónimo de HyperText Markup Language o en español lenguaje de marcado de hipertexto. Se trata de un lenguaje de marcado muy extendido, que describe la estructura y el contenido de un documento utilizando una notación de etiquetas.

HTTP Acrónimo de HiperText Transfer Protocol o en español protocolo de transferencia de hipertexto. Es un protocolo definido por la World Wide Web Consortium y la Internet Engineering Task Force que define la sintaxis y la semántica que utilizan los elementos software en una arquitectura web.

HTTPS HTTP Secure o HTTP Seguro. Extensión de HTTP que soporta cifrado en la comunicación, aumentando la utilidad de éste y permitiendo tener comunicaciones seguras mediante este protocolo.

8 Glosario de términos

Python Lenguaje de programación interpretado. Se usa cada vez más en administración como lenguaje de scripting mucho más potente que bash.

Sistema Operativo Conjunto de software que junto con los controladores específicos de dispositivos sirve de capa intermedia entre el hardware del dispositivo y las aplicaciones que se utilizan en él. Gestiona todos los recursos del sistema.

9 Anexo

9.1 Manual de usuario

Con este manual de usuario se pretende facilitar el uso de este sistema.

Se ha diseñado específicamente para que sea muy intuitivo, así que no requiere de ningún tipo de formación especial, más allá del uso de un ordenador con soltura de nivel usuario del sistema, además de un uso "normal" de la navegación web.

La dirección del sistema que se recomienda utilizar, sobre todo para empezar a aprovecharlo es la de consulta simple. Solo con esa url tendremos acceso a todas las demás opciones a través de la navegación.

Pero para dotar al usuario de una mayor flexibilidad, se recomienda conocer mínimamente las diferentes urls de uso, para poder utilizarlas sin necesidad de usar la navegación.

Contando con que <SERVIDOR> se sustituya por el servidor de CPDPocket en cada caso:

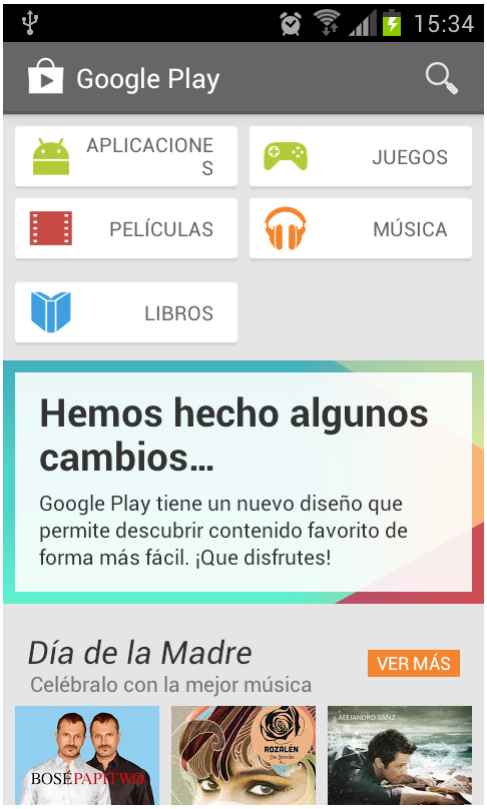
- Consulta simple, que muestra la información estática del <CÓDIGO> "<SERVIDOR>/show/code/<CÓDIGO>".
- Consulta avanzada, que muestra la información estática y dinámica del <CÓDIGO> "<SERVIDOR>/showComplet/code/<CÓDIGO>".
- Editar un código: "<SERVIDOR>/edit/code/<CÓDIGO>".
- Añadir un código: "<SERVIDOR>/add/code/<CÓDIGO>".
- Hacer una búsqueda: "<SERVIDOR>/search"

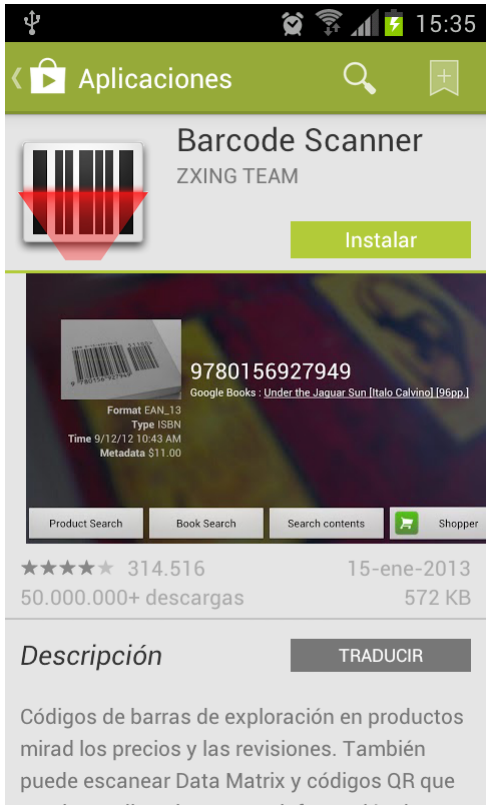
9.1.1 Configuración del entorno en un dispositivo con Android

Para poder usar todas sus funcionalidades no es necesario utilizar un dispositivo móvil, sin embargo puede facilitar mucho la tarea de registrar un código de barras.

Para conseguir esto, desde un dispositivo móvil con sistema operativo Android, instalamos la aplicación Barcode Scanner.

9.1.1 Configuración del entorno en un dispositivo con Android

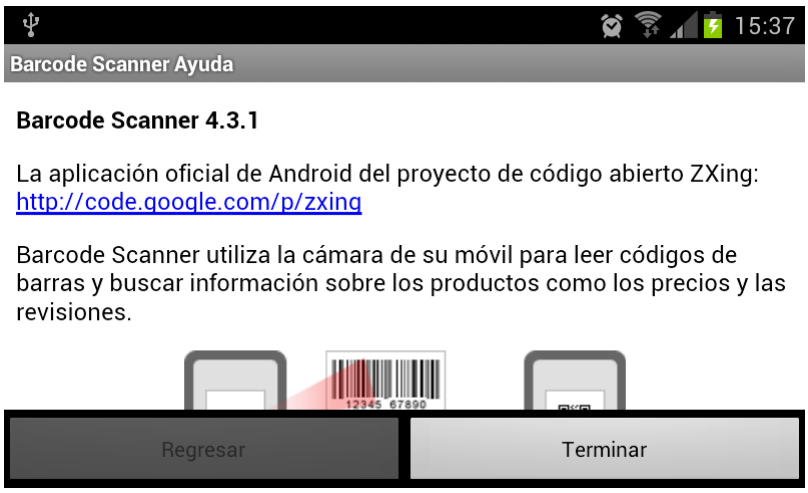




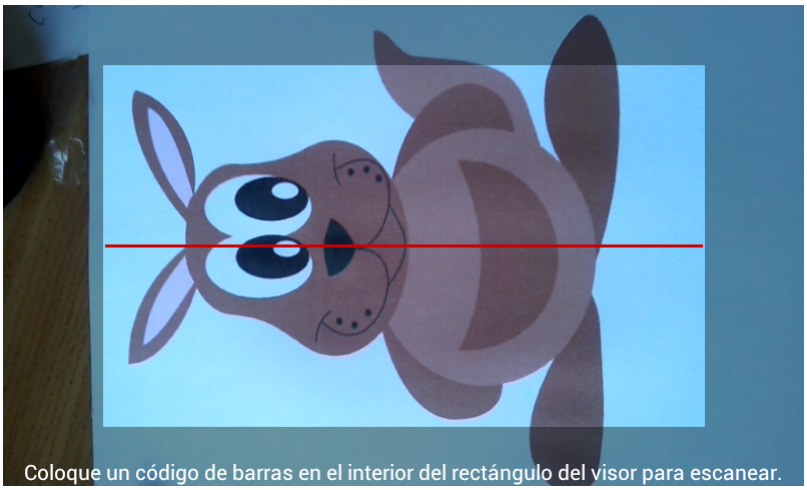
Una vez localizada, la instalamos siguiendo el procedimiento habitual, pulsando en “Instalar” y aceptando la declaración de permisos definidos en el manifiesto de la aplicación.

Bien, ya tenemos la aplicación en nuestro dispositivo, ahora debemos añadirle la información necesaria para poder hacer consultas al servidor de CPDPocket.

Ejecutamos la aplicación, la primera vez que lo hagamos, nos mostrará un mensaje de texto explicando el funcionamiento de esta aplicación.

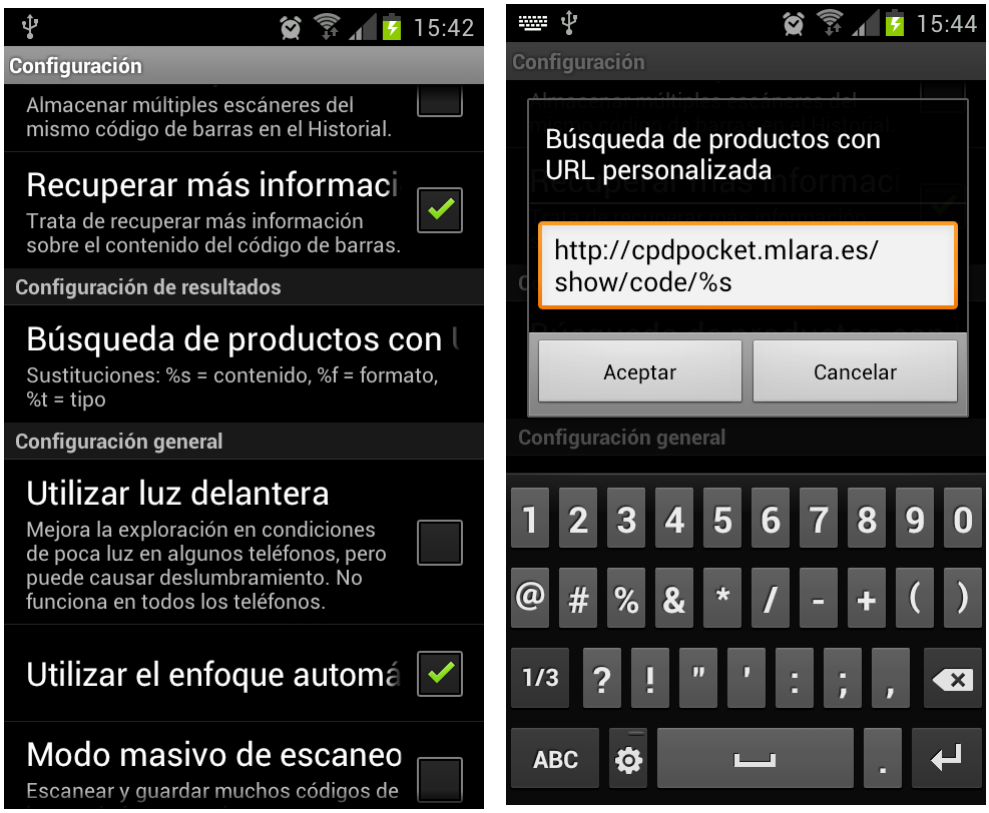


Una vez aceptado, se nos mostrará una pantalla especial. Se activará la cámara, y empezaremos a ver lo que ve nuestra cámara en la pantalla del móvil, y superpuesto por parte de la aplicación, una linea roja que nos servirá de guía a la hora de escanear los códigos.



Con esta pantalla funcionando, pulsamos el botón de menú de nuestro dispositivo. Aparecerán las opciones de la aplicación. Pulsamos configuración, y buscamos la opción “Búsqueda de productos con URL personalizada”.

9.1.1 Configuración del entorno en un dispositivo con Android

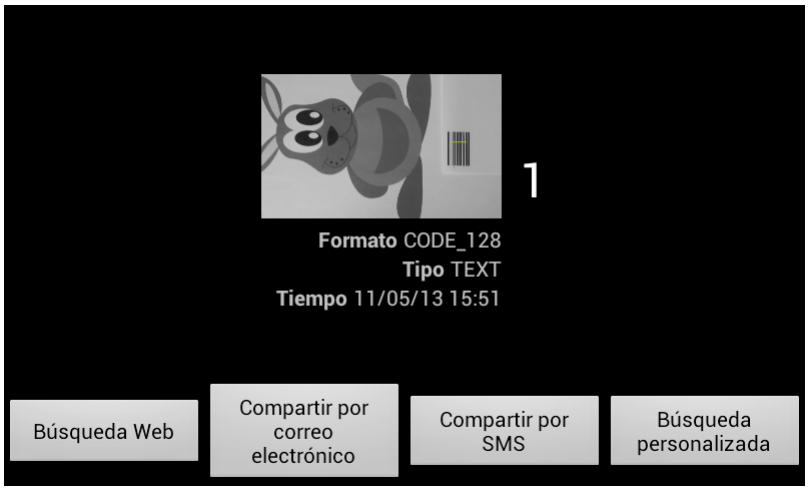


Una vez allí debemos introducir, siguiendo la sintaxis especial descrita en la pantalla, el patrón de url que se utilizará en las búsquedas. Se recomienda utilizar la url relativa a la búsqueda simple, en lugar de la completa, para tratar de evitar el retardo generado por la consulta dinámica que le aplica la red. En nuestro caso, el servidor de CPDPocket utilizado es "http://cpdpocket.mlara.es/" y al aplicarle la sintaxis de Barcode Scanner y utilizar la consulta simple, obtenemos la url que debemos poner en este apartado: "http://cpdpocket.mlara.es/show/code/%s", es decir "<SERVIDOR>/show/code/%s".

Esta configuración previa solo es necesario realizarla una vez, ya que a partir de este momento, la "búsqueda personalizada" estará vinculada a la url definida.

Una vez que tenemos la aplicación móvil lista. En la parte de detección de códigos, la que hemos descrito anteriormente con la linea roja superpuesta sobre la imagen capturada por nuestra cámara en tiempo real, procederemos a leer nuestro código.

Situamos la linea en perpendicular a las lineas del código de barras a detectar. Sin necesidad de pulsar ningún botón, veremos como la cámara modifica su enfoque hasta detectarlo, emitiendo a la vez un pitido.



En este momento tenemos el código “capturado”, se nos muestra el código en valor numérico detectado, y una captura de la pantalla en el momento de la detección.

Con estas opciones en nuestra pantalla, pulsamos el botón de “Búsqueda personalizada”, que construirá la url que hemos definido anteriormente, sustituyendo el “%s” por el código de barras, el “1” en este caso.

9.1.1 Configuración del entorno en un dispositivo con Android

Por último se abrirá el navegador que tengamos por defecto y mostrará el resultado de consultar dicho código en CPDPocket.

A partir de este momento, continuaremos utilizando el sistema siguiendo las funcionalidades descritas por el sistema, siguiendo la interfaz web.

9.2 Manual de instalación

Para realizar una instalación satisfactoria de un sistema completo de CPDPocket, debemos completar 3 partes principalmente.

En este manual se describirá el procedimiento para instalar una versión simple dentro de una arquitectura basada en máquinas Linux, más concretamente, máquinas Debian/Ubuntu y Redhat/Centos.

9.2.1 Base de datos

Para la utilización de CPDPocket podemos utilizar varias soluciones de bases de datos, de hecho podemos utilizar cualquier Sistemas de Gestión de Bases de Datos soportado por Django.

En este caso hemos optado por utilizar MySQL como SGBD. Dentro de las opciones que podemos elegir después de haber escogido MySQL.

Si tenemos instalado MySQL en el sistema, saltad al siguiente paso. A priori no tenemos ningún requisito de versión mínima adicional sobre este software.

9.2.1.1 Instalación de MySQL

Para instalar MySQL utilizaremos los paquetes de los repositorios. Para ello ejecutaremos:

- "sudo aptitude install mysql-server" o "sudo yum install mysql-server" dependiendo de si estamos en una máquina basada en Debian o RedHat.

Una vez hecho esto nos pedirá una contraseña para el usuario de root de MySQL.

9.2.1.2 Crear base de datos y usuario para CPDPocket

Una vez que tenemos el SGBD instalado, debemos crear la base de datos y el usuario para que utilice CPDPocket. Para ello nos conectamos al servidor de MySQL ejecutando

- "mysql -u root -p".

9.2.1 Base de datos

Una vez dentro del interprete de MySQL, creamos la base de datos:

- “create database <NombreDeLaBaseDeDatos>;”

Por último creamos un usuario con permisos sobre la base de datos:

- “GRANT ALL ON <NombreDeLaBaseDeDatos> TO <Usuario>@'localhost' IDENTIFIED BY <ContraseñaParaUsuario>;

En este caso ponemos “localhost” porque estamos creando un usuario que solo podrá ser utilizado desde la propia máquina en la que está el servidor de MySQL, condición que se cumple en este ejemplo, pero no tiene por qué darse en el entorno que el lector esté creando.

Una vez terminados estos pasos, salimos del interprete de MySQL con un:

- “exit;”

y damos por concluida la configuración en la parte de la base de datos.

9.2.2 CPDPocket

Para instalar CPDPocket, descomprimiremos el paquete que lo contiene, pero antes, debemos satisfacer sus dependencias, para que todo funcione correctamente.

9.2.2.1 Dependencias directas

Hay una serie de programas de los que depende directamente CPDPocket.

- El primero de ellos es Python. Necesitamos la versión 2.7.* o superior. Este requisito nos viene impuesto por la versión de Django que utilizamos.
    - Dependiendo de la distribución y la versión que estemos utilizando tendremos este software en los repositorios. Pero como siempre, podemos acceder directamente a las fuentes oficiales. http://www.python.org/download/

- En el desarrollo de CPDPocket se ha utilizado la versión 1.5 de Django. En este caso, se recomienda utilizar la misma versión para tratar de evitar incompatibilidades. A lo largo de este desarrollo se han visto diferentes versiones de Django y su estructura ha ido variando, lo que podría suponer una incompatibilidad con versiones posteriores que obliguen al usuario a modificar la estructura de este proyecto.

  En este caso, recomendamos utilizar las fuentes directamente para poder obtener esta versión concreta. https://www.djangoproject.com/download/1.5/tarball/

- Para las funcionalidades dinámicas de CPDPocket necesitamos:

  - Necesitamos el programa "ping", por tanto necesitamos tener instalado el paquete "iputils", como no necesitamos ninguna versión en particular, lo instalaremos desde los repositorios:

    - "aptitude install iputils" o "yum install iputils"

  - Necesitamos el programa "nmap". Al igual que en el caso del ping, no necesitamos una versión concreta, y puesto que nmap es un paquete en si mismo, lo instalaremos desde los repositorios:

    - "aptitude install nmap" o "yum install nmap"

  - Por último, las librerías correspondientes para el correcto funcionamiento, que complementan la instalación de Python son:

    - "python-dnspython" para la resolución DNS del nombre de la máquina.

    - "python-mysqldb" para permitir que se pueda utilizar MySQL desde Python.

9.2.2.2 CPDPocket

Para instalar el proyecto de Django correspondiente a CPDPocket, debemos seguir una serie de directrices muy sencillas que se detallan a continuación:

9.2.2 CPDPocket

- Una vez conseguido el paquete con la versión correspondiente de CPDPocket, debemos escoger un emplazamiento en el que descomprimirlo. Una vez elegido será necesario referenciarlo, así que es importante escoger un sitio sin demasiadas restricciones de acceso, o al menos un lugar en el que tenerlas controladas.

  Recomendamos como opción /opt/cpdpocket, pero podríamos utilizar cualquier otro.

  Para descomprimirlo:

  - tar -xzf CPDPocket-<VERSIÓN>.tgz

- Una vez terminado esto, debemos modificar el fichero de configuración de Django, que deberá apuntar a la correcta. Para eso vamos al fichero "cpdpocketProject/cpdpocketProject/settings.py" y dentro de la configuración de base de datos, debemos asignar el nombre de la creada en el paso anterior, el usuario y el password del usuario.

  - ```
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': '<NombreDeLaBaseDeDatos>',
            'USER': '<Usuario>',
            'PASSWORD': '<ContraseñaParaUsuario>',
            'HOST': '',
            'PORT': '',
        }
    }
    ```

9.2.3 Servidor web

Para completar el sistema, haciendo una vez más alusión a que Django no se dedica a hacer servidores, procederemos a preparar la interfaz web, que nos hará de intermediario entre nuestro proyecto Django y el cliente web.

Al igual que en el caso de la base de datos, para esto hemos escogido utilizar Apache con su módulo wsgi para ejecución de Python. Recordar que existen más opciones igualmente válidas.
    ```

9.2.3.1 Instalación de Apache

Para instalar Apache utilizaremos los paquetes de los repositorios. Para ello ejecutaremos:

- "sudo aptitude install apache2" o "sudo yum install httpd" dependiendo de si estamos en una máquina basada en Debian o RedHat.

Además instalamos el módulo wsgi que ya hemos mencionado:

- "sudo aptitude install libapache2-mod-wsgi" o "sudo yum install mod_wsgi" dependiendo de si estamos en una máquina basada en Debian o RedHat.

Por último para poder utilizar el protocolo HTTPS y que Apache nos dote de algo más de seguridad, utilizaremos su módulo para ssl. En el caso de Debian, se instala por defecto, mientras que en RedHat debemos instalarlo.

- "a2enmod ssl" para activar el servicio en Debian o "yum install mod_ssl" para instalarlo en RedHat

9.2.3.2 Configurando Apache

Para completar la instalación terminamos de configurar el apache.

Recomendamos utilizar un VirtualHost para tener todo lo relativo a CPDPocket controlado.

Creamos un fichero de configuración en el que detallaremos:

- El puerto en el que escuchará este VirtualHost. En nuestro caso, el 443 ya que queremos que utilice HTTPS:

```
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
    SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
</VirtualHost>
```

- El directorio de ficheros estáticos. Hemos optado por utilizar un Document Root, y en él hemos creado un enlace simbólico, apuntando al directorio de estáticos que se encuentra en "cpdpocket/cpdpocketProject/cpdpocket/static/". Por otra parte, no es necesario realizar este enlazado, y podemos apuntar directamente a él.

```
Alias /static   /var/www/cpdpocket/static
```

```
<Directory /var/www/cpdpocket/static>
    Order deny,allow
    Allow from all
</Directory>
```

- La configuración del módulo WSGI. En CPDPocket se facilita un fichero de configuración para WSGI totalmente operativo, así que solamente debemos apuntar a él:

  WSGIScriptAlias / <PathHastaCPDPocket>/cpdpocketProject/cpdpocketProject/wsgi.py

- Por último, le añadimos un control de usuarios para evitar modificaciones de los registros.

```
<Directory <PathHastaCPDPocket>>
    AllowOverride AuthConfig
    AuthName "Para entrar necesitas identificación."
    AuthType Basic
    AuthUserFile <PathAlFicheroDeContraseñas>
    AuthGroupFile /dev/null
    Require valid-user
</Directory>
```

  Para completar este paso necesitamos haber creado un fichero de contraseñas, para ello utilizamos el comando de apache

  - htpasswd -c <PathAlFicheroDeContraseñas> <UsuarioWeb>

Esta vez la diferencia de versiones de sistema operativo nos hace ser algo más concienzudos en la descripción.

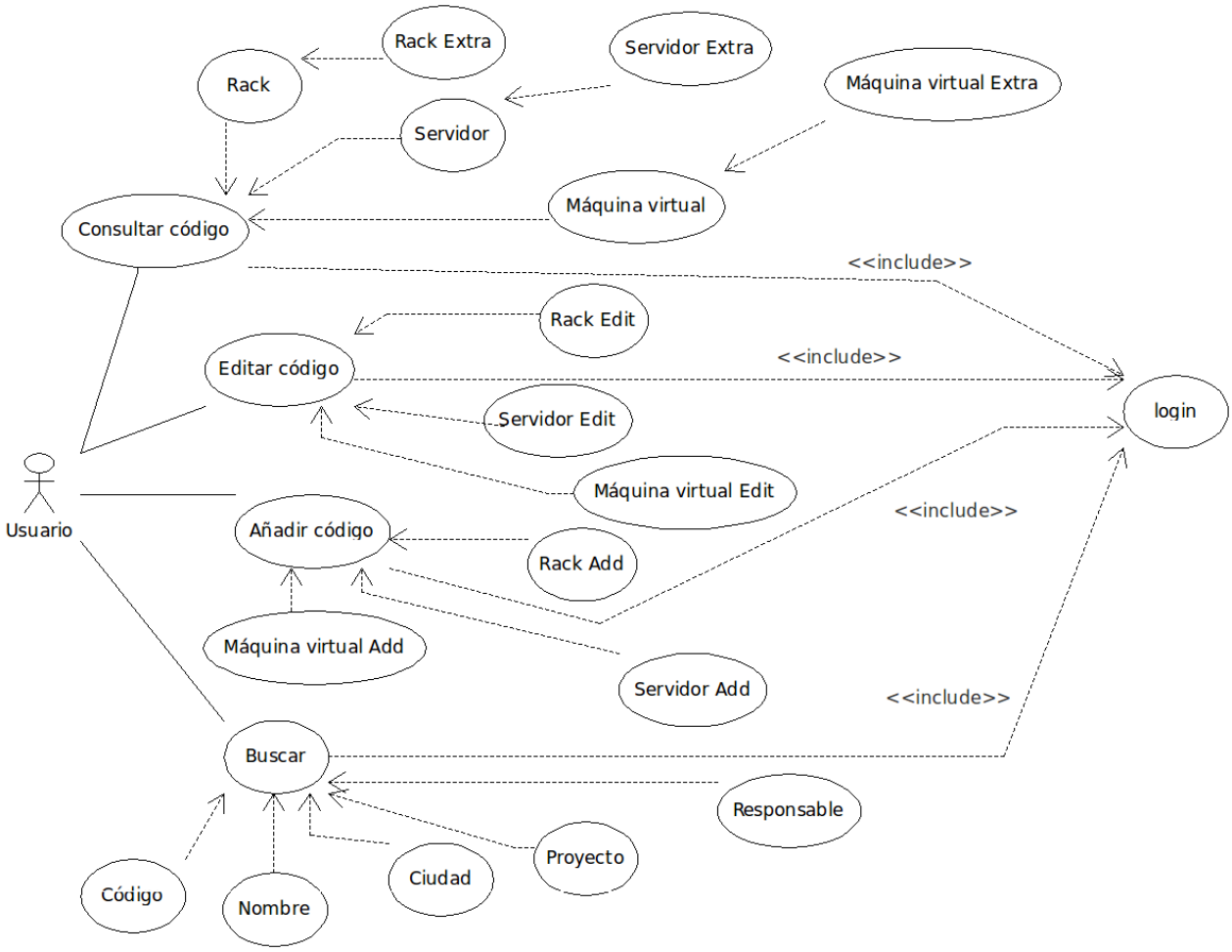
Para Debian, deberíamos crear el fichero en "/etc/apache2/sites-available" y posteriormente añadirlo a las páginas habilitadas "a2ensites nombrefichero"

Para RedHat, deberíamos crear el fichero en "/etc/httpd/conf.d/" y llamar al fichero de configuración "*.conf".

Una vez llegados a este punto, debemos reiniciar el servicio del Apache, y ya tendremos todo listo.

- "sudo /etc/init.d/apache2 restart" o "sudo /etc/init.d/httpd restart"

9.3 Diagramas de casos de uso



9.3 Diagramas de casos de uso

Consultar código rack

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a un rack.	
	2. El sistema recupera el registro y muestra la información relativa a ese código.
3. El usuario recibe la información solicitada.	

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario

Consultar código servidor

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a un servidor.	
	2. El sistema recupera el registro y muestra la información relativa a ese código.
3. El usuario recibe la información solicitada.	

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario

9.3 Diagramas de casos de uso

Consultar código máquina virtual

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a una máquina virtual.	
	2. El sistema recupera el registro y muestra la información relativa a ese código.
3. El usuario recibe la información solicitada.	

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario

Consultar código rack extra

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a un rack, utilizando la opción de extra.	
	2. El sistema recupera el registro, opera con los datos estáticos, genera datos dinámicos y muestra la información relativa a ese código.
3. El usuario recibe la información solicitada.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “Consultar código rack”.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir de una búsqueda para llegar a este punto.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.

9.3 Diagramas de casos de uso

Consultar código servidor extra

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a un servidor, utilizando la opción de extra.	
	2. El sistema recupera el registro, opera con los datos estáticos, genera datos dinámicos y muestra la información relativa a ese código.
3. El usuario recibe la información solicitada.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “Consultar código servidor”.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir de una búsqueda para llegar a este punto.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.

Consultar código máquina virtual extra

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a una máquina virtual, utilizando la opción de extra.	
	2. El sistema recupera el registro, opera con los datos estáticos, genera datos dinámicos y muestra la información relativa a ese código.
3. El usuario recibe la información solicitada.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “Consultar código máquina virtual”.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir de una búsqueda para llegar a este punto.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.

9.3 Diagramas de casos de uso

Editar código rack

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a un rack.	
	2. El sistema recupera el registro y muestra la información relativa a ese código en un formulario preparado para su edición
3. El usuario recibe el formulario con los datos registrados actualmente y las herramientas necesarias para su modificación.	
4. El usuario modifica los datos y envía el formulario.	
	5. El sistema recopila los datos facilitados por el usuario y realiza los cambios oportunos.
	6. El sistema envía el resultado final al usuario.
7. El usuario ve los cambios aplicados.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “mostrar un rack”.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir del caso de uso “mostrar un rack extra”.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.

Camino alternativo al paso 4: 4.1 El usuario cancela la operación.

Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.

Editar código servidor

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a un servidor.	
	2. El sistema recupera el registro y muestra la información relativa a ese código en un formulario preparado para su edición
3. El usuario recibe el formulario con los datos registrados actualmente y las herramientas necesarias para su modificación.	
4. El usuario modifica los datos y envía el formulario.	
	5. El sistema recopila los datos facilitados por el usuario y realiza los cambios oportunos.
	6. El sistema envía el resultado final al usuario.
7. El usuario ve los cambios aplicados.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “mostrar un servidor”.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir del caso de uso “mostrar un servidor extra”.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.

Camino alternativo al paso 4: 4.1 El usuario cancela la operación.

Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.

9.3 Diagramas de casos de uso

Editar código máquina virtual

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código vinculado a una máquina virtual.	
	2. El sistema recupera el registro y muestra la información relativa a ese código en un formulario preparado para su edición
3. El usuario recibe el formulario con los datos registrados actualmente y las herramientas necesarias para su modificación.	
4. El usuario modifica los datos y envía el formulario.	
	5. El sistema recopila los datos facilitados por el usuario y realiza los cambios oportunos.
	6. El sistema envía el resultado final al usuario.
7. El usuario ve los cambios aplicados.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “mostrar una máquina virtual”.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir del caso de uso “mostrar una máquina virtual extra”.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.

Camino alternativo al paso 4: 4.1 El usuario cancela la operación.

Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.

Añadir código rack

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código sin vincular para usarlo en un rack.	
	2. El sistema comprueba que el código no está siendo utilizado actualmente. Y envía el formulario para recoger los datos.
3. El usuario recibe el formulario y las herramientas necesarias para su modificación.	
4. El usuario inserta los datos y envía el formulario.	
	5. El sistema recopila los datos facilitados por el usuario y realiza los cambios oportunos.
	6. El sistema envía el resultado final al usuario.
7. El usuario ve los cambios aplicados.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “mostrar un rack”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir del caso de uso “mostrar un rack extra”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado ya está siendo utilizado e informa del error al usuario.

Camino alternativo al paso 4: 4.1 El usuario cancela la operación.

Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.

9.3 Diagramas de casos de uso

Añadir código servidor

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código sin vincular para usarlo en un servidor.	
	2. El sistema comprueba que el código no está siendo utilizado actualmente. Y envía el formulario para recoger los datos.
3. El usuario recibe el formulario y las herramientas necesarias para su modificación.	
4. El usuario inserta los datos y envía el formulario.	
	5. El sistema recopila los datos facilitados por el usuario y realiza los cambios oportunos.
	6. El sistema envía el resultado final al usuario.
7. El usuario ve los cambios aplicados.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “mostrar un servidor”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir del caso de uso “mostrar un servidor extra”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado ya está siendo utilizado e informa del error al usuario.

Camino alternativo al paso 4: 4.1 El usuario cancela la operación.

Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.

Añadir código máquina virtual

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario utiliza un código sin vincular para usarlo en una máquina virtual.	
	2. El sistema comprueba que el código no está siendo utilizado actualmente. Y envía el formulario para recoger los datos.
3. El usuario recibe el formulario y las herramientas necesarias para su modificación.	
4. El usuario inserta los datos y envía el formulario.	
	5. El sistema recopila los datos facilitados por el usuario y realiza los cambios oportunos.
	6. El sistema envía el resultado final al usuario.
7. El usuario ve los cambios aplicados.	

Camino alternativo al paso 1: 1.1 El usuario puede partir del caso de uso “mostrar una máquina virtual”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Camino alternativo 2 al paso 1: 1.2 El usuario puede partir del caso de uso “mostrar una máquina virtual extra”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado ya está siendo utilizado e informa del error al usuario.

Camino alternativo al paso 4: 4.1 El usuario cancela la operación.

Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.

9.3 Diagramas de casos de uso

Buscar

Este caso de uso parte de que el usuario se ha identificado correctamente en el sistema.

Actor	Sistema
1. El usuario solicita el recurso de búsquedas.	
	2. El sistema devuelve un formulario para realizar las búsquedas.
3. El usuario visualiza el formulario.	

Buscar código

Este caso de uso parte del caso de uso “buscar”.

Actor	Sistema
1. El usuario elige la opción de código e introduce un código completo con la opción de valor exacto.	
	2. El sistema realiza una búsqueda, y devuelve una lista estructurada de los resultados.
3. El usuario visualiza la lista.	

Camino alternativo al paso 1: 1.1 el usuario no elige el valor exacto e introduce un valor parcial.

Buscar nombre

Este caso de uso parte del caso de uso “buscar”.

Actor	Sistema
1. El usuario elige la opción de nombre e introduce un nombre completo con la opción de valor exacto.	
	2. El sistema realiza una búsqueda, y devuelve una lista estructurada de los resultados.
3. El usuario visualiza la lista.	

Camino alternativo al paso 1: 1.1 el usuario no elige el valor exacto e introduce una parte del nombre.

Buscar ciudad

Este caso de uso parte del caso de uso “buscar”.

Actor	Sistema
1. El usuario elige la opción de ciudad e introduce un nombre de ciudad completo con la opción de valor exacto.	
	2. El sistema realiza una búsqueda, y devuelve una lista estructurada de los resultados.
3. El usuario visualiza la lista.	

Camino alternativo al paso 1: 1.1 el usuario no elige el valor exacto e introduce una parte del nombre de la ciudad.

Buscar proyecto

Este caso de uso parte del caso de uso “buscar”.

Actor	Sistema
1. El usuario elige la opción de proyecto e introduce un nombre de proyecto completo con la opción de valor exacto.	
	2. El sistema realiza una búsqueda, y devuelve una lista estructurada de los resultados.
3. El usuario visualiza la lista.	

Camino alternativo al paso 1: 1.1 el usuario no elige el valor exacto e introduce una parte del nombre del proyecto.

9.3 Diagramas de casos de uso

Buscar responsable

Este caso de uso parte del caso de uso “buscar”.

Actor	Sistema
1. El usuario elige la opción de responsable e introduce un nombre de responsable completo con la opción de valor exacto.	
	2. El sistema realiza una búsqueda, y devuelve una lista estructurada de los resultados.
3. El usuario visualiza la lista.	

Camino alternativo al paso 1: 1.1 el usuario no elige el valor exacto e introduce una parte del nombre del responsable.

9.4 Modelo de análisis

Caso de uso: Consultar código rack

Diagrama de clases:

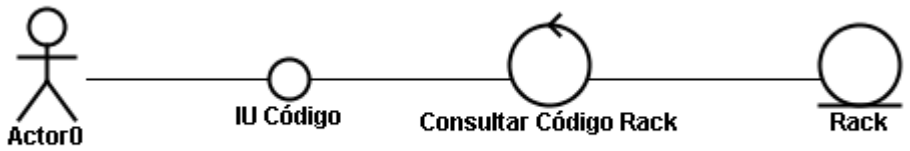
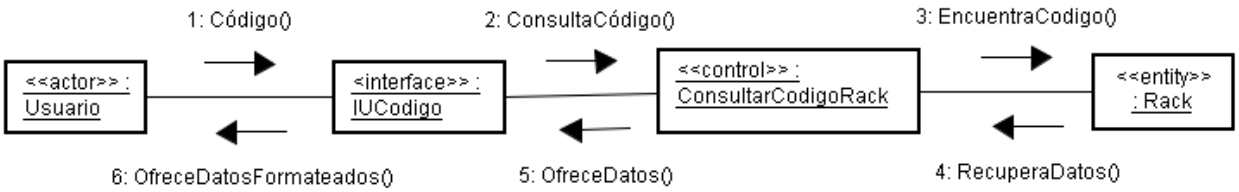
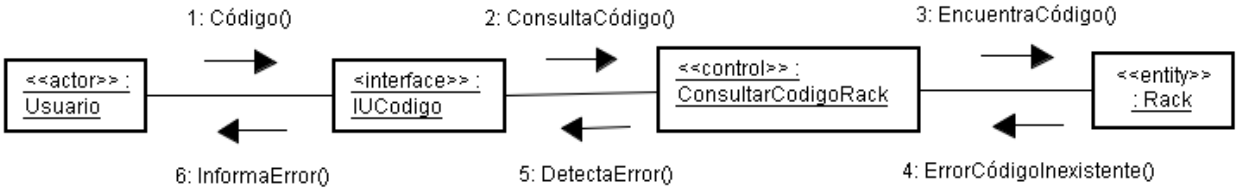


Diagrama de colaboración:



Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.



9.4 Modelo de análisis

Caso de uso: Consultar código servidor

Diagrama de clases:

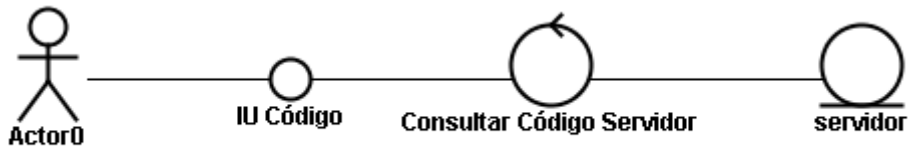
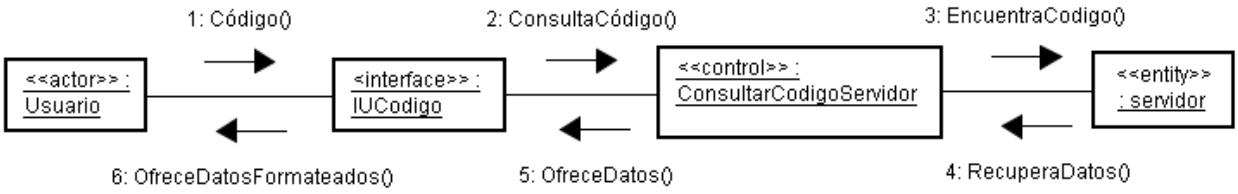
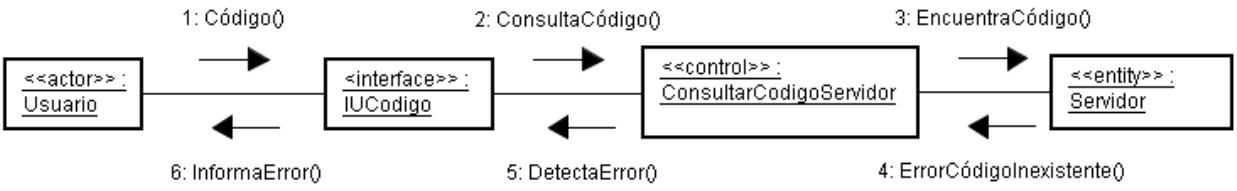


Diagrama de colaboración:



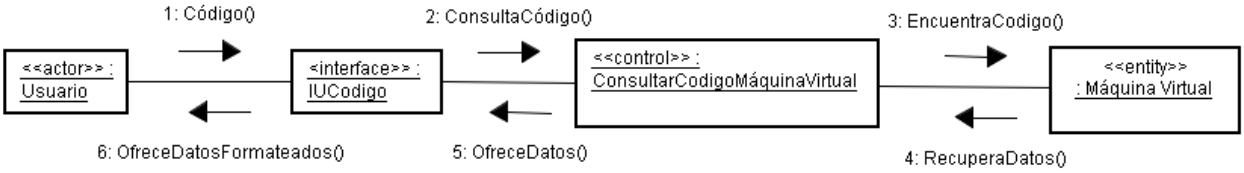
Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.



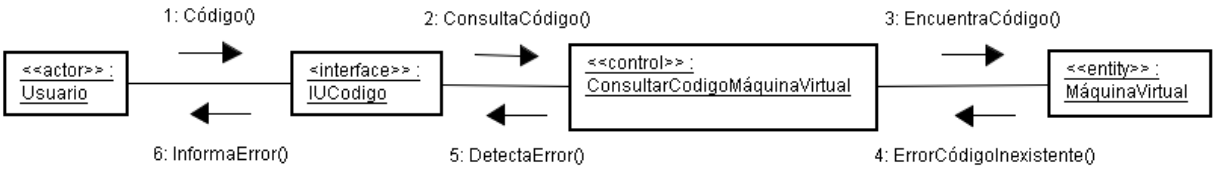
Caso de uso: Consultar código máquina virtual

Diagrama de clases:

Diagrama de colaboración:



Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario



9.4 Modelo de análisis

Caso de uso: Consultar código rack extra

Diagrama de clases:

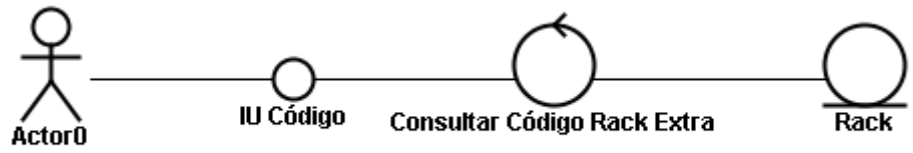
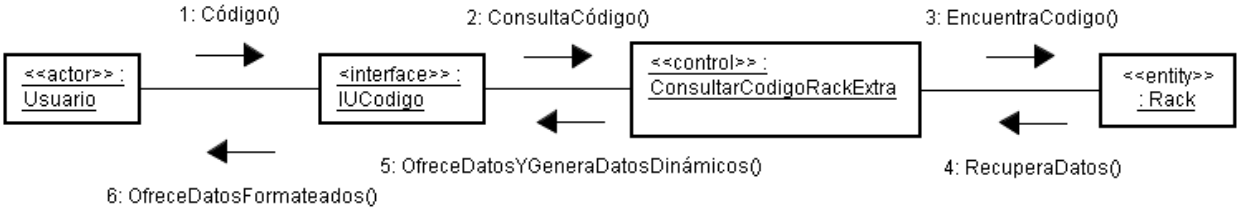


Diagrama de colaboración:



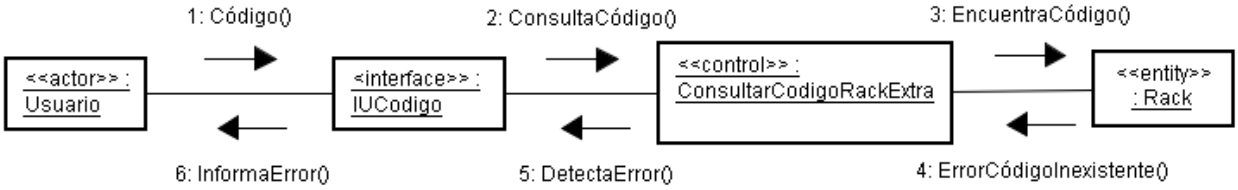
Camino alternativo 1.1: El usuario puede partir del caso de uso “Consultar código rack”.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

Camino alternativo 1.2: El usuario puede partir de una búsqueda para llegar a este punto.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

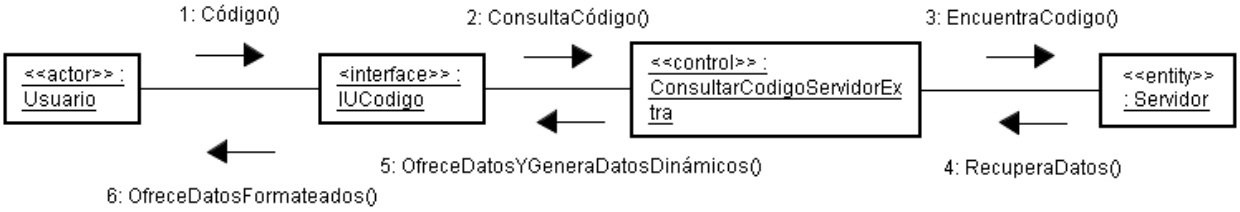
Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.



Caso de uso: Consultar código servidor extra

Diagrama de clases:

Diagrama de colaboración:



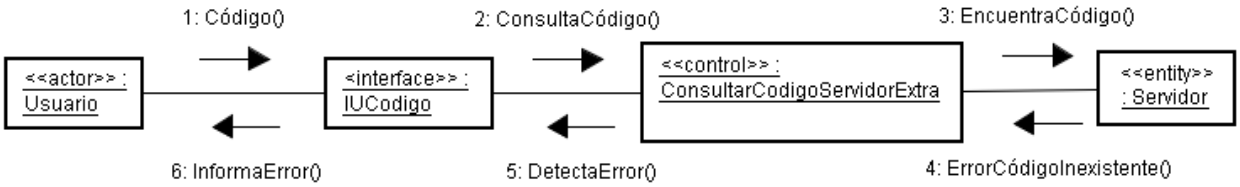
Camino alternativo 1.1: El usuario puede partir del caso de uso "Consultar código servidor".

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

Camino alternativo 1.2: El usuario puede partir de una búsqueda para llegar a este punto.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.

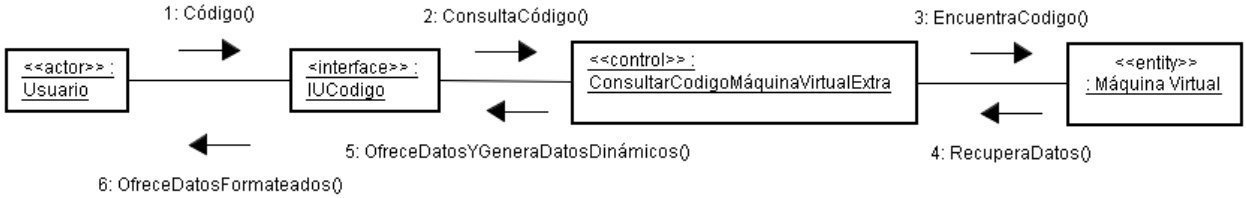


9.4 Modelo de análisis

Caso de uso: Consultar código máquina virtual extra

Diagrama de clases:

Diagrama de colaboración:



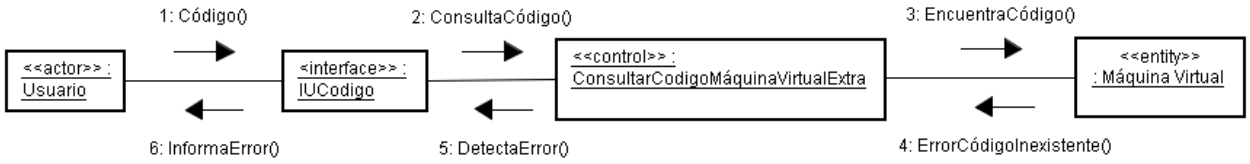
Camino alternativo 1.1: El usuario puede partir del caso de uso "Consultar código máquina virtual".

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

Camino alternativo 1.2: El usuario puede partir de una búsqueda para llegar a este punto.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

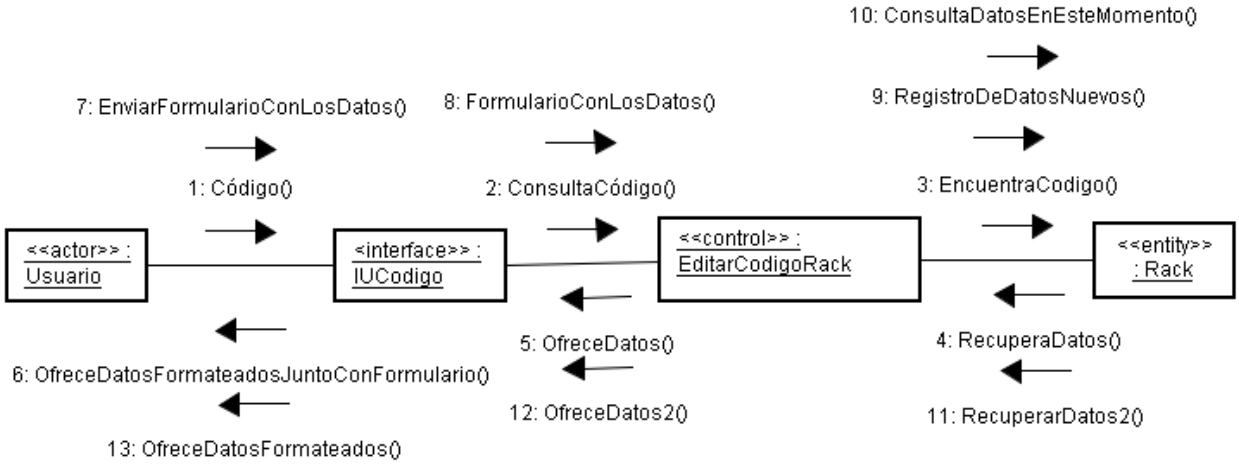
Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.



Caso de uso: Editar código rack

Diagrama de clases:

Diagrama de colaboración:



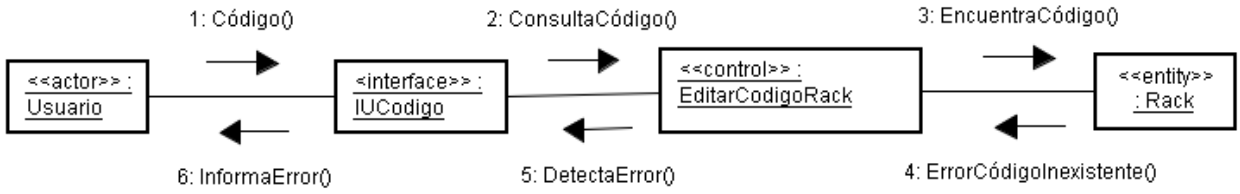
Camino alternativo 1.1: El usuario puede partir del caso de uso “mostrar un rack”.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

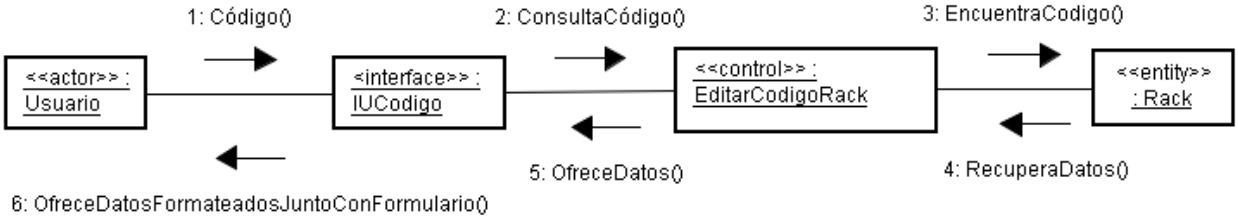
Camino alternativo 1.2: El usuario puede partir del caso de uso “mostrar un rack extra”.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

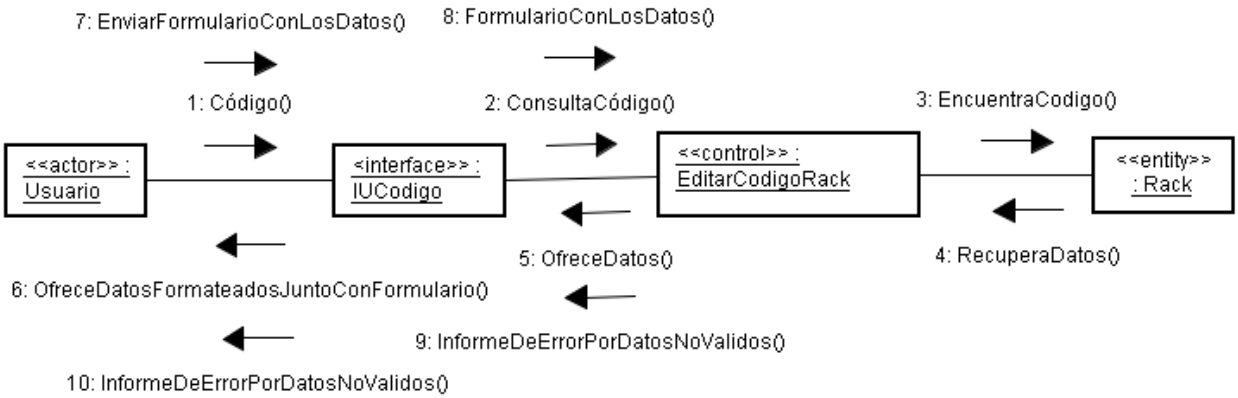
Camino alternativo al paso 2: 2.1 El sistema detecta que el código consultado no existe e informa del error al usuario.



Camino alternativo al paso 4: 4.1 El usuario cancela la operación.



Camino alternativo al paso 5: 5.1 El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.



Caso de uso: Editar código servidor

Diagrama de clases:

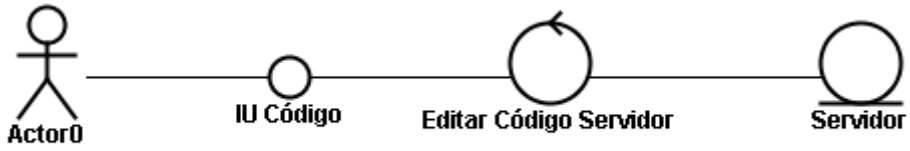
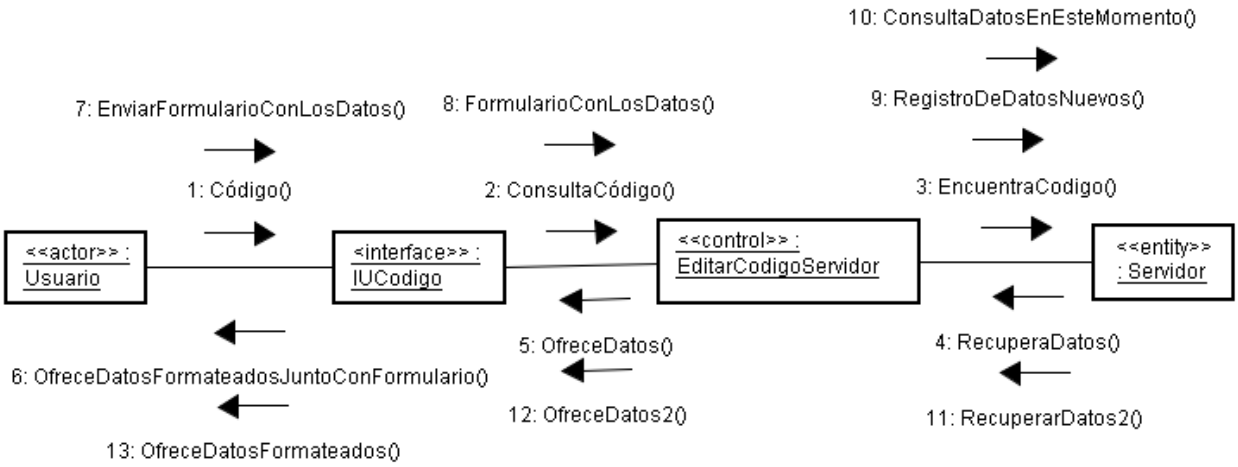


Diagrama de colaboración:



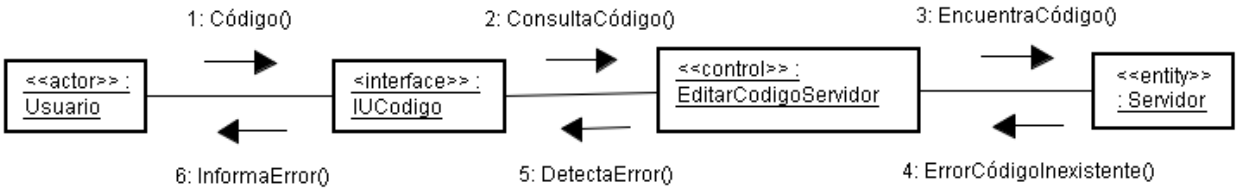
Camino alternativo 1.1: El usuario puede partir del caso de uso “mostrar un servidor”.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

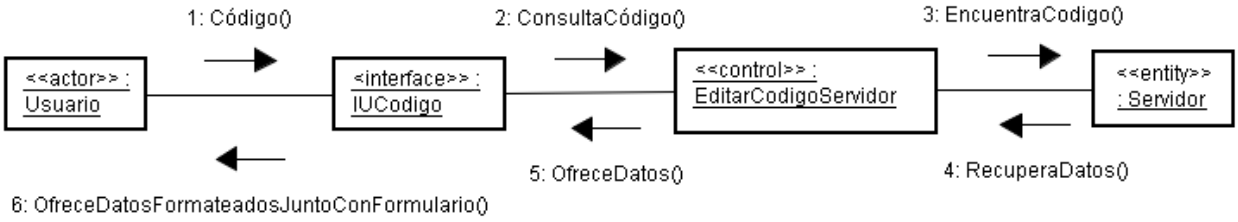
Camino alternativo 1.2: El usuario puede partir del caso de uso “mostrar un servidor extra”.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

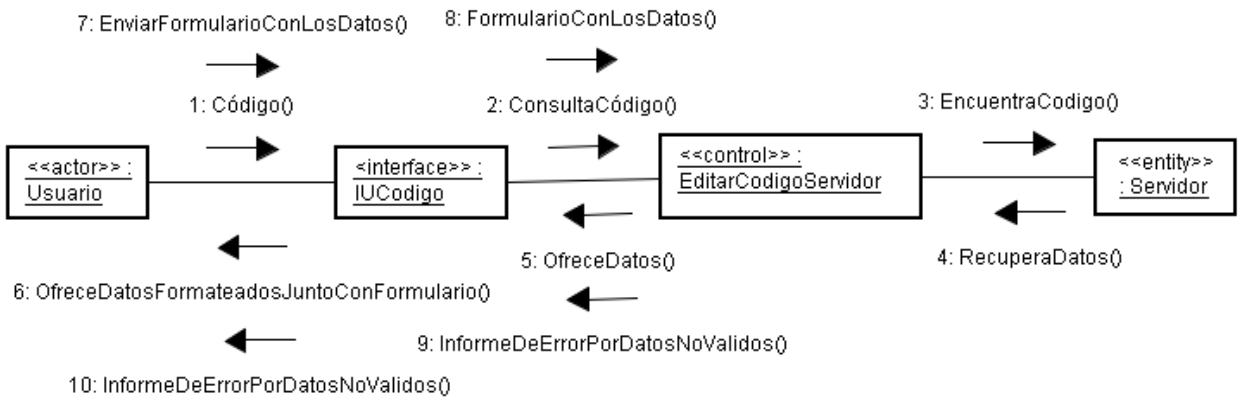
Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.



Camino alternativo 4.1: El usuario cancela la operación.



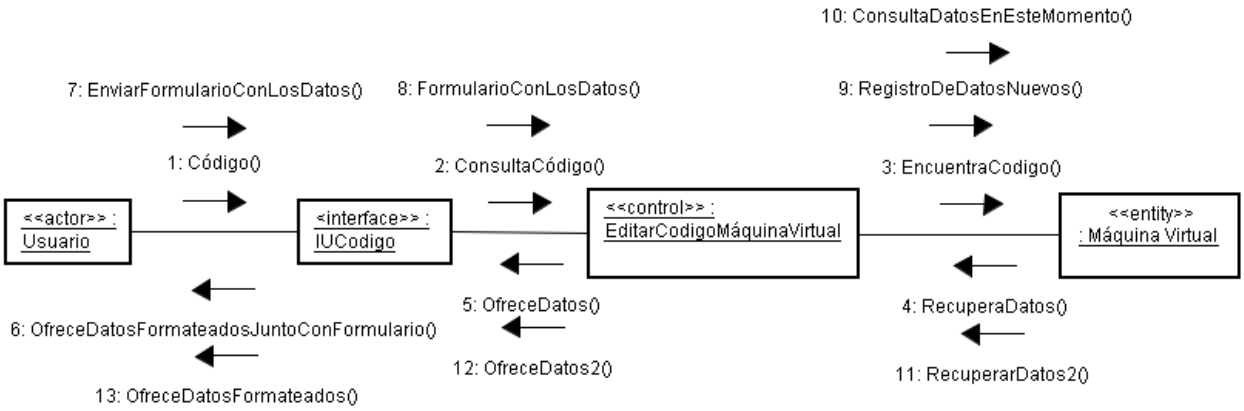
Camino alternativo 5.1: El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.



Caso de uso: Editar código máquina virtual

Diagrama de clases:

Diagrama de colaboración:



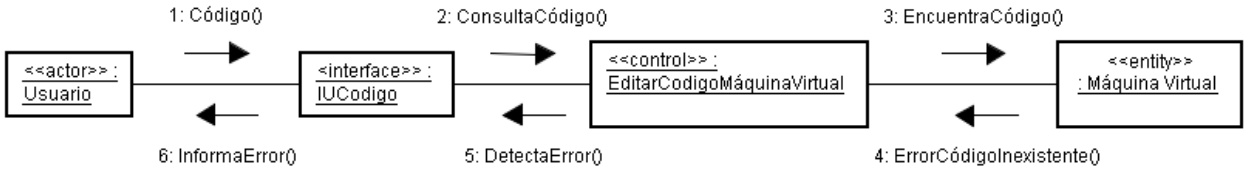
Camino alternativo 1.1: El usuario puede partir del caso de uso "mostrar una máquina virtual".

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

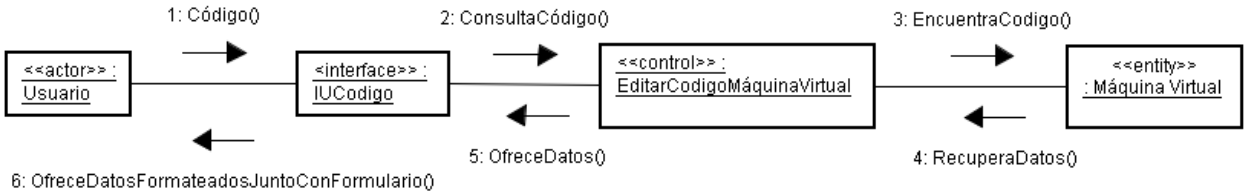
Camino alternativo 1.2: El usuario puede partir del caso de uso "mostrar una máquina virtual extra".

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

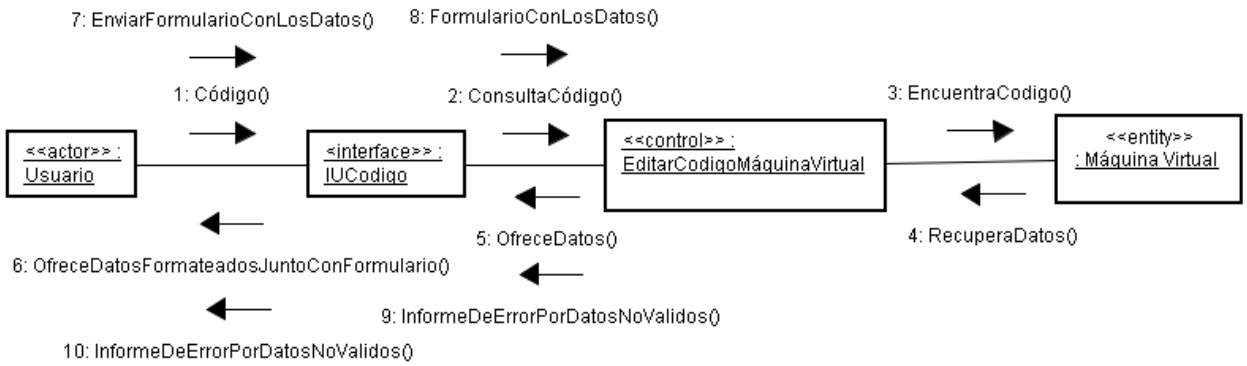
Camino alternativo 2.1: El sistema detecta que el código consultado no existe e informa del error al usuario.



Camino alternativo 4.1: El usuario cancela la operación.



Camino alternativo 5.1: El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.



Caso de uso: Añadir código rack

Diagrama de clases:

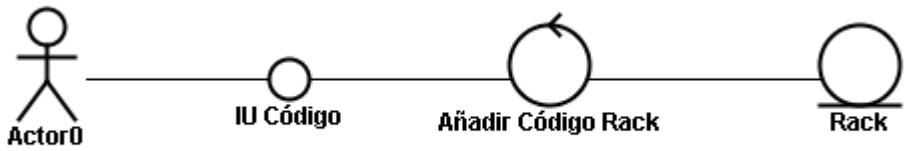
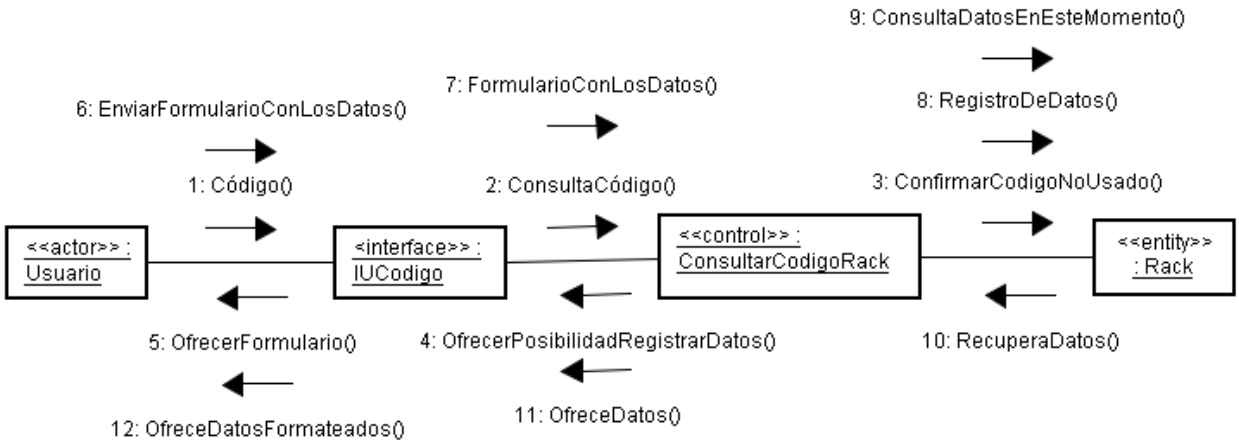


Diagrama de colaboración:



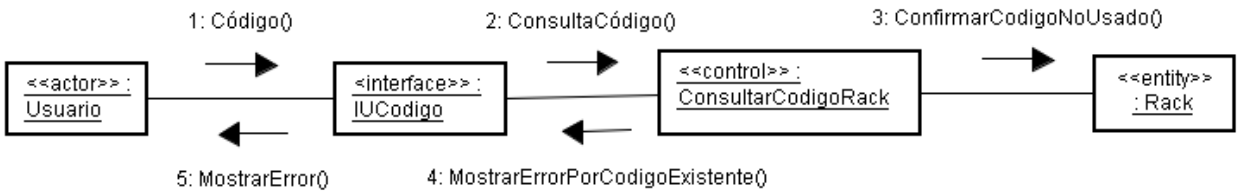
Camino alternativo 1.1: El usuario puede partir del caso de uso “mostrar un rack”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

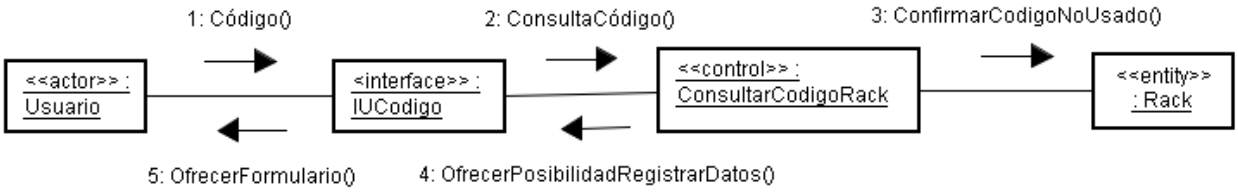
Camino alternativo 1.2: El usuario puede partir del caso de uso “mostrar un rack extra”, parte de la página de error ofrecerá un enlace para facilitar la adicción.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

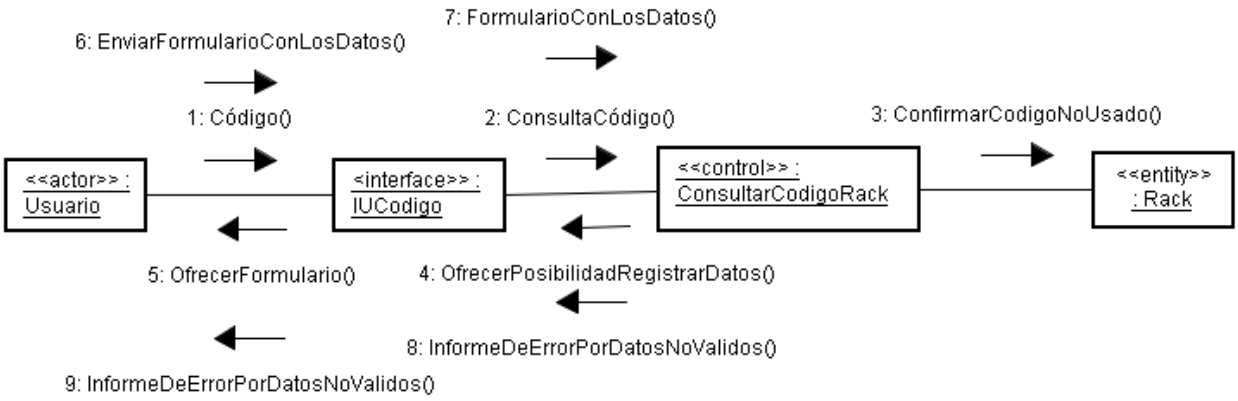
Camino alternativo 2.1: El sistema detecta que el código consultado ya está siendo utilizado e informa del error al usuario.



Camino alternativo 4.1: El usuario cancela la operación.



Camino alternativo 5.1: El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.



Caso de uso: Añadir código servidor

Diagrama de clases:

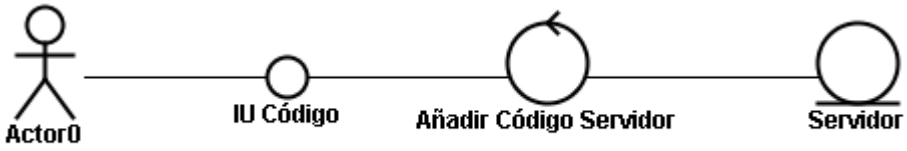
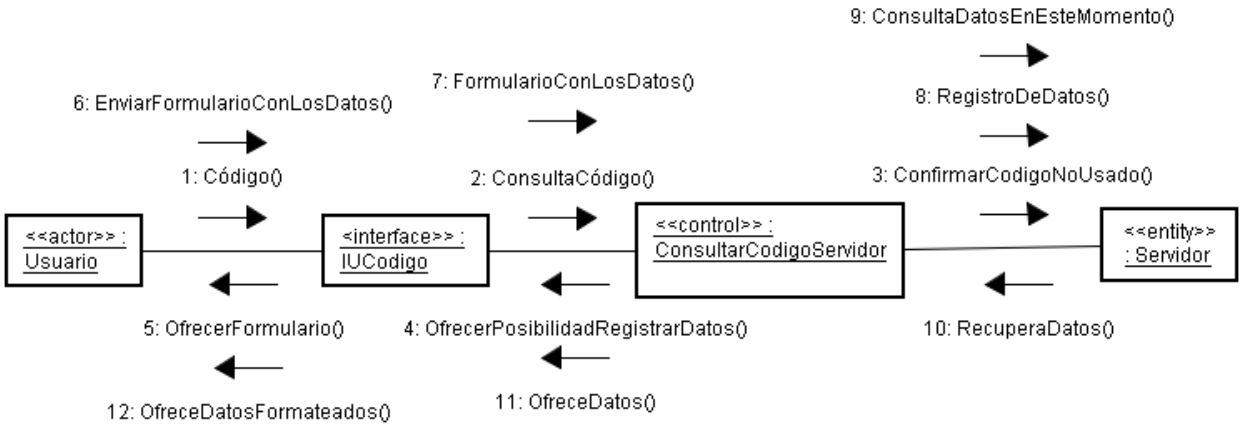


Diagrama de colaboración:



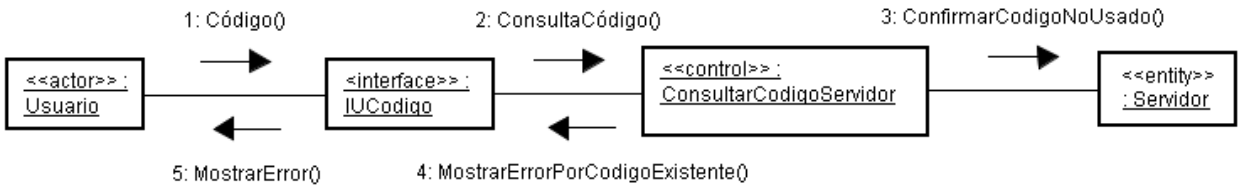
Camino alternativo 1.1: El usuario puede partir del caso de uso "mostrar un servidor", parte de la página de error ofrecerá un enlace para facilitar la adicción.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

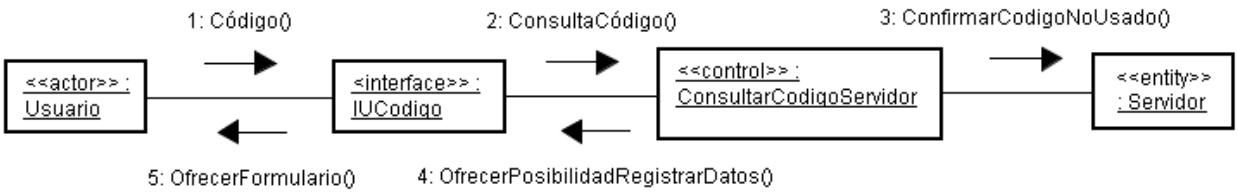
Camino alternativo 1.2: El usuario puede partir del caso de uso "mostrar un servidor extra", parte de la página de error ofrecerá un enlace para facilitar la adicción.

Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

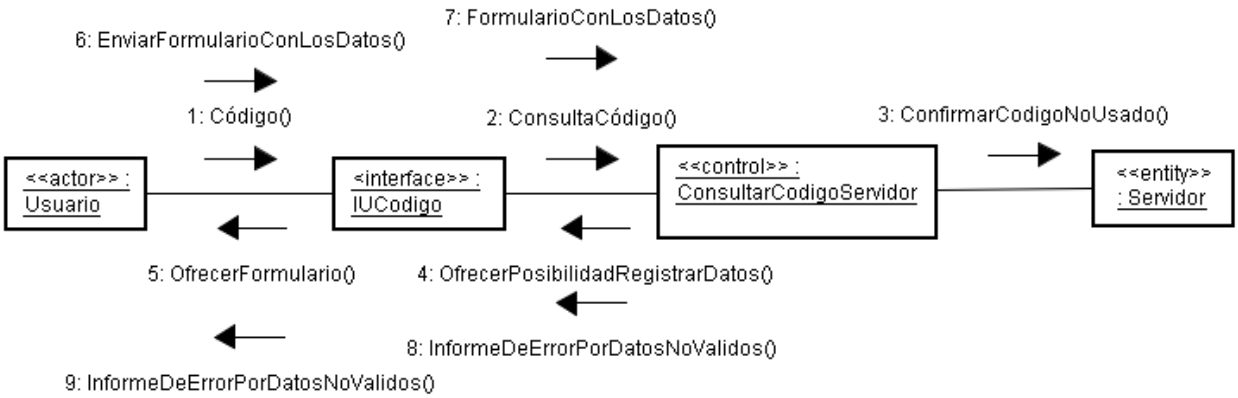
Camino alternativo 2.1: El sistema detecta que el código consultado ya está siendo utilizado e informa del error al usuario.



Camino alternativo 4.1: El usuario cancela la operación.



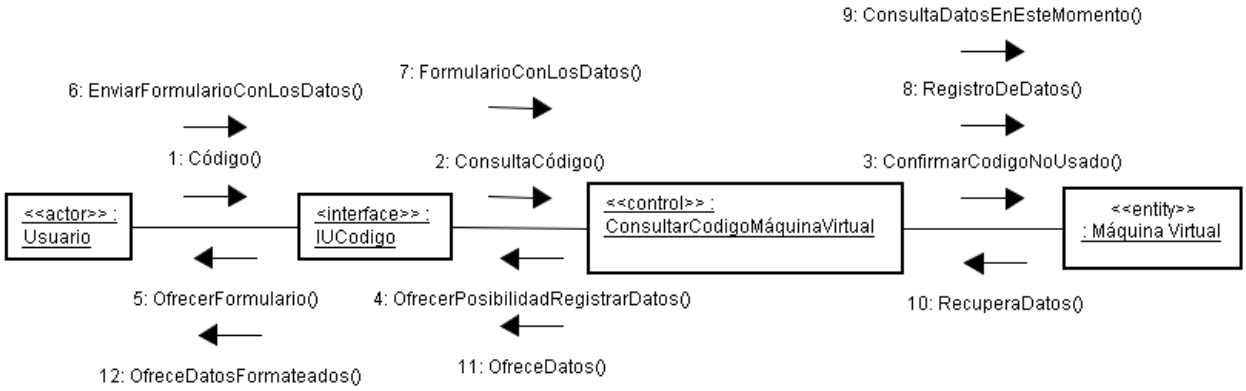
Camino alternativo 5.1: El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.



Caso de uso: Añadir código máquina virtual

Diagrama de clases:

Diagrama de colaboración:



Camino alternativo 1.1: El usuario puede partir del caso de uso "mostrar una máquina virtual", parte de la página de error ofrecerá un enlace para facilitar la adicción.

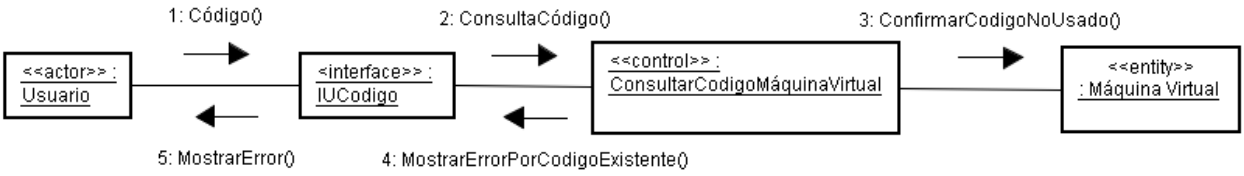
Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

Camino alternativo 1.2: El usuario puede partir del caso de uso "mostrar una máquina virtual extra", parte de la página de error ofrecerá un enlace para facilitar la adicción.

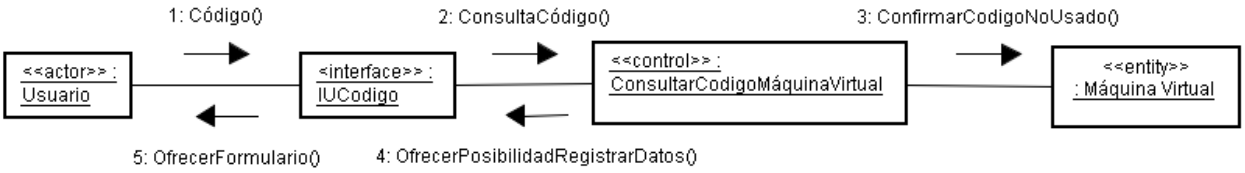
Mismo diagrama que el anterior, ya que no especificamos el origen de la petición, y el comportamiento de las diferentes partes es el mismo.

Camino alternativo 2.1: El sistema detecta que el código consultado ya está siendo utilizado e informa del error al usuario.

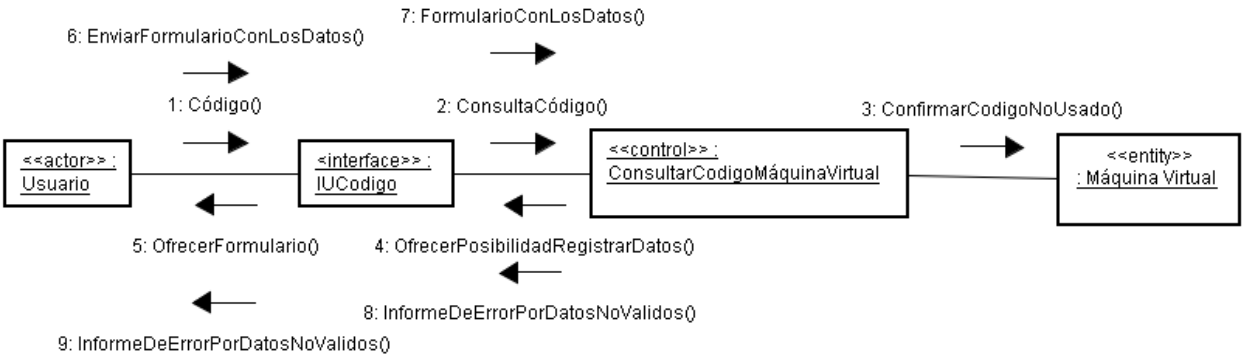
9.4 Modelo de análisis



Camino alternativo 4.1: El usuario cancela la operación.



Camino alternativo 5.1: El sistema detecta un error en los datos facilitados, muestra un error al usuario y la posibilidad de volver a empezar en el punto 3.



Caso de uso: Buscar

Diagrama de clases:

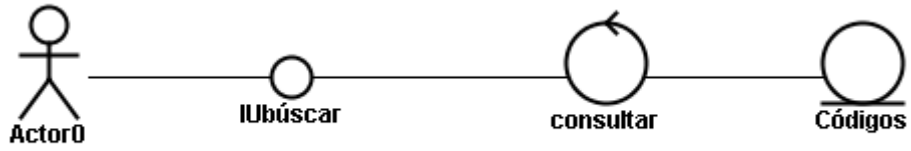


Diagrama de colaboración: