Moove-it

Descripción de la Arquitectura

Versión [1.4]

Historia de revisiones

Fecha	Versión	Descripción	Autor
31/08/2013	[1.0]	Creación del documento	Cecilia Marcora
01/09/2013	[1.1]	Revisión SQA	Ezequiel Jardim
04/09/2013	[1.2]	Actualizacion del documento	Cecilia Marcora
08/09/2013	[1.3]	Actualización del documento con la arquitectura impuesta por Rails	Cecilia Marcora
08/09/2013	[1.4]	Revisión SQA	Ezequiel Jardim

Contenido

1. INTRODUCCIÓN 3

1.1. PROPÓSITO 3
1.2. ALCANCE 3
1.3. DEFINICIONES, SIGLAS Y ABREVIATURAS. 3
1.4. REFERENCIAS 3
1.5. VISIÓN GENERAL 3

2. ARQUITECTURA IMPUESTA POR RAILS 4

2.1. INTRODUCCIÓN 4
2.2. ESTRUCTURA DE UN PROYECTO CON MVC 4
2.3. MÓDULOS 5
2.3.1. Modelo: *5*
2.3.2. Vista: *5*
2.3.3. Controlador: *6*
2.3.4. Otros módulos: *6*
2.4. GEMAS: 6
2.5. COMPONENTES DE RAILS 3 6
2.6. ATENCIÓN A UNA PETICIÓN HTTP EN LA ARQUITECTURA MVC IMPUESTA POR RAILS 7

3. NUESTRA APLICACIÓN 9

4. VISTA DEL MODELO DE DISEÑO 10

4.1. DESCOMPOSICIÓN EN SUBSISTEMAS 10
4.1.1. UI Móvil *10*
4.1.2. UI Web *10*
4.1.3. UI Plug-in Chrome *10*
4.1.4. UI Backend Administrativo *10*
4.1.5. REST *10*
4.1.6. Backend *10*
4.1.7. Lógica plug-in *11*
4.1.8. Persistencia *11*

5. VISTA DEL MODELO DE DISTRIBUCIÓN 12

5.1. DIAGRAMA DE DISTRIBUCIÓN 12
5.2. NODOS 12
5.2.1. Móvil *12*
5.2.2. PC *12*
5.2.3. Web Server *12*
5.2.4. Application server *12*
5.2.5. Herramienta externa *12*
5.2.6. Database *12*
5.3. CONEXIONES 12
5.3.1. HTTP/internet *12*
5.3.2. HTTP/LAN *12*

1. Introducción

1.1. Propósito

Este documento proporciona una apreciación global y comprensible de la arquitectura del sistema usando diferentes puntos de vista para mostrar distintos aspectos del sistema. Intenta capturar y llegar a las decisiones de arquitectura críticas que han sido tomadas en el sistema.

1.2. Alcance

Se intenta mostrar una visión global de la distribución de la arquitectura.

Será una referencia para los implementadores, para comprender mejor el producto.

Además, dará una idea estructural del proyecto, para aquellos que no esten directamente involucrados en él.

En esta primera instancia del documento, se dará a conocer la arquitectura a grandes rasgos, ya que aún no está definido el backlog del producto.

Una vez definido, se actualizará en la siguiente revisión del documento.

1.3. Definiciones, siglas y abreviaturas.

- REST: REpresentation State Transfer, estilo de arquitecura.

1.4. Referencias

- Documento de historias v1.0

1.5. Visión general

Para comenzar, se explica la arquitectura MVC impuesta por Rails, dando detalle de sus componentes y de la estructura.

Luego se explica la distribución de nuestra aplicación.

Y por último se detallan los diagramas de diseño y despliegue de nuestra aplicación, explicando cada uno de los componentes.

2. Arquitectura impuesta por Rails

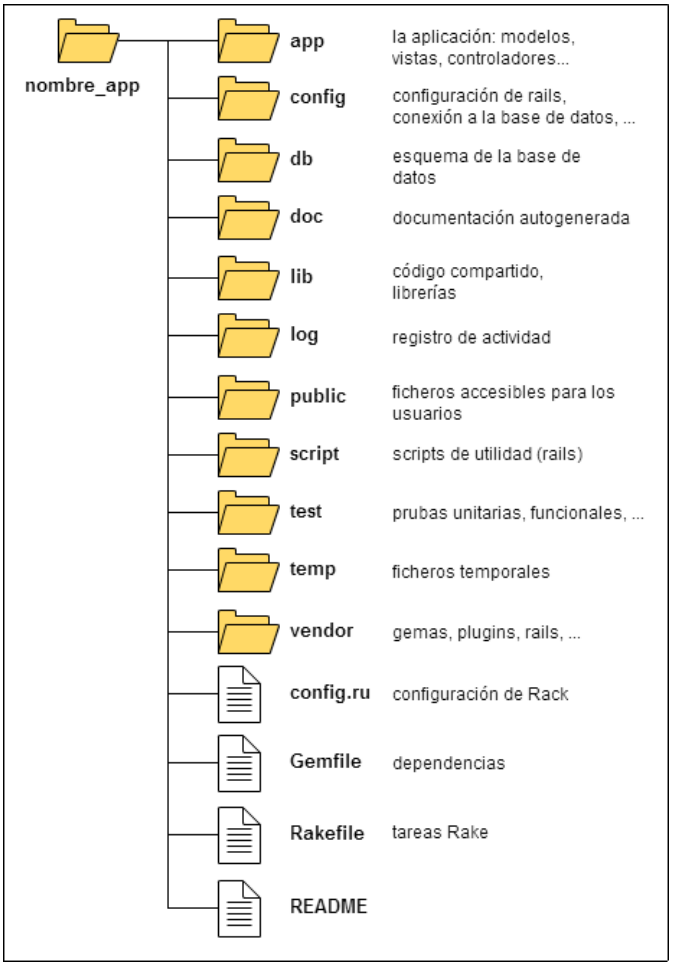
2.1. Introducción

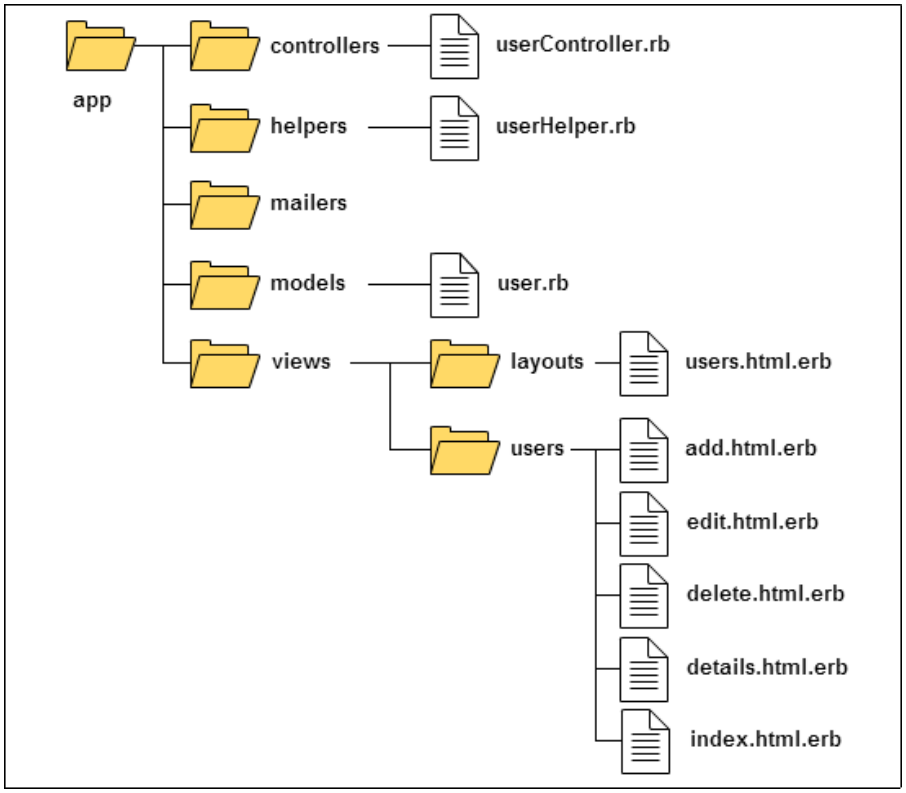
Rails es un framework de aplicaciones web Open Source (de Código Abierto) para trabajar con Ruby – lenguaje de programación orientado a objetos – ideado para programar de forma rápida y ágil.

La rapidez en el desarrollo de proyectos con Rails está fundamentada en la idea de construir la aplicación separando de forma clara las capas de Modelo (Datos), Vista (Presentación) y Controlador(Funciones, métodos …) para reducir el acoplamiento entre la lógica de negocios y la de presentación. De modo que, antes de empezar a trabajar con Rails, no está de más tener claro los conceptos que engloba la arquitectura Modelo Vista Controlador (MVC).

2.2. Estructura de un proyecto con MVC

A continuación presentamos un ejemplo de lo que sería la estructura de un proyecto Ruby on Rails.





Estas carpetas hacen que la metodología web sea bastante dinámica, ya que están separadas adecuadamente para hacer más eficiente y realizar cambios repentinos, además de que es más cómodo leer el código.

2.3. Módulos

2.3.1. Modelo:

Este código sirve para el manejo de la base de datos, le damos instrucciones para que sepa manipular la lectura y escritura de los datos del sistema.

El modelo consiste en las clases que representan a las tablas de la base de datos.
Las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar un método de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

2.3.2. Vista:

La vista es la parte de la aplicación que será mostrada al usuario.
Por lo general contiene:

- código html y css
- variables de nuestras funciones en ruby

Es necesario escribir un pequeño fragmento de código en HTML para cada método del controlador que necesita mostrar información al usuario.
Se crea una vista por cada acción de cada controlador.

2.3.3. Controlador:

El controlador es el "cerebro" de la aplicación. Decide cómo debe interactuar el usuario con el sistema, controlando el acceso desde el modelo y qué partes de la vista serán presentadas.

Las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. Los métodos del controlador son invocados por el usuario usando el navegador web.

La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController.
Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web.

- Cada controlador, una clase de Ruby
- Cada método, una acción

2.3.4. Otros módulos:

Además, Rails ofrece otros módulos, como Action Mailer (para enviar correo electrónico) o Active Resource que proporciona la infraestructura necesaria para crear de manera sencilla recursos REST, algo por lo que apuesta claramente Rails en sus últimas versiones desplazando así a otros modelos como SOAP y XML-RPC a los que se les daba soporte en versiones anteriores mediante Action Web Service.

2.4. Gemas:

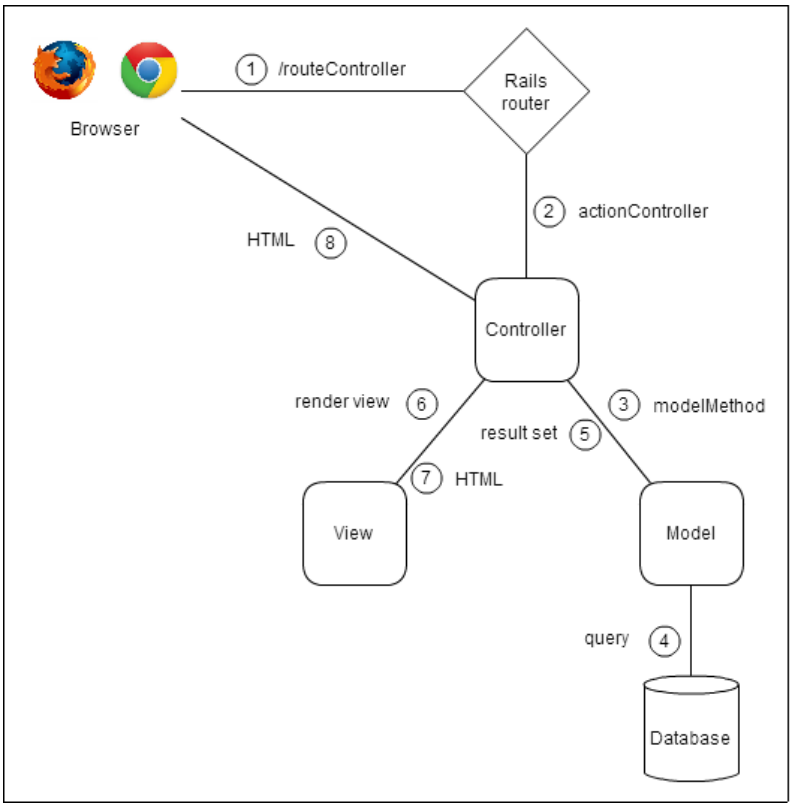
Las gemas son plugins y/o códigos añadidos a nuestros proyectos Ruby on Rails, que nos permiten nuevas funcionalidades.

2.5. Componentes de Rails 3

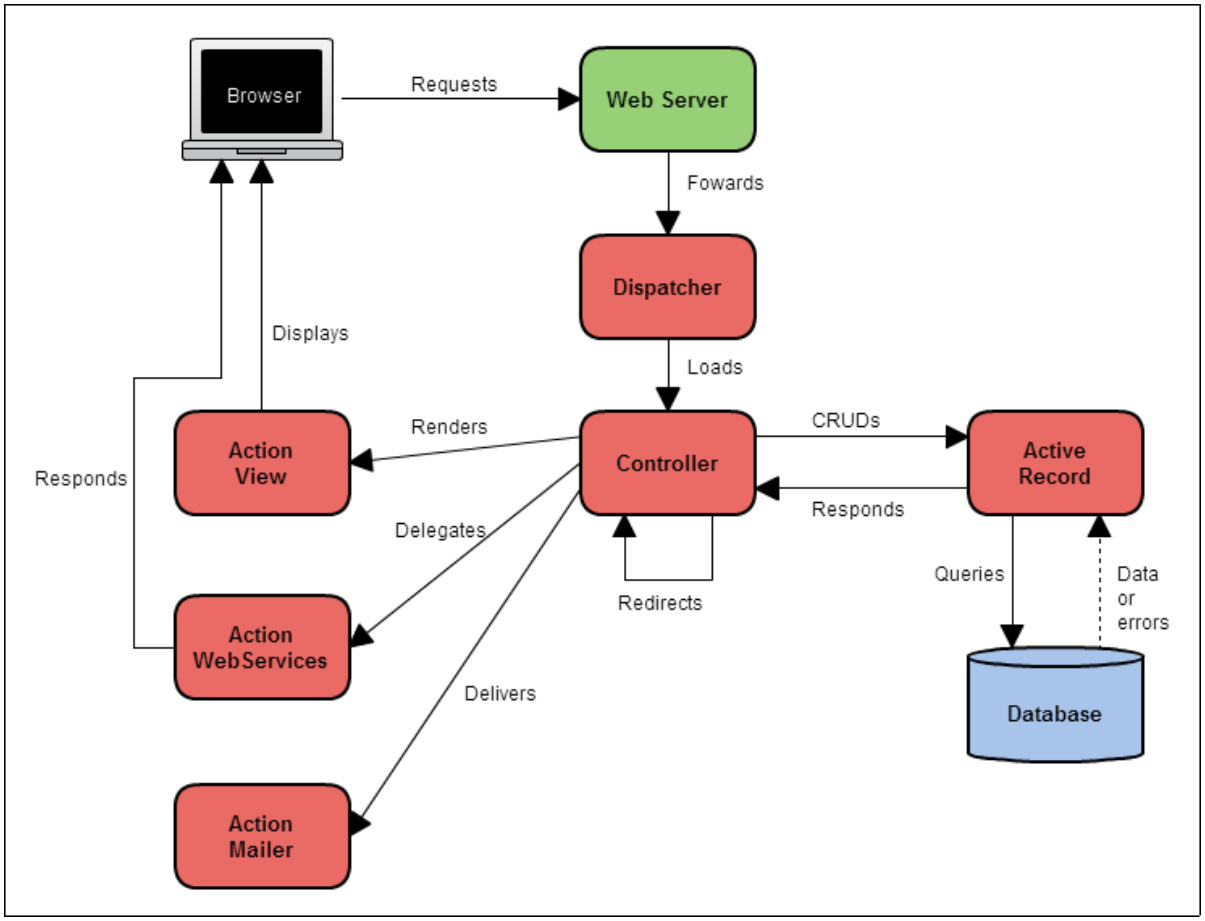
- **Action Pack** Gema que contiene tres componentes
  - **Action Controller** parámetros HTTP, sesiones, redirecciones.
  - **Action View** genera HTML o XML a partir de plantillas
  - **Action Dispatch** enruta las peticiones dentro de la aplicación o a otra aplicación Rack
- **Action Mailer** para enviar y recibir correos electrónico
- **Active Model** interfaz entre Action Pack y la capa de persistencia
- **Active Record** ORM, base de los modelos
- **Active Resource** permite utilizar recursos REST como objetos locales

- **Active Support** clases útiles y extensiones a las de Ruby
- **Railties** para crear aplicaciones y unir los demás componentes

2.6. Atención a una petición HTTP en la arquitectura MVC impuesta por Rails

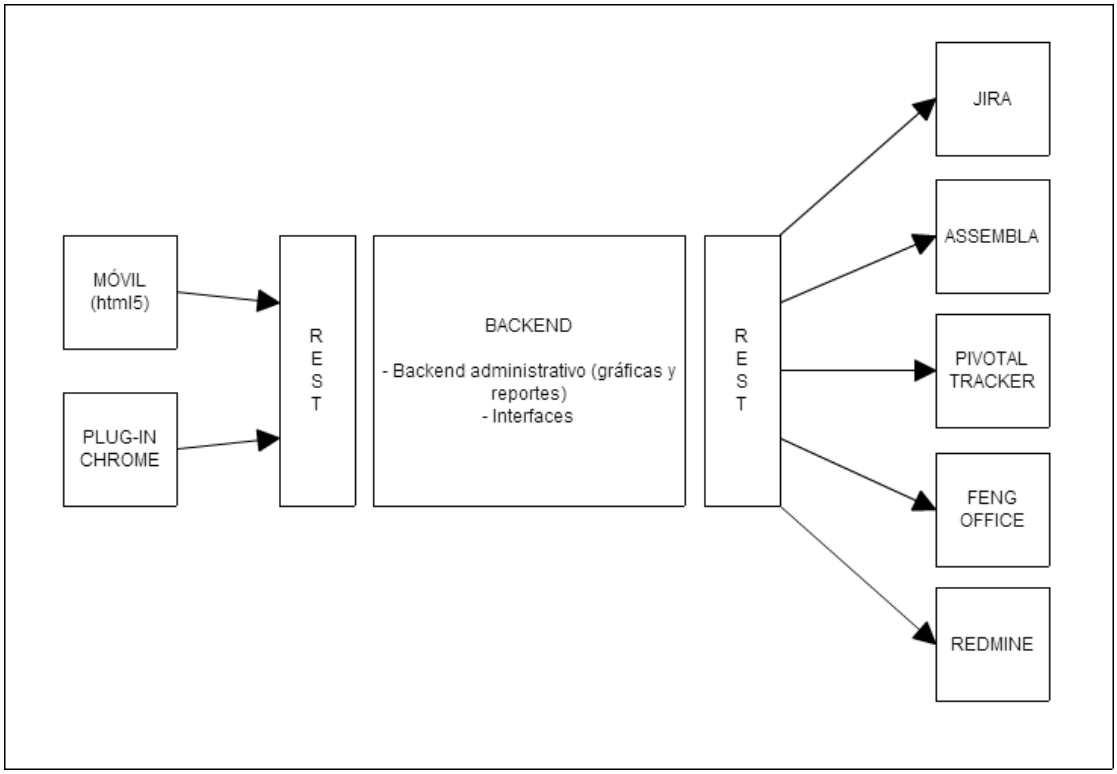


1. El explorador emite una solicitud a una URL.
2. Rails enruta la URL a la acción que corresponda del controlador que corresponda.
3. La acción pide al modelo que correponda para recuperar todos los datos necesarios para cumplir la solicitud.
4. El modelo obtiene todos los datos de la base de datos.
5. El modelo devuelve los datos al controlador.
6. El controlador captura los datos en una variable, que se pasa a la vista que correponde a la acción.
7. La vista utiliza código ruby embebido para representar la página como HTML.
8. El controlador pasa el código HTML al explorador.


Browser
Requests
Web Server
Fowards
Dispatcher
Displays
Loads
Action View
Renders
Controller
CRUDs
Active Record
Responds
Responds
Delegates
Redirects
Queries
Data or errors
Action WebServices
Delivers
Database
Action Mailer

3. Nuestra aplicación

La propuesta del cliente es la siguiente:



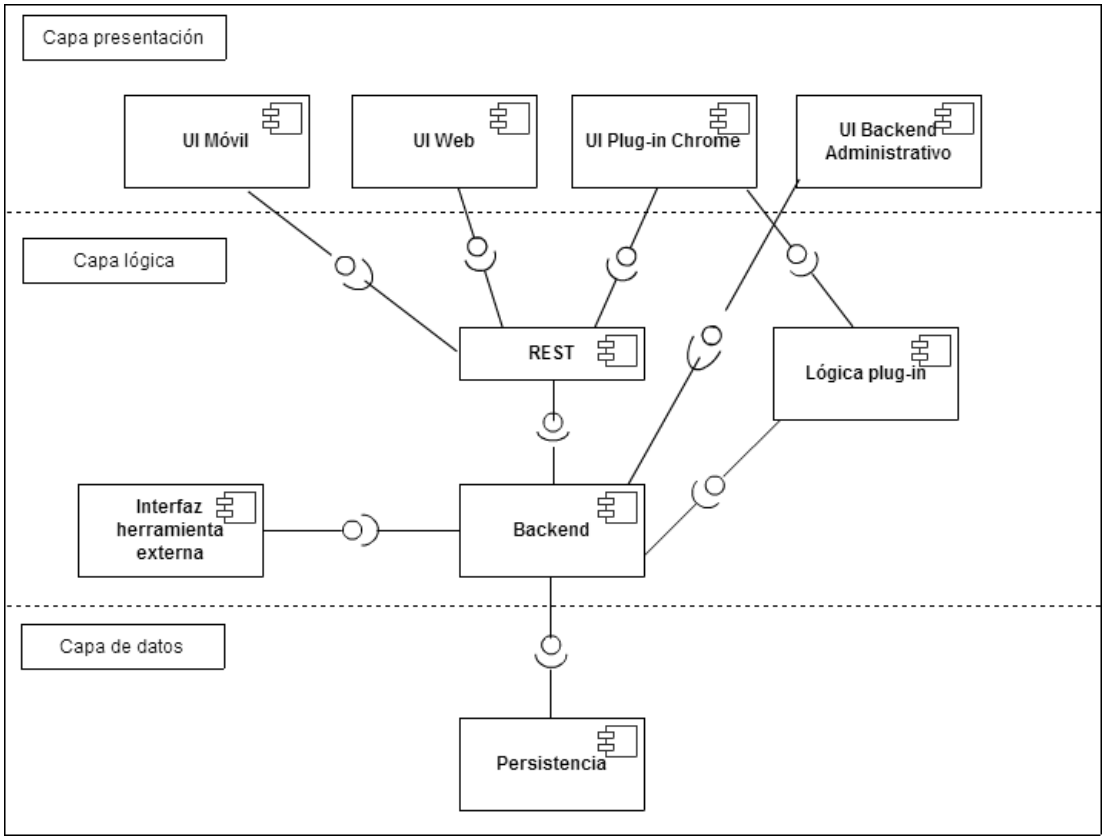
Esta propuesta nos da una idea global del producto.

El proyecto se va a adaptar a la arquitectura MVC propuesta por Rails.

Para la comunicación entre los subsistemas se va a utilizar el estilo de arquitectura REST.
Más adelante en el proyecto, se van a definir los recursos, su representación y cómo van a ser referenciados.

4. Vista del Modelo de Diseño

4.1. Descomposición en Subsistemas



4.1.1. UI Móvil

Este módulo contiene las vistas para la parte móvil. Será en HTML5.

Debe ser responsive y sumamente fácil de usar.

4.1.2. UI Web

Este módulo contiene las vistas para la parte web. Será en HTML5.

4.1.3. UI Plug-in Chrome

Este módulo contiene las vistas para el plug-in. Debe ser fácil de usar e intuitivo.

4.1.4. UI Backend Administrativo

Este módulo contiene las vistas para la parte administrativa del producto. Será en HTML5.

4.1.5. REST

Este módulo contiene la interfaz de web service, realizada con REST, para la comunicación con la parte móvil/web y el plug-in chrome.

4.1.6. Backend

Este módulo contiene toda la lógica de la aplicación.

4.1.7. Lógica plug-in

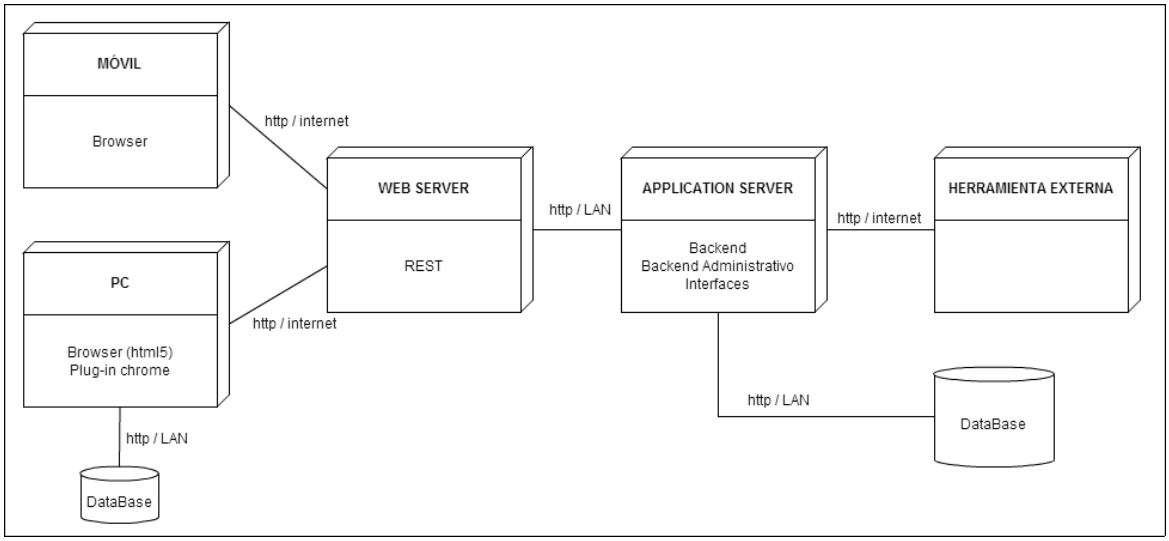
Este módulo contiene una mínima lógica para el plug-in, ya que es necesario que el cliente pueda utilizar la aplicación en modo offline.

4.1.8. Persistencia

Este módulo contiene toda gestión de datos para realizar la persistencia.

5. Vista del Modelo de Distribución

5.1. Diagrama de Distribución



5.2. Nodos

5.2.1. Móvil

Es donde se ejecuta el navegador de internet para acceder a la página de la aplicación.

5.2.2. PC

Es donde se ejecuta el navegador de internet, al igual que en el móvil, pero además se va a ejecutar el plug-in chrome para poder acceder a la aplicación.

5.2.3. Web Server

Es donde se ejecuta una interfaz con los servicios web basados en REST.

5.2.4. Application server

Es donde se ejecuta la aplicación en sí. Habrá un backend, que es el encargado de la lógica, un backend administrativo, encargado de ofrecerle al usuario una interfaz para administrar la aplicación, e interfaces para comunicarse con herramientas externas, como ser JIRA, Assembla, Pivotal Tracker, REDMINE, Open office, etc.

5.2.5. Herramienta externa

5.2.6. Database

5.3. Conexiones

5.3.1. HTTP/internet

Debe estar disponible un 99% del tiempo.

5.3.2. HTTP/LAN

Debe estar disponible un 99% del tiempo.