|  |
| --- |
| Descripción de Diseño de Software |
| SDD de SnoutPoint |
| **SnoutPoint Network**  Camilo Oviedo Lizarazo  Esteban Hernández Losada  Fabiana Díaz Cedeño  Camilo Benavides Franco  Sebastián Jiménez Nieto  Alejandra Rocha Sabogal  David Suarez Guerrero |

# Historial de Cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Historial de Cambios** | | | |
| Versión | Cambios efectuados | Fecha de Actualización | Área(s)  Encargada(s) |
| **V0.1** | Documento estructurado para su desarrollo.  Agregado 5. Introducción. | 8 de Mayo de 2015 | EAD |
| **V0.2** | Agregada la arquitectura del sistema (Numeral 6) | 25 de Mayo de 2015 | EAD |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 2. Tabla de Contenidos

Propósito: Que el lector encuentre rápidamente una sección específica del documento

# 3. Lista de Figuras

Propósito: Que el lector encuentre rápidamente una figura específica del documento

# 4. Lista de Tablas

Propósito: Que el lector encuentre rápidamente una tabla específica del documento

# 5. Introducción

El documento de Descripción de Diseño de Software (Software Design Description o SDD) es una representación del diseño del software que será utilizada para registrar la información de diseño y comunicar esa información para la construcción del código a partir del diseño [[1](#IEE091)].

Para el caso de SnoutPoint no será la excepción, pues este documento tendrá consignados todos los modelos, diagramas y artefactos de diseño relevantes para la construcción del software, con una correspondiente descripción para que el equipo de análisis y diseño, así como el equipo de desarrollo y pruebas y la gerencia puedan tener una abstracción sobre la composición del sistema en aspectos físicos, lógicos y funcionales, facilitando también la comprensión de como el sistema estará compuesto, que elementos de software y hardware requiere, cómo sería el comportamiento ideal del mismo, como se almacenará y manipulará la información y cuál será la interacción de SnoutPoint con el usuario final.

# 6. Arquitectura

Por medio de la arquitectura del sistema, se busca explicar y mostrar el comportamiento del sistema en términos generales, en un alto nivel de abstracción para poder desarrollar el diseño detallado ([ver Diseño Detallado](#_7._Diseño_Detallado)).

## Vista Lógica del Sistema

Por medio del Diagrama de componentes se mostrará los diferentes artefactos que representan elementos concretos del mundo real que son resultado de los procesos de desarrollo del equipo de análisis y diseño [[2](#UML14)]. Aquí se tienen en cuenta también aspectos de interfaces, puertos y relaciones entre componentes.

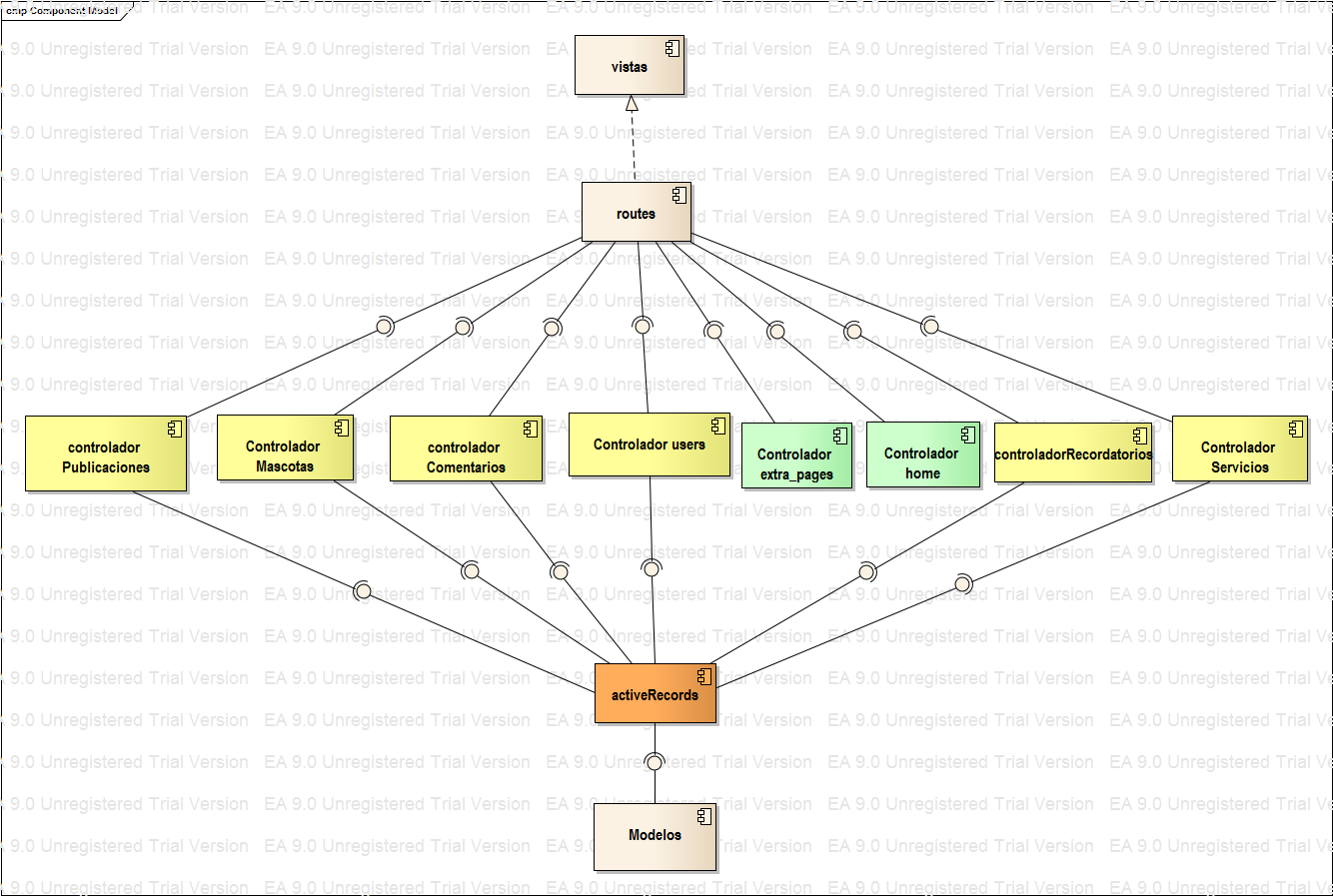
En este caso, por tratarse de una página web, se utiliza un patrón de Modelo-Vista-Controlador que facilita el desarrollo en Ruby on Rails [[3](#Ray15)], por lo que los componentes del sistema se encontrarán segmentados entre Modelos, Vistas y Controladores, según el comportamiento buscado y la información mínima que debe tener el sistema según los casos de uso y requerimientos levantados anteriormente.

Ilustración 1. Diagrama de Componentes

El Diagrama de componentes muestra que existe una serie de modelos, los cuales contienen todos los datos e información que sea necesaria guardar y buscar su persistencia para su reutilización en las operaciones del sistema, almacenándolo como registros en una base de datos.

Gracias al framework que ofrece Ruby on Rails, por medio del componente de Active Records que provee una interfaz para realizar consultas desde los modelos [[4](#Dav151)], facilitándole a los controladores el obtener información cada que se les solicite y poder aplicar las modificaciones dentro de los modelos, asegurando la consistencia de la información.

Se determina que se debe tener un controlador para las entidades principales del modelo: Usuario, Mascota, Servicios, Publicaciones, Comentarios y Recordatorios. Cada controlador se encarga de realizar operaciones sobre la información de los modelos según las solicitudes que reciba por parte del cliente. También se tienen un controlador de home y uno de extra pages, los cuales se encargan de cargar la información de páginas que no requieran de entidades u operaciones lógicas, como la redirección a la página de Términos y Condiciones de SnoutPoint o la carga de la página inicial de SnoutPoint.

El componente de routes [[5](#Dav152)] es el que determina cuál es el controlador que se va a utilizar según lo que el usuario realice dentro de la página y cual vista se debe desplegar según la operación realizada dentro de la página. Este determina cual vista debe cargarse para el cliente y qué acciones de qué controlador se deben ejecutar para el despliegue de esta vista.

## Vista Física del Sistema

La vista física del sistema servirá para entender cuáles son los componentes físicos en los cuales el sistema va a ser instalado y realizará sus operaciones. [[1](#IEE091)] Para realizar esta vista, se utiliza un diagrama de despliegue que explica a grandes rasgos cual es la composición a nivel de hardware y archivos de SnoutPoint, para entender cómo será instalado y como debe ser su organización interna a nivel de carpetas, paquetes y archivos.

Dado que uno de los principales requerimientos del sistema es que debe cumplir una arquitectura de cliente-servidor [[6](#Pav)], se mantendrá la aplicación en un servidor web que podrá ser accedido mediante HTTP sobre el protocolo TCP/IP por parte de los clientes que deseen utilizar nuestra red social.

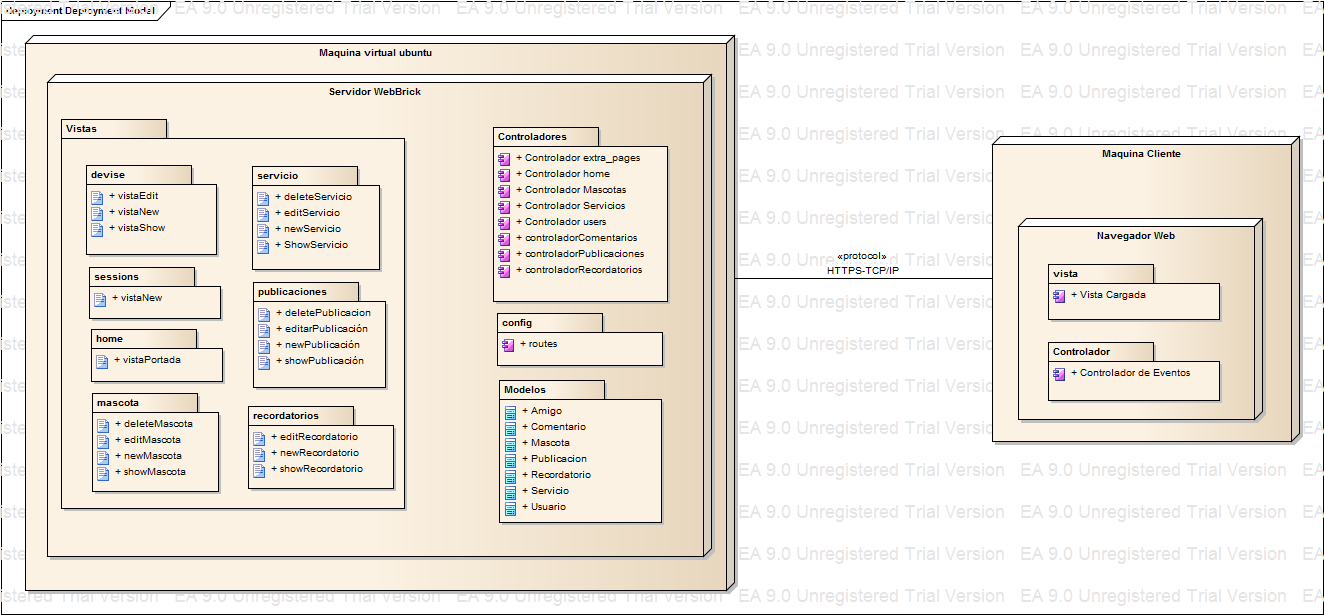


Ilustración . Diagrama de Despliegue

**Describir el diagrama.**

## Vista de Procesos del Sistema

En la vista de procesos de sistema se muestra de manera general, cuales son las acciones que el usuario puede realizar mientras se encuentra dentro del sistema [[1](#IEE091)]. Esto es, mostrar que cuáles son las posibles acciones que se pueden realizar el usuario y cuál es la salida o resultado de esta acción. En el caso de SnoutPoint, se determina la acción como la operación que se realiza en una vista y el resultado como la vista cargada con la información determinada.

Propósito: que el desarrollador sepa cuáles son los principales procesos en los cuales el usuario interactúa con el sistema.

Contenido: Representar a través de diagramas de actividades las principales acciones que los usuarios realizan en el sistema. Estos diagramas deben ser consistentes con el Diagrama de Navegabilidad de la interfaz gráfica. Los diagramas deben ir acompañados de texto o tablas que expliquen cada parte del diagrama.

Los diagramas a utilizar pueden ser de los siguientes tipos:

* Diagramas de Actividad
* Diagrama BPMN
* Diagrama de Estados (del sistema completo)

Puede utilizar más de un tipo de diagrama, pero sin ser redundante en la información presentada. Por ejemplo, evite hacer un diagrama de Actividad y uno de Estados que representen las mismas acciones.

Referencias: [4]

# 7. Diseño Detallado

Propósito: Mostrar los detalles más importantes del diseño de bajo nivel del sistema.

Contenido: Ver subsecciones

## Estructura del Sistema

Propósito: Que el desarrollador comprenda los detalles de la estructura de todos los componentes de software del sistema. Proveer una guía para implementar las clases utilizadas en el software.

Contenido: Para cada componente de software identificado en la sección 6.2, un diagrama de clases acompañado de texto o tablas que lo expliquen. Todas las clases, métodos y asociaciones deben estar explicados.

## Comportamiento del Sistema

Propósito: Que el desarrollador entienda los detalles de cómo cambia el sistema en el tiempo. Proveer una guía para implementar los métodos más complejos de las clases del software.

Contenido: Para cada acción compleja descrita en la sección 6.3, un diagrama de secuencia, de comunicación o de estados que describa cómo se realiza dicha acción. Dichos diagramas de secuencia deben utilizar instancias y métodos de las clases descritas en 7.1.

Incluya diagramas de secuencia sólo para aquellas acciones que sean suficientemente complejas. Por ejemplo, si una acción involucra sólo una o dos llamadas a métodos, no vale la pena describirla en un diagrama.

No describa en estos diagramas lo que sucede dentro de las librerías que Ud. utilice y que no sean de su creación.

## Persistencia

Propósito: Que el desarrollador entienda cómo se van a almacenar los datos del sistema en forma persistente. Proveer una guía para crear las bases de datos, archivos u otros medios de almacenamiento persistentes utilizados por el software.

Contenido: Diagramas que describan la forma en que la información será almacenada, ya sea en bases de datos, archivos, etc. Los diagramas que se pueden usar son: Diagramas E-R, Diagramas Relacionales o Diagramas de Clases con estereotipos [5]. Los elementos de los diagramas deben ser descritos en párrafos o tablas.

## Interfaz de Usuario

Propósito: Que el desarrollador entienda todos los detalles de la interfaz de usuario. Contenido: Diagramas que expliquen las principales pantallas de la interfaz gráfica y la navegación entre ellas. Por ejemplo, diagramas de navegación, diagramas de estado, Diagramas de Flujo de Interfaz de Usuario [6]. Todos los diagramas deben ir acompañados de texto o tablas que expliquen sus componentes

Nota: El concepto de interfaz de usuario es más general que el de interfaz gráfica, por lo cual en una situación real podría ser necesario describir aspectos adicionales al de la interfaz gráfica. Para efectos de esta entrega sólo se le pide especificar esto último.

# 8. Referencias

Indique aquí todas las referencias bibliográficas utilizadas en el documento. Utilice formato IEEE o APA para definirlas. Para administrar automáticamente las referencias, se recomienda el uso de la herramienta Zotero (www.zotero.org).

----

Referencias usadas en el formato

[1] ISO/IEC/IEEE 42010

[2] IEEE Std 1016-2009

[3] RUP. Software Architecture Document <http://www.ts.mah.se/RUP/RationalUnifiedProcess/webtmpl/templates/a_and_d/rup_sad.htm>

[4] Philippe Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture. <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

[5] Scott Ambler, UML Data Modeling Profile. <http://www.agiledata.org/essays/umlDataModelingProfile.html>

[6] Scott Ambler, User Interface Flow Diagrms. <http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm>