Diseño

Decisiones de Diseño e Implementación

Criterios de Implementación

Decisiones de Funcionalidad

- Las bajas de los datos en la base, serán bajas lógicas, no se borrara el dato físico.
- Un lote no se dará de alta, hasta que éste llegue a la empresa.
- Tanto en fardo como en maquinaria, si el tipo real, no existe en la base, se deberá solicitar al DBA (Administrador de base de datos), que registre el tipo antes de dar de alta dichos objetos.

Decisiones de Interfaces

- Las interfaces serán intuitivas y agradable a la vista.
- Permitirán el acceso autentificado a usuarios registrados.
- Herramienta utilizada para las interfaces, sera BOOTSTRAP 3.

Diseño arquitectónico

Arquitectura Cliente - Servidor

El diseño constara de una arquitectura Cliente - Servidor. Distintos usuarios con distintos perfiles podrán conectarse con el Servidor de forma concurrente desde diferentes plataformas.

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.

Cliente Liviano: Dado que todos los procedimientos significativos se realizarán en el servidor, solo tendremos clientes livianos, navegadores web.

Aplicación Web

La aplicación será web, debido a la necesidad de la empresa de poder conectarse con el sistema desde cualquier parte, dentro y fuera de las instalaciones de Furhmann.

El servidor que utilizaremos será de tipo Web (HTTP Apache), debido a su gran conjunto de funcionalidades y portabilidad.

Implementar la arquitectura Cliente – Servidor atraves de una aplicación web, nos brinda portabilidad, clientes podrán acceder al sistema sin depende de que plataforma o navegador este utilizando. También nos permite acceder al sistema sin necesidad de instalar la aplicación, y desde cualquier dispositivo con internet.

Como mencionamos anteriormente el software se implementara en Python con el framework Django, gracias a éste, contaremos con la arquitectura MVC (Modelo Vista Controlador), que se basa en 3 capas, vista-modelo-controlador. Django llama a este patrón MTV.

Diagrama de Bloques

Para empezar a entender MTV debemos fijarnos en la analogía con MVC.

- El modelo en Django sigue siendo modelo
- La vista en Django se llama Plantilla (Témplate)
- El controlador en Django se llama Vista

El modelo: El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

La vista: La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados. El ORM de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Lo más importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.

La plantilla: La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML (también XML, CSS, JavaScript, CSV, etc).

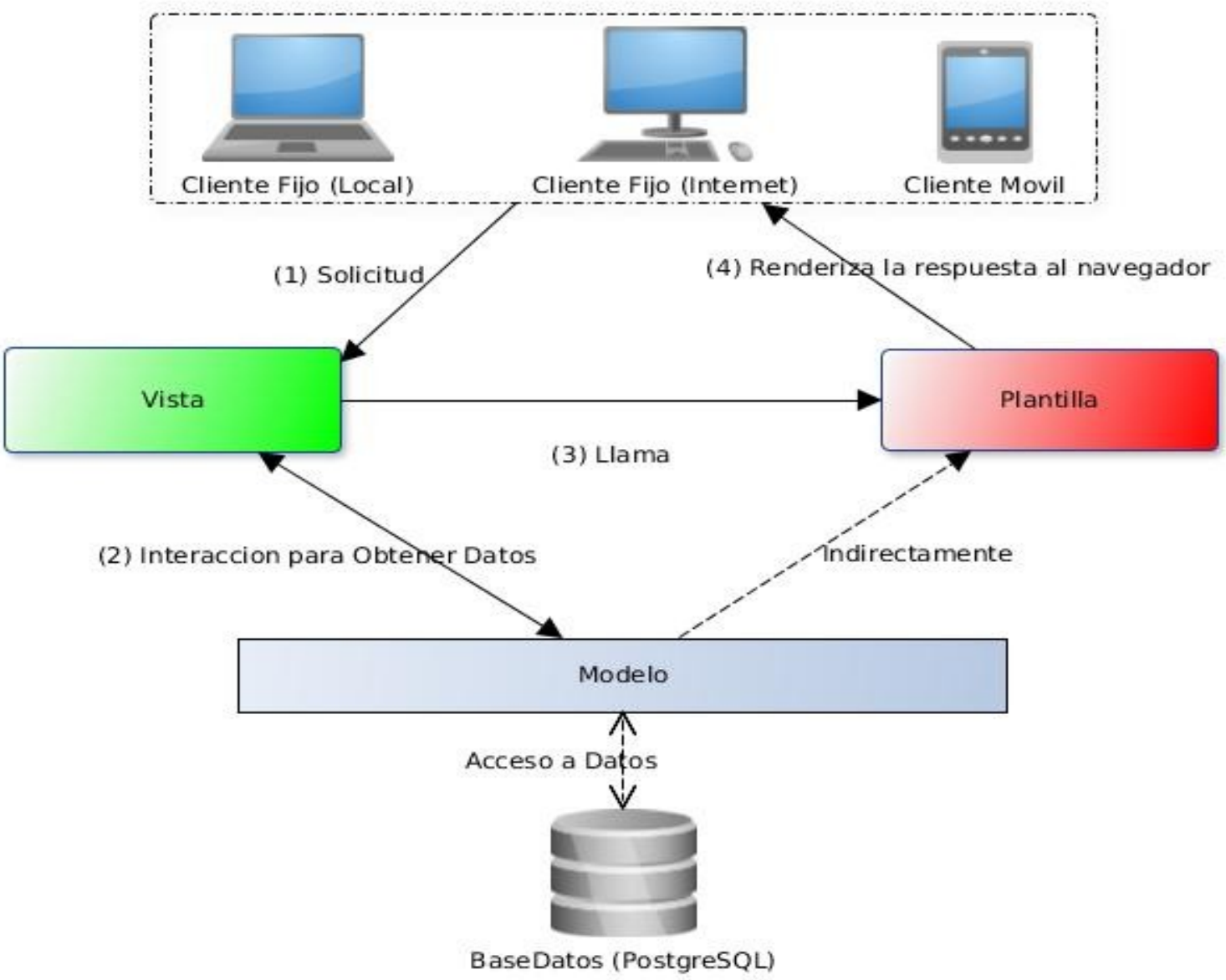

Diagrama de Bloque reflejando el funcionamiento del Framework Django.

Diagrama de Despliegue

En el diagrama de despliegue se puede observar que habrá un único servidor, que contendrá tanto el servidor HTTP APACHE, y el servidor de BASE DE DATOS (PostgreSQL), por otro lado se podrán conectar tantos clientes como el servidor soporte concurrentemente. Los clientes podrán establecer una conexión al servidor por medio de una red HTTP, o por medio de una red LAN. A su vez estos clientes podrán ser “móviles” (celulares, tablet, etc) o clientes “fijos” (PC, notebook, etc).

Los clientes enviaran consultas al servidor por medio de un navegador web. Éste ultimo se encargará de mostrar los datos devueltos por dicho servidor.

El servidor soportara concurrencia, es decir, varios clientes (1 .. n), podrán comunicarse de forma paralela con el servidor. Tendremos un único servidor accesible por internet, o por red Local, que nos dará acceso a la base de datos.

Todos los datos e información de la base estará alojada en un único servidor. Que se encargara de hacer backup y mantener la información actualizada y disponible. *El servidor esta compuesto por el servidor de Base de datos (PostgreSQL) y el servidor web HTTP Apache.*

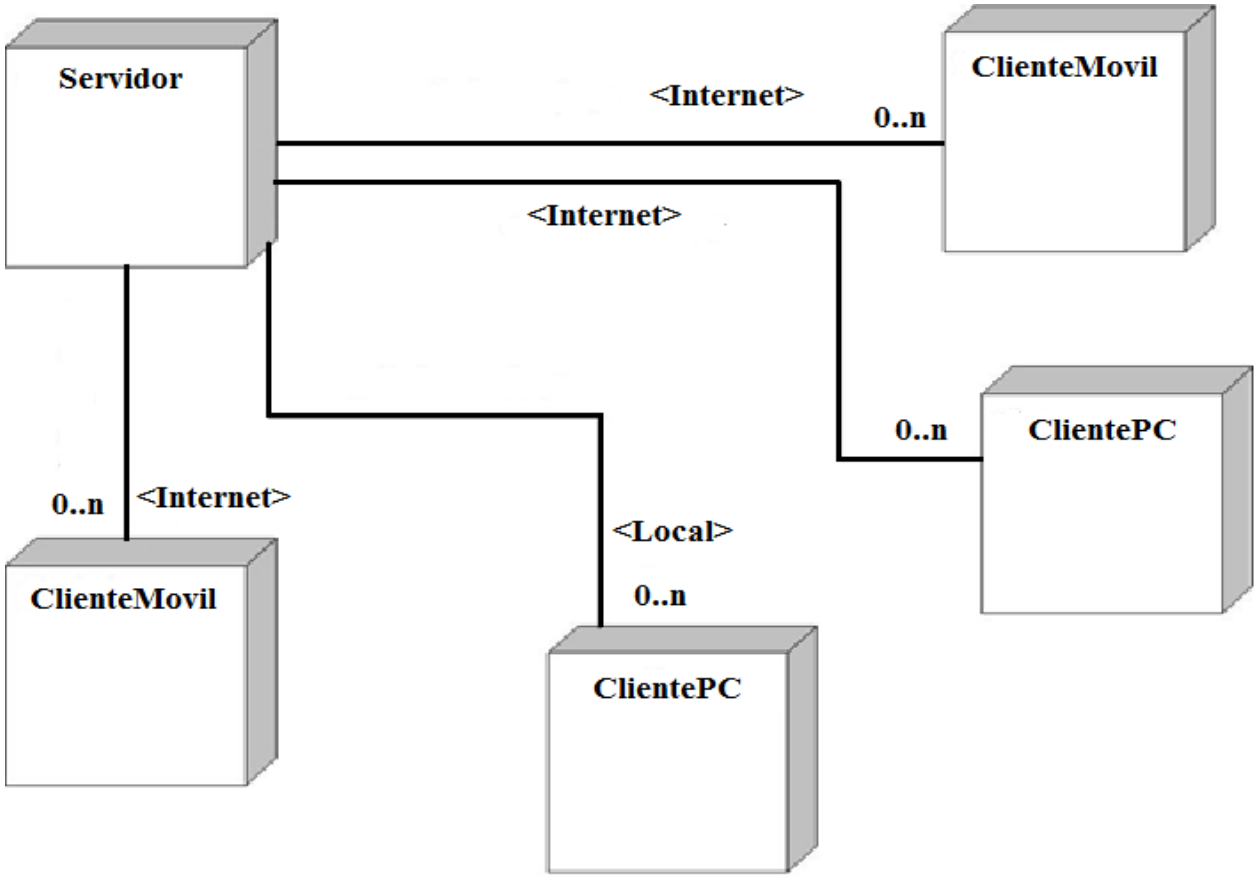

Diagrama de Paquetes

Dado que el servidor sera el único componente que implementaremos, basamos el diagrama de paquetes en una única arquitectura de capas que refleja dicho servidor.

Éste llevara a cabo todas las funcionalidades, brindando servicios a los clientes que se conecten con él. Debido a esto el servidor debe hacerse cargo de mantener todos los datos resguardados, mantener seguridad ante posibles amenazas, brindar vistas de acceso a la lógica de negocio para cada perfil y por ultimo permitir la persistencia de datos por un tiempo indeterminado.

Si bien Django nos provee 3 capas del patrón “MVC”, decidimos implementar 2 capas más por ser una aplicación Web. Básicamente la arquitectura se basa en 5 nivel o capas que nos permites realizar una implementación coherente y con bajo acoplamiento.

Una primera capa de seguridad (1er Capa), donde se validara datos de acceso al sistema. Una segunda capa de vistas (2er Capa), donde cada perfil tendrá vistas acorde a sus funciones dentro de la empresa (es decir interfaces asociadas a perfiles), que interactúan internamente con la aplicación de negocios (3er Capa) que se encarga de toda la lógica de negocio requerida para el funcionamiento correcto del sistema.

La 4ta Capa, consta del ORM que nos provee Django para la interacción directa con la base de datos atraves de código de Python. Esta capa funciona de intermediario entre la 3era capa y la 5ta. Permite a la capa de Negocios comunicase con la DB y acceder a los datos.

La 5ta Capa se encarga de la persistencia de datos en la base.

Nivel 1: Capa de Seguridad (validación y control de usuario).

Nivel 2: Capa de Presentación (vistas dependiendo perfil).

Nivel 3: Capa de lógica de Negocio (Permite correcto funcionamiento).

Nivel 4: Capa de acceso a Datos (ORM proveído por Django).

Nivel 5: Capa de datos (Persistencia de Datos PostgreSQL).

Gracias a al bajo acoplamiento que logramos con la implementación de 5 niveles en el patrón arquitectónico capas, la mantenibilidad del sistema y las pruebas serán mas eficientes y fáciles.

Diagrama: DiagramaPaquetes.