**Universidad Tecnológica de Nezahualcóyotl**

División de Tecnologías de la Información y Comunicación Área Sistemas Informáticos

Gráfico de proyección solar

Descripción generada automáticamente con confianza baja

**Materia:**

Desarrollo móvil Integral

**Docente:**

Jovan del Prado López

**Integrantes:**

* Castro López Jaime
* Piña López Brayan
* Rocha Cortés Alondra

**Grupo:**

ITIC-1001V

**Actividad:**

**Plan del Proceso de Desarrollo**

**Fecha de entrega:**

26 de enero del 2022

# **Plan de Proceso de Desarrollo**

# **Definición de Reglas de Negocio**

Tomando en cuenta que la aplicación que vamos a desarrollar como una aplicación que va a dar servicio vía una app para realizar citas en una veterinaria de mascotas, para definir las reglas de negocio, veamos repasemos algunas definiciones de los Roles de negocio:

* Del Sistema:
  1. Solo se acepta 2 tipos de mascotas (Perros y gatos)
  2. Solo son citas con veterinario, no venta de productos
* El cliente:
  1. Registro de mascota en la app
  2. Registro de cita con veterinario en la app
  3. Consulta del registro de su mascota en la app
* Administrador de sistema:
  1. Revisa y completa registro de mascotas
  2. Depura registros de mascotas
  3. Borra registros de mascotas
* Médico Veterinario
  1. Consulta registros de mascotas
  2. Actualiza el diagnóstico de mascotas
  3. Actualiza el tratamiento de mascotas

**Reglas de negocio**

1. El cliente puede registrar una mascota, en otro caso la puede generar o completar el administrador del sistema

* **Si** se necesita registrar una nueva mascota, **entonces** el cliente debe registrar la mascota, **de lo contrario** la puede registrar el administrador del sistema.

1. Para las citas médicas de mascotas, el cliente puede generar la cita en línea en otro caso la puede generar el administrador

* **Si** se necesita registrar una cita con Veterinario, **entonces** el cliente debe registrar la cita, **de lo contrario** la puede registrar el administrador del sistema.

1. Un cliente puede tener varias mascotas

* **Si** un cliente tiene varias mascotas **entonces** el cliente debe crear 1 registro por cada mascota.

1. El Administrador de sistema es el único que puede depurar un registro

* **Si** se requiere depurar o modificar un registro, **entonces** el administrador debe depurar o modificar un registro de mascota

1. El Administrador de sistema es el único que puede borrar un registro

* **Si** se requiere borrar un registro, **entonces** el administrador debe borrar el registro de mascota

1. El Médico Veterinario es el único que puede actualizar el diagnóstico de las mascotas

* **Si** se tiene un diagnóstico de la cita, **entonces** el Médico Veterinario debe actualizar el diagnostico en el registro de la mascota

1. El Médico Veterinario es el único que puede actualizar el tratamiento de las mascotas

**Si** se tiene un tratamiento resultado de la cita, **entonces** el Médico Veterinario debe actualizar el tratamiento en el registro de la mascota

# **Justificación de la Arquitectura**

Con MVVM organizaremos nuestra app en tres módulos o partes distintas.

Model: Representa la parte de datos, es decir, cuando recuperamos de una base de datos o de un servicio web, toda esa información la almacenaremos en modelos de datos.

View: Es la parte de la UI, los XML, las activities y los fragments. Estos actuarán como siempre, ejecutando acciones por ejemplo al pulsar un botón, pero no realizarán las acciones, se suscribirán al view model y este les dirá cuándo y cómo pintar.

ViewModel: Este sería la conexión entre el modelo y la vista. Las vistas se suscriben a sus respectivos viewModels y estos al percatarse de que el modelo ha sido modificado lo notificarán a la vista.

Kotlin usa un View Binding para realizar acciones entre nuestras vistas y el modelo, con esta arquitectura podremos hacer uso de un ViewModel que dará las instrucciones para comunicar los datos y modificará tanto la vista como el modelo.

# **Diagrama de la arquitectura:**

Diagrama

Descripción generada automáticamente

Ilustración 1: Arquitectura MVVM

# **Patrón de Diseño**

El patrón de diseño Factory Method (Método Fábrica) sirve para abstraer el proceso de creación de un objeto. A raíz de una clase se crearían todos los objetos con las propiedades y funciones tanto del objeto padre como de los hijos.

Este patrón consideramos que es el más adecuado para nuestra aplicación, ya que, en nuestra aplicación la base de datos tiene muchas propiedades particulares, las colecciones tienen documentos y cada uno de ellos puede ser diferente, pese a compartir la misma colección, cada documento puede tener diferentes atributos. Este patrón de diseño es adecuado para nuestra aplicación gracias a esta propiedad que tiene nuestra base de datos, el patrón de diseño crea un objeto padre y de este pueden salir diferentes tipos de objetos hijo, sin la necesidad de crear varios objetos padres.

Imagen que contiene Diagrama

Descripción generada automáticamente

Ilustración 2: Patrón de Diseño

# **Justificación del Framework**

**Kotlin**

Para nuestra aplicación nos proporciona una compatibilidad muy grande, la base de datos que usaremos será Firebase y tiene un soporte muy amplio para Kotlin, permitiendo un desarrollo más amplio sobre futuras adiciones de módulos.

La base de datos al estar en la nube nos permite tener una gran compatibilidad tanto en la web como en móvil cumpliendo con el objetivo de la materia.

Ventajas del Framework

* Ofrece una 100% interoperable con Java, pudiendo hacer uso de cualquier librería escrita en Java.
* Este tipo de framework es muy seguro, no acepta valores nulos, ofreciendo una mayor seguridad en las aplicaciones.
* La codificación en Kotlin es muy corta, concisa y directa. Mismo trabajo con menos líneas de código que en Java.
* La curva de aprendizaje es más fácil que otros lenguajes de programación.



Ilustración 3: Framework