Diseño de la base de datos

Sirius

Protectora animal

Selene González Curbelo
Mariana Surós Álvarez

05/11/2023

ULPGC
Escuela de Ingeniería Informática

Contents

Selene González Curbelo
Mariana Surós Álvarez

Introducción

En este informe se enfoca en los pasos clave que se deben seguir para diseñar y seleccionar la base de datos adecuada para el proyecto final. Este paso es fundamental, ya que la elección y diseño adecuado de la base de datos tiene un impacto significativo en el funcionamiento, rendimiento y escalabilidad del proyecto en su conjunto.

1. Análisis de modelos y componentes de persistencia

Modelos

Usuario. Representa a los usuarios de la aplicación. Incluiría atributos como nombre, dirección de correo electrónico, contraseña y cualquier otro dato relacionado con la cuenta de usuario. Se debe considerar cómo se manejará la autentificación y gestión de sesiones.

Animal. Modelo central de la aplicación que representa a los animales disponibles para adopción. Incluiría atributos como tipo, nombre, edad, descripción, estado (adopción, acogida), información clínica (para los empleados de la protectora), raza, tamaño, características (color, actitud), fotos y más. Puede tener relaciones con otros modelos, como eventos o noticias relacionados con animales específicos.

Evento/Noticia. Dependiendo de la gestión de eventos y noticias relacionados con la protectora, se separará en dos modelos. Incluiría atributos como título, descripción, fecha, imágenes y detalles relevantes.

Petición de acogida/apadrinamiento. Dependiendo de la gestión de apadrinamiento y acogida de la protectora, se separará en dos modelos. Este modelo registraría las solicitudes pertinentes y contendría información como el animal solicitado, el solicitante y el estado de la solicitud.

Notificación. Informa a los usuarios sobre eventos importantes o actualizaciones relacionadas con los animales.

Pago. Dependiendo de la gestión de apadrinamiento o donaciones de la protectora, se necesitará un modelo para rastrear transacciones financieras.

Componentes

Patrón DAO. Se utilizará para separar la lógica de acceso a la base de datos de la lógica de negocio de la aplicación. Al asilar la lógica de acceso a la base de datos, facilita la modularidad y el mantenimiento. Además, proporciona un mecanismo consistente para interactuar con la base de datos, lo que simplifica el código y reduce la duplicación.

Entidades. Representan la estructura de datos utilizada en la aplicación, definiendo cómo se almacenan y organizan los datos en la base de datos. Además, aseguran la integridad de los datos en la aplicación.

Repositorios. Se utilizan para simplificar y abstraer la lógica de acceso a datos. Proporcionan métodos que permiten a la capa de presentación acceder a los datos de manera coherente y abstraen la complejidad de la interacción con la base de datos.

2. Tecnología de base de datos

Después de una evaluación exhaustiva de nuestras necesidades de almacenamiento y persistencia de datos, hemos optado por Room como la tecnología principal para la base de datos de nuestra aplicación. La elección de Room se basa en varias razones sólidas que la destacan en comparación con otras opciones disponibles, como Firebase, SQLite, Realm, DataStore, entre otras.

Room ofrece una abstracción sólida de la base de datos y reduce significativamente la cantidad de código repetitivo necesario para gestionar la persistencia de datos. Esto se traduce en un desarrollo más rápido y limpio, permitiendo al grupo focalizar en la lógica de negocio y mejorar la experiencia del usuario. En comparación con SQLite puro, Room proporciona una interfaz más intuitiva y de alto nivel, lo que agiliza el desarrollo y facilita la comprensión del código.

Además, su integración perfecta con el lenguaje de programación Kotlin, ampliamente utilizado en el desarrollo de aplicaciones Android, es una ventaja significativa para nuestro equipo. Al estar especialmente diseñado para trabajar con Kotlin, Room facilita la escritura de código limpio y legible, y permite aprovechar al máximo las características y ventajas de este lenguaje moderno.

También, Room proporciona una capa adicional de seguridad y aísla la aplicación de la complejidad subyacente de SQLite. Al proporcionar un conjunto de herramientas y técnicas para interactuar con la base de datos de forma segura, Room es esencial para mantener la integridad y seguridad de los datos sensibles que manejamos, especialmente en un contexto tan importante como la adopción y apadrinamiento de animales.

A pesar de que Room se enfoca en simplificar el acceso a bases de datos SQL, sigue siendo altamente personalizable y versátil, lo que nos permite adaptarlo a nuestras necesidades específicas de almacenamiento y consulta de datos.

Cuando comparamos Room con otras opciones como Firebase, destaca la flexibilidad y la capacidad de personalización. Aunque Firebase ofrece una solución integral que incluye bases de datos en tiempo real y funciones de autenticación, Room proporciona una mayor flexibilidad en cuanto a la estructura de la base de datos y las operaciones que se pueden realizar, lo que se alinea mejor con nuestras necesidades específicas.

En resumen, la elección de Room como tecnología principal para la base de datos de nuestra aplicación es una decisión estratégica que se basa en su capacidad para proporcionar una abstracción sólida y eficiente de la base de datos, su integración perfecta con Kotlin, su contribución a la seguridad de los datos y su versatilidad para adaptarse a nuestras necesidades específicas. Esta elección es crucial para el desarrollo exitoso y la operación segura y confiable de nuestra aplicación de gestión y promoción de adopciones y apadrinamientos de animales en la protectora.

3. Integración de la lógica de persistencia

Módulo de consulta de animales

Este módulo se centraría en brindar a los usuarios una experiencia de navegación y búsqueda de animales en adopción. Los usuarios pueden acceder a esta vista sin necesidad de crear una cuenta o iniciar sesión. La capa de persistencia de este módulo se encargaría de manejar la consulta y visualización de los detalles de los animales.

Entidades

- **AnimalEntity** -> representa los animales disponibles para adoptar. Contiene atributos como ID, nombre, edad, estado (en adopción o en casa de acogida), características, enfermedades, entre otros.
- **UserEntity** -> los usuarios pueden guardar animales favoritos o enviar solicitudes de apadrinamiento/acogida, por lo que esta entidad representa la información del usuario. Contiene atributos como ID de usuario, nombre, correo electrónico, contraseña, lista de animales favoritos, animales en apadrinamiento, entre otros. Ésta es una entidad asociada perteneciente a otro módulo.

DAOs (Data Access Objects)

- **AnimalDao** -> contendría métodos para interactuar con la entidad 'AnimalEntity', como recuperar la lista de animales disponibles, buscar animales por características específicas, filtrar por estado de adopción, entre otros.
- **UserDao** -> permitiría operaciones como registrar usuarios, buscar usuarios por ID o correo electrónico, actualizar la lista de favoritos del usuario, actualizar la lista de animales en apadrinamiento, entre otros. Éste es un DAO asociado perteneciente a otro módulo.

Repositorios

- **'AnimalRepository'** -> este repositorio actuaría como un intermediario entre la capa de presentación y los DAOs. Coordinaría la recuperación de datos de animales desde 'AnimalDao' y se aseguraría de que los datos se presenten de manera efectiva en la interfaz de usuario. También podría realizar operaciones de filtrado y búsqueda en caso de ser necesario.
- **'UserRepository'** -> este repositorio coordinaría la lógica relacionada con los usuarios, como el registro y la gestión de listas (animales favoritos, apadrinados, etc.). Éste es un repositorio asociado perteneciente a otro módulo.

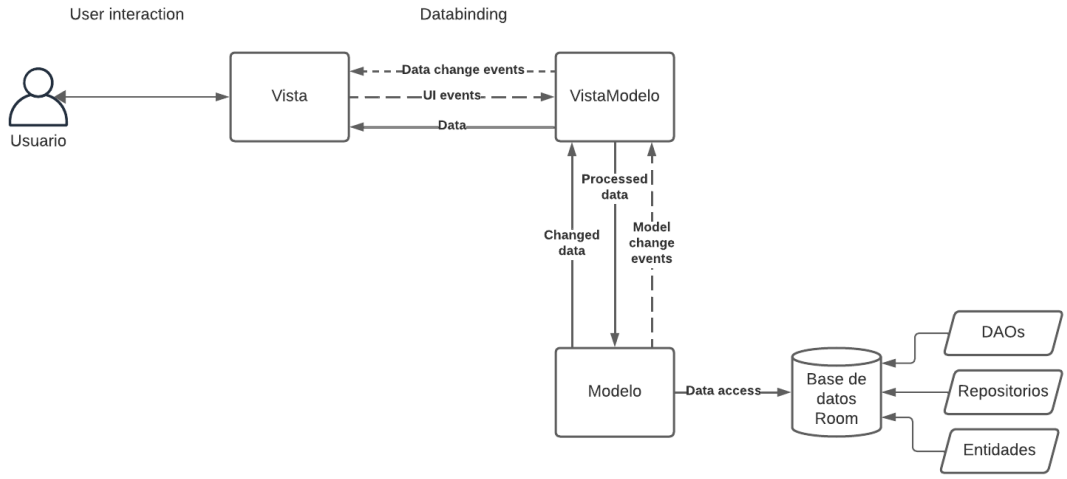
Relaciones

- Relación entre 'UserEntity' y 'AnimalEntity' para registrar los animales favoritos del usuario.
- Relación entre 'UserEntity' y 'AnimalEntity' para registrar los animales apadrinados/adoptados por el usuario.

Módulo asociado: Usuarios

Los usuarios pueden registrarse y guardar sus animales favoritos o solicitar el apadrinamiento o acogida de un animal. El módulo estaría diseñado para permitir que los usuarios realicen estas acciones tras iniciar sesión o crear una cuenta, pero la visualización de los detalles de los animales estaría disponible para cualquier usuario, ya sea que haya iniciado sesión o no.

Diagrama



4. Preparación para el futuro

¿Qué pasa si en un futuro se quisiera cambiar el motor de base de datos?

En caso de considerar un cambio en el motor de la base de datos, anticipamos una serie de desafíos que deberíamos afrontar. Dado que cada motor de base de datos tiene un modelo de datos único y requisitos específicos, sería necesario planificar y ejecutar la migración de los datos existentes. Dependiendo de la complejidad y el volumen de los datos, es probable que se requiera el desarrollo de scripts o herramientas personalizadas para transferir los datos del antiguo motor de base de datos al nuevo.

¿Qué partes de tu aplicación tendrías que modificar?

Las áreas de la aplicación que necesitarían modificaciones incluirían:

- **Capa de persistencia (Room):** toda la lógica de persistencia que utiliza Room tendría que ser reescrita o adaptada para el nuevo motor de base de datos.
- **Repositorios y DAOs:** los repositorios y DAOs diseñados específicamente para interactuar con Room requerirían ajustes para trabajar con el nuevo motor de base de datos.
- **Modelos de datos:** si los modelos de datos están específicamente diseñados para Room (anotaciones, relaciones, etc.), es posible que necesiten ajustes para adaptarse al nuevo motor.
- **Consultas y operaciones:** las consultas y operaciones de base de datos que están optimizadas para SQLite (el motor de Room) requerirían una revisión y modificación para funcionar con el nuevo motor.

¿Qué dificultades anticipas?

Anticipamos una serie de dificultades, incluyendo:

- **Sintaxis y características diferentes:** cada motor de base de datos tiene su propia sintaxis y conjunto de características únicas. La transición requerirá tiempo para familiarizarse con las nuevas formas de interactuar con la base de datos.

- **Migración de datos:** si la estructura de los datos o las relaciones cambian entre los motores de base de datos, puede ser un desafío migrar los datos existentes a la nueva base de datos.
- **Optimización de consultas:** las consultas que eran eficientes en el antiguo motor pueden no serlo en el nuevo. Se necesitará tiempo para optimizar y ajustar las consultas para el nuevo motor.
- **Pruebas y validación:** la transición requerirá pruebas exhaustivas para asegurarse de que todas las operaciones de base de datos funcionen correctamente con el nuevo motor.

¿Cómo podrías diseñar tu aplicación para minimizar el impacto de tal cambio?

Para minimizar el impacto de un cambio en el motor de base de datos, hemos aplicado buenas prácticas de diseño y considerado lo siguiente:

- **Abstracción de la capa de persistencia:** utilizaremos patrones de diseño como Repositorios y DAOs para abstraer la lógica de persistencia. Esto facilita la sustitución del motor de la base de datos sin afectar otras partes de la aplicación.
- **Utilización de interfaces y abstracciones:** diseñaremos el código para depender de interfaces en lugar de implementaciones concretas, lo que facilita la sustitución de componentes, como el motor de la base de datos.
- **Pruebas automatizadas:** implementaremos pruebas unitarias y de integración exhaustivas que cubren la capa de persistencia. Esto nos permite validar rápidamente que los cambios en el motor de la base de datos no rompan funcionalidades existentes.
- **Documentación detallada:** mantendremos una documentación clara y detallada sobre cómo se interactúa con la capa de persistencia, facilitando así futuras migraciones y cambios en la base de datos.
- **Seguimiento de buenas prácticas de diseño de bases de datos:** nos aseguraremos de seguir buenas prácticas al diseñar la base de datos, independientemente del motor que estemos utilizando, facilitando así futuras transiciones.