Solución Segundo punto

En este documento se adjunta la solución para el segundo punto de la prueba técnica de Habi.co. Encontrará una breve descripción de la propuesta y la razón de su diseño en diferentes niveles (Base de datos y arquitectura).

Imágenes

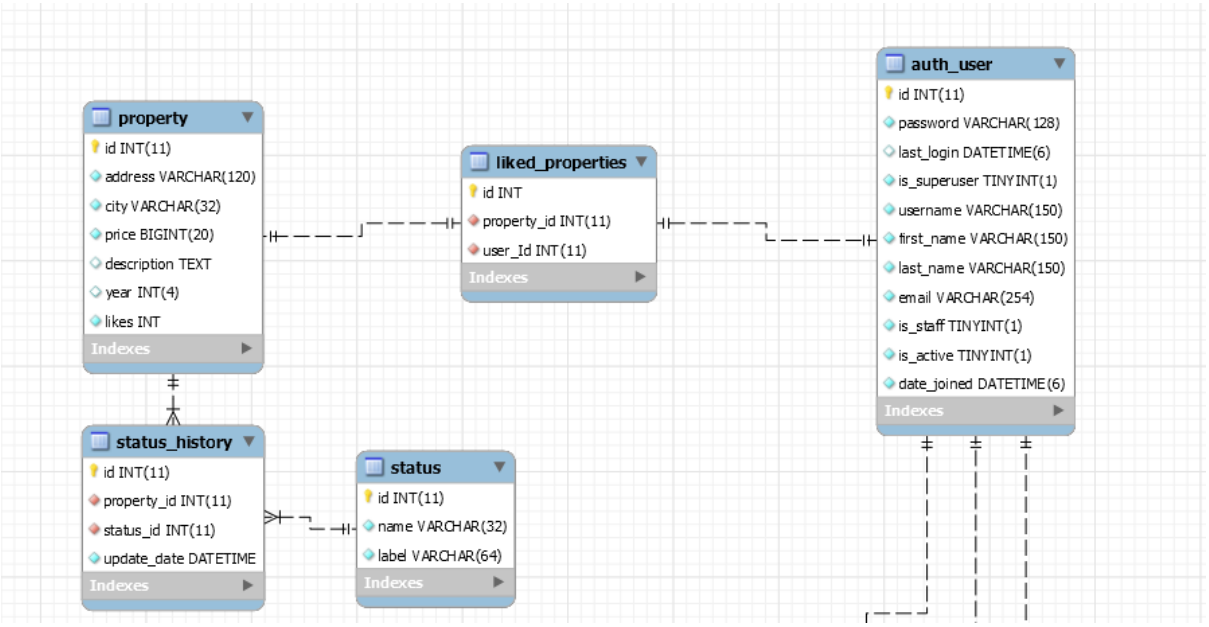

Imagen 1 - Modelo Base de datos

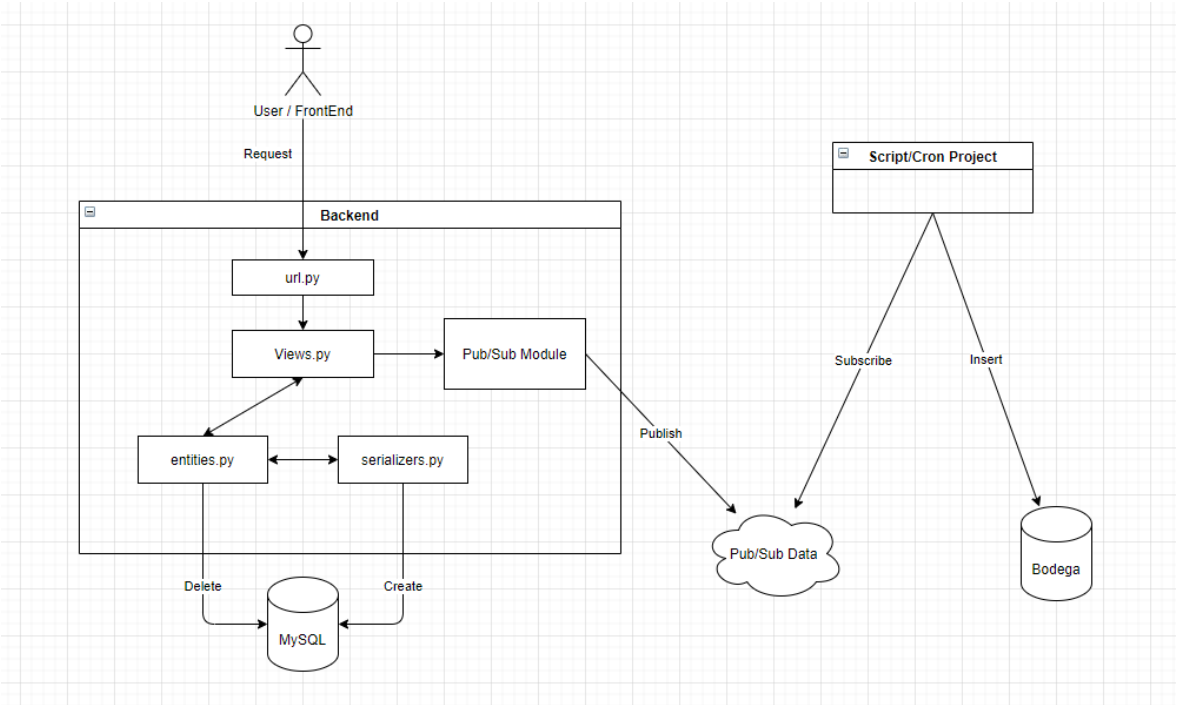

Imagen 2 - Modelo arquitectura de software

Modelo BD

Se propone generar una tabla intermedia llamada likes (Opciones de otros nombres como like_properties, si el modelo fuera mucho mayor, pues el usuario puede darle like a otro tipo de ítems), en el cual se representa qué propiedad le dio like de qué usuario. El modelo puede verse en la imagen 1.

Adicional se agrega una propiedad en la tabla properties (likes) que representa la cantidad de likes que cada inmueble tiene. Este número no debe ser menor a 0 en ningun escenario, y esto se tiene en cuenta en las consultas SQL.

En esta tabla se obvia la fecha en la cual el usuario le dio like a que propiedad por varias razones. La **primera** es que el ordenamiento de los resultados por usuario se hace por el ID (Si el usuario desea ver las propiedades que le gustan. Estas pueden ordenarse por el ID que es autoincremental, las últimas propiedades saldrán de últimas o de primeras si se hace desc). La **segunda** es que esta tabla es netamente transaccional, y el histórico de acciones (Like or Unlike) no debe verse reflejado en esta tabla; estos datos se deben llevar directamente a una bodega de datos (encargada de llevar los históricos) mediante un modelo de publisher/subscriber. Todo esto se explica en el modelo propuesto para la arquitectura.

NOTA: La consulta del primer punto puede verse alterada, pues ahora debe traer una columna adicional, y si se desea ver en la lista que propiedades tienen el like del usuario, debe hacerse un JOIN adicional con la tabla de likes. Esto lo define negocio, pues el like puede verse en la lista o en el detalle de cada propiedad

Aunque las consultas están en el código, se adjuntan aquí las siguientes acciones:

- Revisar si un usuario tiene o no like en la propiedad:

```
query = f"""
SELECT * FROM likes
WHERE user_id = {user.pk}
AND property_id = {property_id}
"""
```

- En caso de darle like a una nueva propiedad, debe insertar el like:

```
query = f"""
INSERT INTO likes (user_id, property_id)
values ({user.pk}, {property_id})
"""
```

- Incrementar la cantidad de likes de una propiedad

```
query = f"""
UPDATE property
SET likes = likes + 1
WHERE id = {property_id}
```

```
                    """
```

- En caso de remover un like a una propiedad:

```
                    query = f"""
                    DELETE from likes WHERE
                    WHERE property_id = {property_id}
                    AND user_id = {user.pk}
                    """
```

- Reducir la cantidad de likes de una propiedad

```
                    query = f"""
                    UPDATE property
                    SET likes = likes - 1
                    WHERE id = {property_id}
                    AND likes > 0
                    """
```

Modelo Arquitectura

Como se puede ver en la imagen 2, se propone una arquitectura a nivel de software, con la cual se quiere mantener una simplicidad en el modelo de base de datos propuesto y lograr capturar los datos de históricos por usuario (al menos con respecto a like/unlike). Esta propuesta se puede replicar en todo el aplicativo.

La idea es que una vez la interacción sea guardada (Ya sea que el usuario deje de gustarle una propiedad **UNLIKE,** o le de un nuevo like a una propiedad **LIKE),** esta acción salte hasta el archivo views.py, y este pueda invocar un módulo intermediario que guarde de manera asincrónica un mensaje en un sistema de PUB/SUB (publish/subscribe).

Este mensaje queda almacenado hasta que un suscriptor pueda leerlo, en este caso, un segundo proyecto de Scripts/Crons. Aquí se capturan todos los mensajes y son enviados hacia el histórico de la base de datos. La estructura del mensaje debe contener lo siguiente:

- Datos de usuario (Como mínimo el user_id)
- Datos de propiedad (Como mínimo el property_id)
- Tipo de acción (Select - Remove)
- Fecha de acción (Fecha en la que se ejecutó la acción, ósea fecha de publicación del mensaje).

De esta manera se garantiza la simplicidad en la base de datos transaccional (Solo se guardan las propiedades que el usuario le dio like) así como el histórico de acciones por usuario para fines de análisis.