Práctica 2

Iván Gijón
Alejandro Ramos
Víctor Mojica
Gonzalo Alganza

Abril 2024

UNIVERSIDAD
DE GRANADA

Índice

1 Mantenimiento

1.1 Mantenimiento Perfectivo

Antes de comenzar la implementación, decidimos modificar el UML del ejercicio 3, ya que usábamos 3 veces el patrón Builder, lo que lo hacía muy repetitivo y pesado de implementar. Ahora, las clases Banio y Cocina, no se van a crear usando el patrón Builder, si no que vamos a usar el patrón Decorador, por lo que el usuario puede añadir aparte de los elementos predeterminados un bidet y un jacuzzi al baño y una isla y un lavavajillas a la cocina. Tomando el caso del baño por ejemplo, se podrán añadir un bidet, un jacuzzi, o un bidet y un jacuzzi, pero nunca se podrán añadir más de dos elementos del mismo tipo, como añadir dos bidet o dos jacuzzi.

1.2 Mantenimiento Correctivo

No hemos tenido que realizar mantenimiento correctivo, ya que no había ningún problema a la hora de ejecutar el programa y todas las funciones cumplían con lo que se les pedía.

1.3 Mantenimiento Preventivo

No hemos tenido que realizar mantenimiento preventivo debido a que tomamos todas las medidas necesarias en la práctica 1 para evitar posibles fallas.

1.4 Mantenimiento Adaptativo

Para el mantenimiento adaptativo hemos tenido mucho trabajo, ya que además de modificar el código para aplicar los cambios en el modelo, tenemos que adaptar el código de Python a Dart, que es el lenguaje de programación que usa Flutter.

1.4.1 Adaptacion del codigo a Dart

En primer lugar para adaptar el código de la anterior práctica en Python a Dart tenemos que tener en cuenta que cambia la sintaxis y las estructuras utilizadas. En Dart puedes programar usando tipos o no, pero nosotros hemos optado por usarlos para ser más restrictivos y evitar posibles errores

futuros. En el siguiente ejemplo se ve la implementación de una clase abstracta en Python y como se haría en Dart.

Listing 1: Código en Python

```
from abc import ABC, abstractmethod
from src.Casa.Casa import Casa

class CasaBuilder(ABC):

    def __init__(self):
        self.casa = None

    def crear_casa(self):
        self.casa = Casa()

    def aniadir_cocina(self,cocina):
        self.casa.cocina = cocina

    def aniadir_banio(self,banio):
        self.casa.banio = banio

    @abstractmethod
    def aniadir_sala_de_estar(self):
        pass

    @abstractmethod
    def aniadir_dormitorio(self):
        pass
```

Listing 2: Código en Dart

```
import 'package:p2/modelo/Dormitorio.dart';

import 'Banio.dart';
import 'Casa.dart';
import 'Cocina.dart';
import 'SalaDeEstar.dart';

abstract class CasaBuilder{
  late Casa casa;

  void crearCasa(){
    casa = Casa.vacia();
  }

  void setCocina(Cocina c){}

  void setBanio(Banio b){}

  void setSalaDeEstar(SalaDeEstar s){}

  void addDormitorio(Dormitorio d){}

}
```

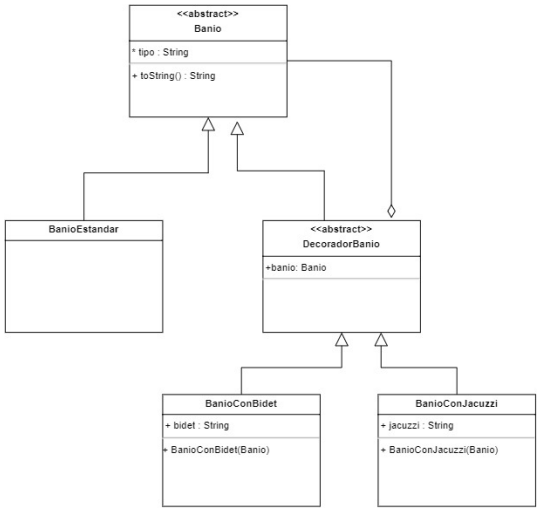
Como podemos ver los cambios mas significativos son el uso de tipos para las variables, la sintaxis para declarar los métodos de la clase (def setBanio(self,banio) / void setBanio (Banio b)), la manera de importar clases y el uso de paréntesis para delimitar métodos y clases.

Una vez conocidos los cambios que hay entre lenguajes de programación, hemos procedido a la adaptación a Dart del patrón builder que teníamos en la practica anterior para la creación de casas. Podremos tener un Chalet, una Casa de Campo o un Apartamento. El funcionamiento es el mismo que en la anterior práctica, solo va a cambiar el lenguaje con el que se ha programado.

Más adelante se podrá ver un ejemplo en funcionamiento de esta adaptacion.

1.4.2 Adaptación del código al nuevo UML

Como ya se ha comentado se va a incorporar el patrón decorador, que va permitir poder decorar espacios de la casa como son el Baño y la Cocina. Al baño se le va a poder añadir un bidet o un jacuzzi o los dos. A la cocina se le va a poder añadir una isla o un lavavajillas.
El patrón Decorador del Baño estará compuesto por la clase Abstracta Banio de la que heredan BanioEstandar (que es un baño normal) y DecoradorBanio que es otra clase abstracta que usaremos para crear los decoradores.
De DecoradorBanio heredan BanioConBidet y BanioConJacuzzi que serán los decoradores del baño y añadirán dichos objetos al baño. Es decir añadirán nueva funcionalidad al objeto envolviendo la anterior funcionalidad y usando el mismo objeto original sin modificar el codigo ni su comportamiento.



El decorador de la cocina se implementara de la misma manera que el del Banio, de modo que se pueda decorar la cocina con una isla o un lavavajillas o ambas.

1.4.3 Funcionamiento del nuevo código Implementado tras el mantenimiento Adaptativo

A continuación se muestra un ejemplo de como usar los decoradores en el código y de su funcionamiento.

```
CocinaEstandar cocinaChalet = CocinaEstandar();
CocinaConIsla cocinaConIsla = CocinaConIsla(cocinaChalet);
CocinaConLavavajillas cocinaConLavavajillas = CocinaConLavavajillas(cocinaConIsla);


BanioEstandar banioChalet = BanioEstandar();
BanioConJacuzzi banioConJacuzzi = BanioConJacuzzi(banioChalet);
```

Y como resultado se obtiene lo siguiente:

```
C:/flutter/bin/cache/dart-sdk/bin/dart.exe --enable-assert
PRUEBA DECORADORES
Baño Con jacuzzi
Cocina Con isla Con lavavajillas
```

Podemos ver que hemos añadido a una CocinaEstandar una Isla y un lavavajillas envolviendo siempre al objeto original y sin modificar su comportamiento.

Para el Baño tan solo lo hemos decorado con un Jacuzzi.Ahora mostramos un ejemplo de como funciona el patrón builder y los decoradores.

```
print("PRUEBA BUILDER");


ChaletBuilder builderChalet = ChaletBuilder();
builderChalet.cocina = cocinaConLavavajillas;
builderChalet.banio = banioConJacuzzi;

List<Dormitorio> dormitorios = [];

Dormitorio dormitorio1 = Dormitorio("Dormitorio1");
Dormitorio dormitorio2 = Dormitorio("Dormitorio2");
Dormitorio dormitorio3 = Dormitorio("Dormitorio3");

dormitorios.add(dormitorio1);
dormitorios.add(dormitorio2);
dormitorios.add(dormitorio3);

builderChalet.dormitorios = dormitorios;

DirectorCasa directorChalet = DirectorCasa(builderChalet);
directorChalet.construirCasa();
print(directorChalet.builder.casa.toString());
```

Y como resultado se obtiene lo siguiente:

```
PRUEBA BUILDER
ESPECIFICACIONES CASA:
Cocina Con isla Con lavavajillas Chalet
Baño Con jacuzzi Chalet
Sala De Estar Chalet
[Dormitorio1, Dormitorio2, Dormitorio3]
```

En este código creamos un Chalet con ChaletBuilder y le añadimos la cocina y el baño que creamos anteriormente, los cuales están decorados.Por ultimo añadimos los dormitorios y la sala de estar y con el Director de la casa la construimos.

2 Actualización del diagrama UML

2.1 Diagrama Práctica 1

En la práctica anterior no se podía observar bien el diagrama UML, por lo que volvemos a adjuntarlo aquí junto con un enlace para tener una vista del diagrama que permite hacer zoom y verlo con más claridad.

Diagrama UML del antiguo modelo

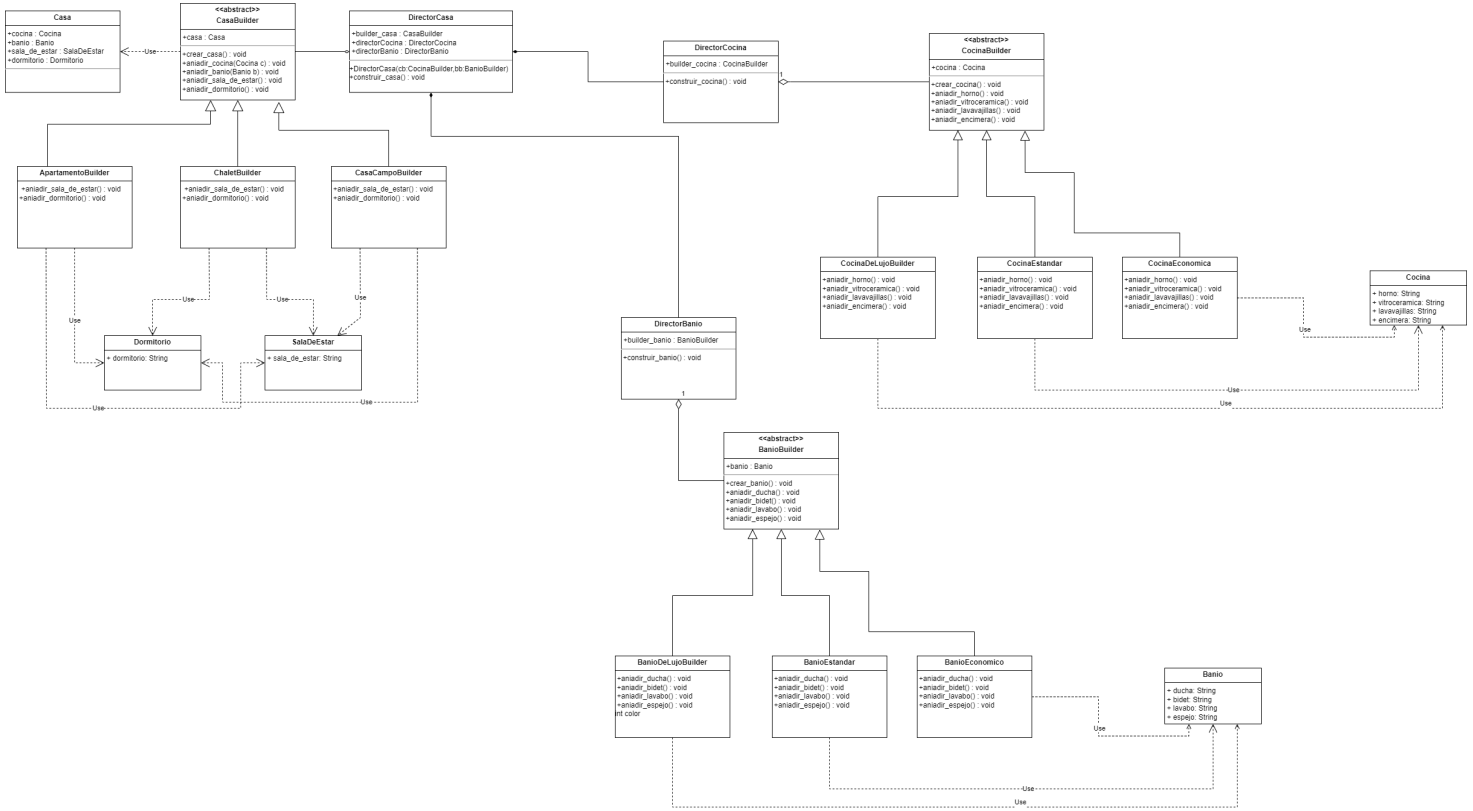


Figure 1: Imagen del diagrama de clases del tercer ejercicio en Python (Práctica 1)
Click aquí: Enlace a una vista completa que permite zoom

2.2 Diagrama Práctica 2

A continuación incluimos el diagrama UML que nos ha resultado para hacer la práctica 2. Al igual que la anterior, adjuntamos una imagen y un enlace para verlo mejor.
Los cambios más significativos que podemos ver entre UMLs son los siguientes:

- Eliminación de las clases 'BanioDirector' y "CocinaDirector". En su lugar, hemos creado las clases abstractas "Banio" y "Cocina" para representar la estructura de un baño y una cocina respectivamente.

- Implementación del Patrón Decorador. Hemos introducido las clases abstractas "DecoradorBanio" y "DecoradorCocina" como base de los decoradores y las clases "BanioEstandar" y "CocinaEstandar" que son los objetos sin modificaciones. De aquí salen sus respectivos decoradores para añadir al objeto base nuevos atributos sin necesidad de modificar la estructura base. Estas clases son: "BanioConJacuzzi", "BanioConBidet", "CocinaConLavavajillas" y "CocinaConIsla", que permiten agregar características adicionales a los baños y cocinas básicas.

Diagrama UML del modelo nuevo

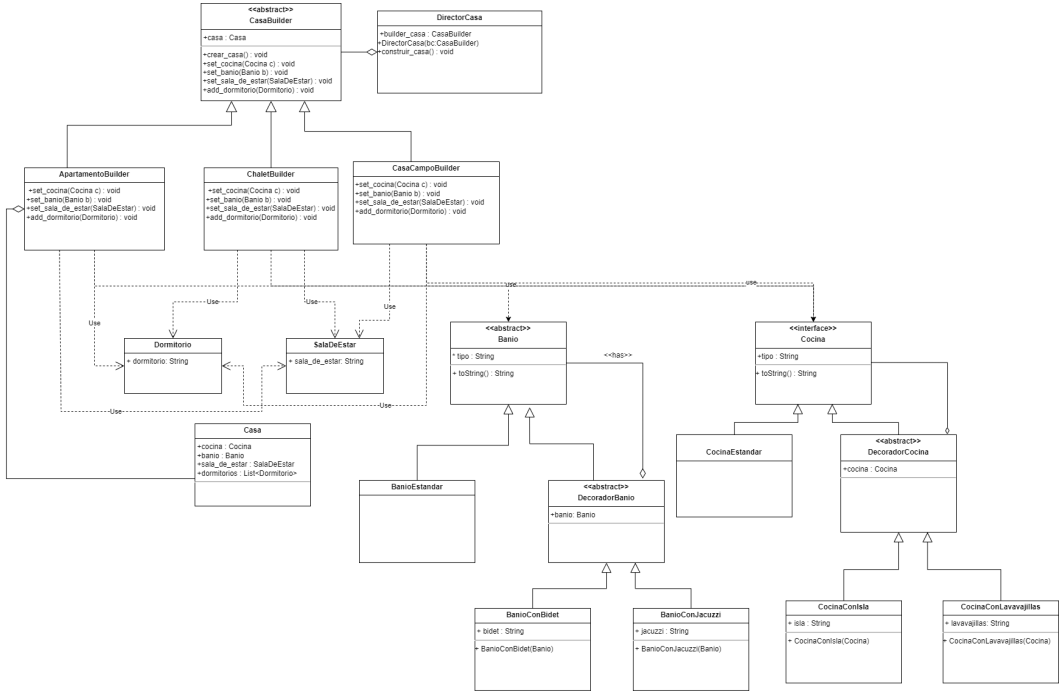


Figure 2: Imagen del diagrama de clases del tercer ejercicio en Python (Práctica 2)
Click aquí: Enlace a una vista completa que permite zoom

3 Prototipo de la Interfaz Gráfica de la aplicación

Antes de realizar el código, decidimos hacer un prototipo de como se vería la aplicación usando Figma. Este prototipo nos ayudo a imaginar como sería el funcionamiento de la aplicación y a ver que componentes nos harían falta en Flutter para elaborar algo parecido.

Al igual que los diagramas UML, adjuntamos una imagen general y un enlace que permite verlos mejor.

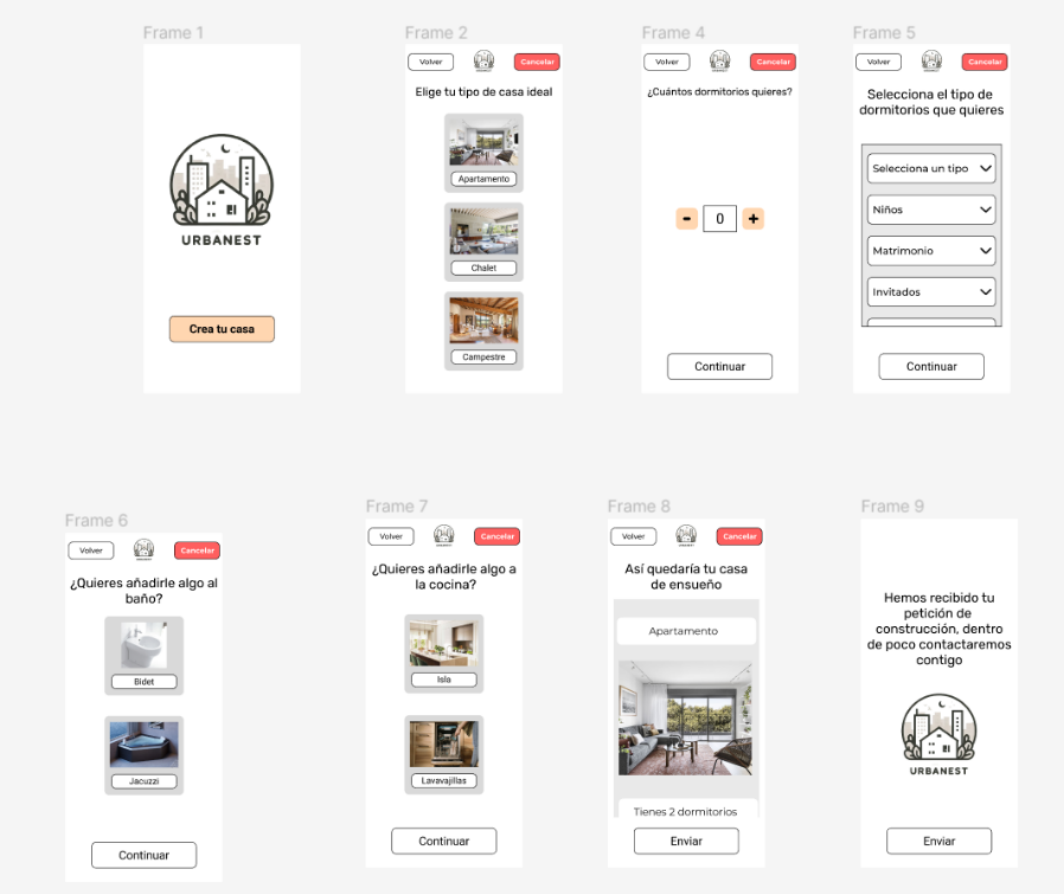


Figure 3: Imagen con los modelos prototipo de Figma
Click aquí: Enlace a una vista completa que permite zoom

4 Creacion de la Interfaz Gráfica y uso de los patrones

El proyecto está diseñado para móvil, por lo tanto, la visualización en un ordenador puede mostrar elementos colocados de manera incorrecta. Para solucionarlo, si se ejecuta la aplicación en ordenador, se debe poner la vista móvil para que se vea correctamente.
Para la personalización de la casa uitilizaremos el patrón builder, de manera que iremos pasando progresivamente el builder entre las siguientes interfaces para ir añadiendo los atributos que vayamos creando con nuestra aplicación (Cocina,Apartamentos,Baño...).

4.1 Landing Page

Una vez iniciado el proyecto, la primera página que aparece es la "LandingPage", que es una portada inicial, con el logo de la aplicación y un botón. Para esto, hemos hecho uso de los siguientes widgets:

- Image.asset: Se utiliza para mostrar el logo de la aplicaión en la pantalla. Cargando la imagen de los archivos del proyecto
- Inkwell: Crea un area rectangular interactiva que al pulsar, navega hasta la siguiente pantalla, "SeleccionTipo".

Figure 4: LandingPage del proyecto

4.2 Seleccionar Tipo de Casa

En esta pantalla se elige el tipo de casa que queremos crear entre las opciones disponibles. Para esta página, hemos usado los siguientes widgets:

- Header: es un componente común para todas las pantallas, que hemos creado nosotros formado por dos botones y el logo de la aplicación.
- Luego tenemos la selección del tipo de casa que queremos, dependiendo del botón que pulse, va a crear un Builder distinto y avanzará a la siguiente pantalla donde se eligen los dormitorios.

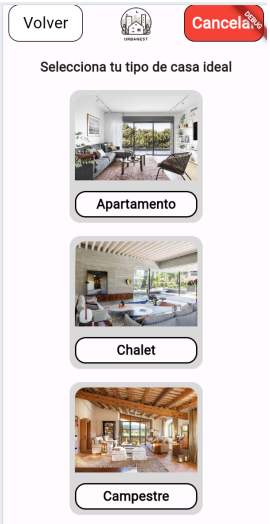


Figure 5: Seleccion Tipo de Casa

4.3 Selección de Dormitorios

La siguiente pantalla permite al usuario seleccionar el número de dormitorios que quiere para su casa (hasta un máximo de 5). Consta de los siguientes elementos:

- Un botón personalizado (botón que muestra el número de habitaciones seleccionadas y permite incrementar o decrementar este número). Contiene dos IconButton, uno para incrementar el número de dormitorios y otro para decrementarlo, y un TextField que muestra el número de dormitorios.

- Y otro botón para navegar a la siguiente pantalla que es un ElevatedButton que al pulsarlo avanza a la siguiente pantalla donde se selecciona el tipo de dormitorios que queremos.

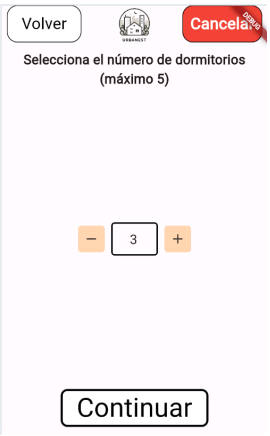


Figure 6: Seleccion Número de Dormitorios

4.4 Seleccion Tipo de Dormitorios

Aquí se elige de qué tipo queremos cada dormitorio que hemos elegido. Para elegir los dormitorios, hemos hecho uso de un ListView.builder, donde cada elemento es un DropdownMenu para elegir el tipo de dormitorio que queremos. Luego, se muestra el mismo botón para avanzar de pantalla que en la anterior.

4.5 Selección del Baño

En esta pantalla se muestran otros dos CustomButton, uno para el jacuzzi y otro para el bidet, la cabecera y el botón de continuar. Aquí se implementa la lógica del Patrón Decorador. Dependiendo de que botones seleccione el usuario, aplicará o no el decorador al baño estándar.

4.6 Selección de la Cocina

Se muestran los mismos botones que en la pantalla anterior, lo único que se modifica es la creación de una cocina en vez de un baño, usando la misma

Figure 7: Selección del Tipo de Dormitorios

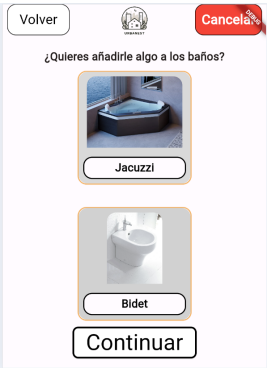


Figure 8: Selección del Baño

lógica. A la cocina estándar le aplicamos los decoradores que elija el usuario.

4.7 Pantalla del Resultado

Esta es la pantalla donde mostramos el resultado final de la casa con los atributos seleccionados. Cabe recalcar que en esta pantalla construimos la Casa a través del Director de la Casa de patrón Builder.

Hemos usado un ListView para mostrar toda la información final de la casa. Para saber qué tipo de casa se ha construido, accedemos a la casa

Figure 9: Selección de la Cocina

construida, y dependiendo de los valores que tenga, mostraremos al usuario la configuración de su casa. Nos fijaremos en cuántos dormitorios hay, si la cocina tiene isla o lavavajillas, que tipo de casa se esta configurando (Chalet,Apartamento o Casa de Campo), si se ha añadido al baño un jacuzzi o un bidet y el tipo de sala de estar que tiene la casa.

Esto permitirá al usuario tener una rápida visualización de la personalización de su casa que ha hecho mediante el uso de la aplicación.

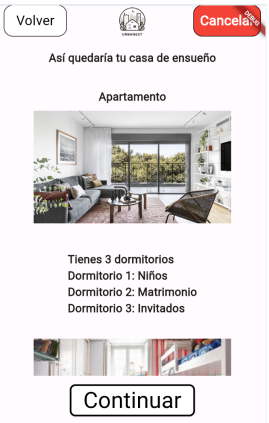


Figure 10: Pantalla Resultado