

Desafío Testing

TuCasita Tasaciones

Autor: Karen Martínez

Agosto 2021

Índice

1. Glosario	3
2. Introducción	4
3. Toma de decisiones	4
División de Repositorios	4
División de Servicios	4
División de Controladores	4
Manejo de Excepciones	5
Manejo de Tests Unitarios/Integración	5
Validaciones extras	6
4. Especificación de endpoints	7
PayLoads utilizados como entrada	7
Endpoints	8
US0001	8
US0002	8
US0003	9
US0004	9
US0005	10
US0006	10
*US0007	11
*US0008	11
*US0009	11
*US0010	12
5. Datos de prueba	12

1. Glosario

District: Sinónimo de la entidad Barrio.

Environment: Sinónimo de la entidad Habitación/Ambiente.

Property: Sinónimo de la entidad Propiedad.

2. Introducción

A continuación se hará una breve descripción sobre el trabajo, la cual será información complementaria a la documentación donde se presenta el desafío. El objetivo no será hacer una ficha técnica exhaustiva; sino que mencione los puntos más relevantes.

Se detallarán temas como la toma de decisiones en el correr del proyecto, y funcionalidades o especificaciones no mencionadas en el documento original.

3. Toma de decisiones

En esta sección se enumerarán los diferentes desafíos por los cuales se tuvo que optar por una decisión; sea esta de diseño y/o funcional.

1. División de Repositorios

En el proyecto se optó por manejar un repositorio para los barrios; *DistrictRepository*. Se optó por dicho camino ya que los únicos datos que se deben almacenar son los de los barrios (recurso District). Los mismos se almacenan en un json en la carpeta *resources*. Se asume que los datos existentes en el json son correctos.

La identificación de los barrios se hace mediante su nombre; por lo cual existe un método en *DistrictRepository* llamado *findByName(String):District*. Por lo general, por más que sepamos que los nombres de los barrios sean el principal identificador (*primary key*), también se suelen manejar identificadores propios (no siempre visibles para el usuario final). Pero, debido a que no habría una utilización definida para ello en base a los requerimientos; se optó por no agregarlo.

Pese a que en la especificación de requerimientos no se solicita, el repositorio de barrios presenta la posibilidad de agregar y quitar barrios por nombre. Y, debido a que la entidad barrio no se relaciona con ninguna otra que se guarde en el sistema, el borrado del mismo no es lógico. En caso que en algún momento se requiera guardar Property, seguramente la entidad District pase a tener un identificador propio, y a la vez los métodos de agregar y eliminar deban ser modificados.

2. División de Servicios

Actualmente existen dos interfaces de servicio, *IPropertyService* e *IDistrictService*. Se ha querido separar las operaciones relacionadas a Property en *IPropertyService*, mientras que las relacionadas a District en *IDistrictService*.

3. División de Controladores

Para la implementación de la solución se hicieron dos controladores; *PropertyController* y *DistrictController*.

PropertyController: Maneja todas las operaciones relacionadas al recurso Property (Propiedad). Todas las requests que sean atrapadas por dicho controlador comienzan con *properties*.

DistrictController: Maneja todas las operaciones relacionadas al recurso District (Barrio). Todas las requests que sean atrapadas por dicho controlador comienzan con *districts*.

4. Manejo de Excepciones

A la hora de realizar el manejo de las excepciones, se pensó en primer lugar en desligar a los repositorios de manejarlos; siendo la capa de servicios la única responsable de instanciarlos. Salvo en algunos casos particulares, como las excepciones *HttpMessageNotReadableException* y *MethodArgumentNotValidException*.

Todas las excepciones previstas en la solución poseen su clase particular y mensaje (derivando todas de *TuCasitaApiException*, que extiende de *RunTimeException*). El responsable de atrapar dichas excepciones es la clase *MiddlewareException*, bajo la anotación de *@ControllerAdvice*, para que la misma atrape todas las excepciones que puedan surgir en nuestra API. Y, mediante la anotación *@ExceptionHandler* definimos los métodos correspondientes para cada tipo de excepción.

Las excepciones relacionadas a las validaciones se muestran como una lista, y, por otro lado, las excepciones de validaciones generales en la capa de servicio se guardan en otra lista.

El proceso de las excepciones es el siguiente:

- Si tenemos una request con un body con la anotación *@Valid* mal formado, se lanzará la excepción *HttpMessageNotReadableException*.
- Si llegamos a tener en un endpoint errores capturados por las validaciones de la anotación *@Valid*, las mismas se mostrarán como una lista, indicando todos los mensajes de error que hubiesen.
- Por otro lado, si todas las validaciones son correctas, mostrará una lista si algunas de las validaciones en la capa de servicio fallan.
- Si no pasa por los tres puntos anteriores, pasará a mostrar una excepción particular en caso que corresponda. Por ejemplo, en el endpoint de agregar un barrio es posible que llegue a pasar por estas cuatro etapas.









5. Manejo de Tests Unitarios/Integración

Para los tests unitarios y de integración se optó por considerar una clase de prueba por implementación. Y, para cada método, casos de prueba exitosos y fallidos.

Las validaciones hechas a partir de la anotación *@Valid* se testean en las pruebas de integración para cada endpoint implementado.

Una vez realizada la solución y sus pruebas asociadas, se llegó a realizar 81 tests (unitarios + integración), con un coverage de líneas del 97%, siendo el paquete de repository con el menor coverage con un 95%. Esto se debe a la dificultad para testear todos los caminos al leer el archivo de properties y json utilizados en la clase *DistrictRepositoryImp*.

El objetivo de las pruebas unitarias relacionadas a la carga del json, es que independientemente de los errores que puedan ocurrir por ello, el sistema va a seguir funcionando.

100% classes, 97% lines covered in package 'com.meli.tucasita'			
Element	Class, %	Method, %	Line, %
 controller	100% (2/2)	100% (12/12)	100% (17/17)
 dto	100% (9/9)	100% (49/49)	100% (49/49)
 exception	100% (7/7)	92% (12/13)	96% (28/29)
 model	100% (1/1)	100% (7/7)	100% (7/7)
 repository	100% (2/2)	100% (9/9)	95% (40/42)
 service	100% (2/2)	100% (20/20)	100% (83/83)
 util	100% (1/1)	100% (2/2)	100% (2/2)
 TucasitaApplication	100% (1/1)	0% (0/1)	33% (1/3)

6. Validaciones extras

Para esta solución, se agregaron las siguientes validaciones extras:

Entidad	Campo	Validación	Texto de validación
DistrictDTO	district_built_price	Mínimo de 0.0	"El precio mínimo permitido por metro cuadrado construido no puede ser menor a 0.0 U\$S."
DistrictDTO	district_unbuilt_price	Mínimo de 0.0	"El precio mínimo permitido por metro cuadrado no construido no puede ser menor a 0.0 U\$S."
PropertyDTO	prop_land_width	Mínimo de 0.1	"El mínimo ancho permitido por terreno es de 0.1 mts."
PropertyDTO	prop_land_length	Mínimo de 0.1	"El mínimo largo permitido por terreno es de 0.1 mts."
EnvironmentDTO	room_width	Mínimo de 0.1	"El mínimo ancho permitido por propiedad es de 0.1 mts."
EnvironmentDTO	room_length	Mínimo de 0.1	"El mínimo largo permitido por propiedad es de 0.1 mts."

			mts."
--	--	--	-------

Se tomó esta decisión debido a que no tiene sentido que se ingresen valores negativos en medidas ni precios. A las medidas se les pone un mínimo de 0.1 ya que si fuesen 0.0 sería equivalente a que la misma no existiera.

Por otro lado, para diferenciar el mensaje de error en el campo `district_unbuilt_price` en relación a `district_built_price`, se sustituye en el mensaje de error del primero “construido” por “no construido”.

4. Especificación de endpoints

A continuación se detallarán todos los endpoints. Indicando las consideraciones pertinentes a tomar en cuenta en cada uno. Cabe destacar que algunas de las condiciones ya han sido mencionadas en la sección anterior; pero se opta por mencionarlas de todas formas por practicidad para el lector. Los requerimientos que terminen con * son aquellos que no fueron solicitados.

- Todas las excepciones que requieran de un body, si su json está mal formado se devolverá el error en formato json `{code:{code} , message:{message}}` con el status *BAD_REQUEST*.
- Las excepciones inesperadas en el sistema devolverán el error en formato json `{code:{code} , message:{message}}` un status *INTERNAL_SERVER_ERROR*.

PayLoads utilizados como entrada

PropertyDTO
<pre>{ "prop_name": "Propiedad", "prop_land_width": 30, "prop_land_length": 24, "district_name": "Jorobas", "rooms": [{ "room_name": "Recamara", "room_width": 4, "room_length": 5 }, { "room_name": "Baño", "room_width": 2, "room_length": 2 }, { "room_name": "Sala", "room_width": 3, "room_length": 4 }] }</pre>

```
}
  ]
}
```

DistrictDTO

```
{
  "district_name":"Barrio",
  "district_unbuilt_price":70.0,
  "district_built_price":70.0
}
```

Endpoints

US0001

Method	Sign
POST	/properties/totalSquareFeet

Body

PropertyDTO

- Si las validaciones de *PropertyDTO* fallan, retornará la lista de errores en formato json *{code:{code} , message:{message}}* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, y las validaciones definidas en la capa de servicios fallan (excepciones *NoSuchDistrictNameException* y *EnviromentsBiggerThanPropertyException*), retornará la lista de errores en formato json *{code:{code} , message:{message}}* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO *PropertyResponseSquareFeetDTO*

US0002

Method	Sign
POST	/properties/totalSquareFeetUnbuilt

Body

PropertyDTO

- Si las validaciones de *PropertyDTO* fallan, retornará la lista de errores en formato json *{code:{code} , message:{message}}* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, y las validaciones definidas en la capa de servicios fallan (excepciones *NoSuchDistrictNameException* y *EnviromentsBiggerThanPropertyException*), retornará la lista de errores en formato json *{code:{code} , message:{message}}* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO *PropertyResponseSquareFeetUnbuiltDTO*

US0003

Method	Sign
POST	/properties/totalValue

Body

PropertyDTO

- Si las validaciones de *PropertyDTO* fallan, retornará la lista de errores en formato json *{code:{code} , message:{message}}* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, y las validaciones definidas en la capa de servicios fallan (excepciones *NoSuchDistrictNameException* y *EnviromentsBiggerThanPropertyException*), retornará la lista de errores en formato json *{code:{code} , message:{message}}* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO *PropertyResponseTotalValueDTO*.
- Se define como el valor total de una propiedad la suma del precio del terreno sin construir y construido en base al barrio.

US0004

Method	Sign
POST	/properties/biggestEnvironment

Body

PropertyDTO

- Si las validaciones de *PropertyDTO* fallan, retornará la lista de errores en formato json `{code:{code} , message:{message}}` y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, y las validaciones definidas en la capa de servicios fallan (excepciones *NoSuchDistrictNameException* y *EnviromentsBiggerThanPropertyException*), retornará la lista de errores en formato json `{code:{code} , message:{message}}` y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO *EnvironmentResponseSquareFeetDTO*

US0005

Method	Sign
POST	/properties/environments/totalSquareFeet

Body
PropertyDTO

- Si las validaciones de *PropertyDTO* fallan, retornará la lista de errores en formato json `{code:{code} , message:{message}}` y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, y las validaciones definidas en la capa de servicios fallan (excepciones *NoSuchDistrictNameException* y *EnviromentsBiggerThanPropertyException*), retornará la lista de errores en formato json `{code:{code} , message:{message}}` y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO *PropertyResponseEnvironmentsSquareFeetDTO*

US0006

Method	Sign
POST	/districts/isValid

Body
DistrictDTO

- Si las validaciones de *DistrictDTO* fallan, retornará la lista de errores en formato json `{code:{code} , message:{message}}` y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el mensaje “El distrito es válido.”

*US0007

Method	Sign
GET	/districts

- Retornará status *OK* y la lista de *DistrictDTO* en el sistema.

*US0008

Method	Sign
GET	/districts/{name}

- Si el nombre del barrio no existe, devolverá el error *NoSuchDistrictNameException* en formato json `{code:{code} , message:{message}}` y status *NOT_FOUND*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO de *DistrictDTO* del barrio encontrado.

*US0009

Method	Sign
POST	/districts

Body
DistrictDTO

- Si las validaciones de *DistrictDTO* fallan, retornará la lista de errores en formato json `{code:{code} , message:{message}}` y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, si el nombre del barrio ya existe, devolverá el error *DistrictWNameAlreadyExistsException* en formato json `{code:{code} , message:{message}}` y status *CONFLICT*.
- Si ocurre algún error al guardar, retornará la excepción *FailedToSaveDataException* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el DTO de *DistrictDTO* del barrio agregado.

*US0010

Method	Sign
DELETE	/districts/{name}

- Si el nombre del barrio no existe, devolverá el error *NoSuchDistrictNameException* en formato json *{code:{code} , message:{message}}* y status *NOT_FOUND*.
- Si ocurre algún error al guardar, retornará la excepción *FailedToSaveDataException* y status *BAD_REQUEST*.
- Si no pasa por el punto anterior, retornará status *OK* y el mensaje "OK".
- Este endpoint sólo borra la primera aparición del elemento.

5. Datos de prueba

En la ruta `"/src/{SCOPE}/resources/{fileName}"`, siendo definido SCOPE y fileName en el archivo de propiedades del respectivo ambiente. Si por alguna razón se necesita volver a cargar el json en la api, el método definido en `IDistrictRepository` llamado `reloadData()` sirve para actualizar el repositorio en base a los datos en el json.

Por otro lado, para las pruebas se definió una clase utilitaria llamada `TestUtilGenerator`, la misma posee entidades válidas e inválidas para poder ser utilizadas, al igual que la lista que debiera haber en el json de prueba. Si por alguna razón se desean cambiar los datos en el json, favor de actualizar dichos métodos.