DP1 2020-2021

Documento de Diseño del Sistema

Proyecto PetExtension

https://github.com/gii-is-DP1/dp1-2020-g2-08

Miembros :

- Miguel Durán González
- Manuel García Marchena
- Álvaro Gómez Pérez
- Pablo Mateos Angulo
- Isaac Muñiz Valverde
- Jorge Toledo Vega

Tutor: Irene Bedilia Estrada Torres

GRUPO G2-08

Versión 1

09/01/2021

Historial de versiones

Estos son ejemplo del contenido que debería tener el historial de cambios del documento a entregar a lo largo de los sprints del proyecto

Fecha	Versión	Descripción de los cambios	Sprint
13/12/2020	V1	● Creación del documento	2
13/12/2020	V2	● Añadido diagrama de dominio/diseño ● Explicación de la aplicación del patrón caché	3
10/01/2021	V2.1	● Añadidos los diagramas de dominio/diseño y de capas del refugio	3

Contents

Introducción

Al proyecto base de Petclinic, le vamos añadir una serie de funcionalidades, aportando valor al proyecto.
A la clínica le añadiremos una tienda de productos para mascotas, como comidas y accesorios donde comprarían los owners y pondrían valoraciones para que otros pudieran guiarse a la hora de comprar algo.
Queremos añadir un hotel para mascotas, en el que cada owner podría alojar a sus mascotas durante un período de tiempo reservando una habitación, teniendo también la opción de poner una valoración y opinión sobre la estancia, de forma que otros owners puedan ver como es el hotel.
Otra funcionalidad sería añadir un refugio en el que habría animales, en el que podríamos ver una serie de animales de cualquier tipo, dándonos la opción de adoptarlos para que pase a ser nuestra mascota.

Algunas de las funcionalidades más interesantes que hemos añadido al proyecto es, poder crear diferentes tipos de usuario dentro del sistema y que cada uno de estos tengan permisos exclusivos una vez inicien sesión, no es lo mismo un cliente que un owner; el cliente es referenciado a la tienda y el owner a los hoteles y a la clínica en sí. Otra cosa interesante es que los tres módulos que hemos implementado son totalmente independientes unos de otros favoreciendo así la cohesión del sistema y haciendo que si uno de los módulos falla, el resto sigue funcionando perfectamente.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

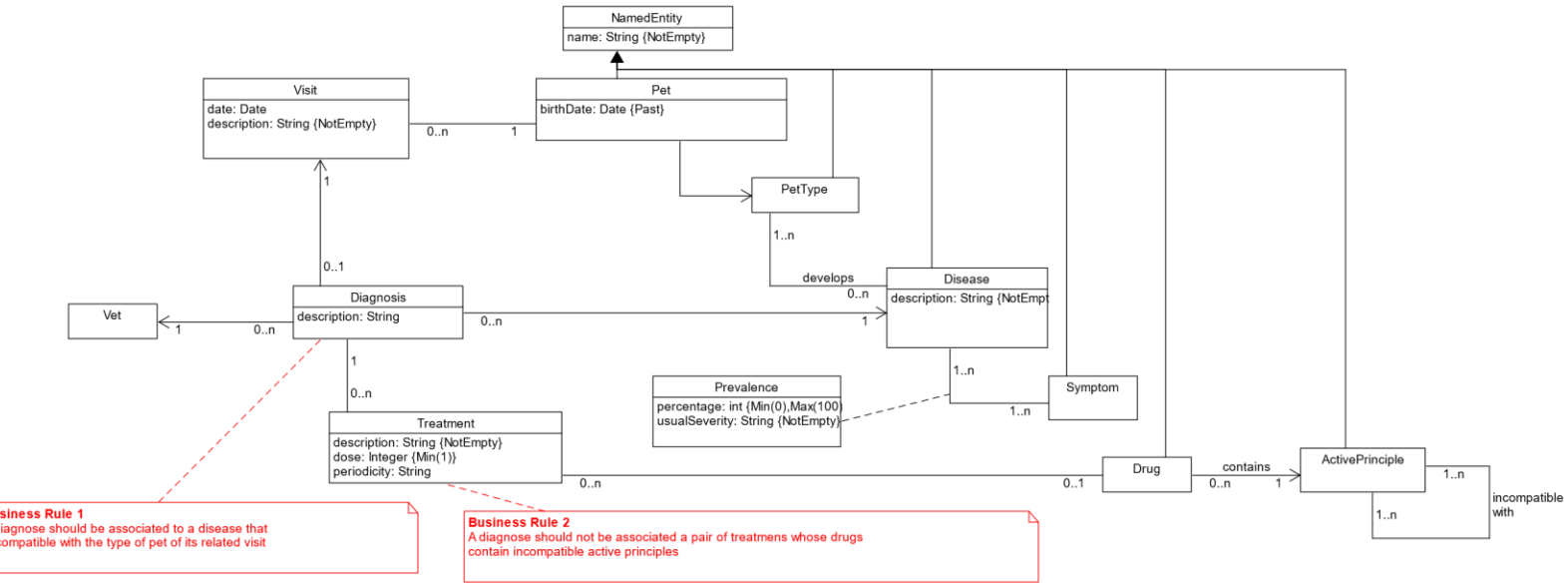
En esta sección debe proporcionar un diagrama UML de clases que describa el modelo de dominio, recuerda que debe estar basado en el diagrama conceptual del documento de análisis de requisitos del sistema pero que debe:

- *Especificar la direccionalidad de las relaciones (a no ser que sean bidireccionales)*
- *Especificar la cardinalidad de las relaciones*
- *Especificar el tipo de los atributos*
- *Especificar las restricciones simples aplicadas a cada atributo de cada clase de domino*
- *Incluir las clases específicas de la tecnología usada, como por ejemplo BaseEntity, NamedEntity, etc.*
- *Incluir los validadores específicos creados para las distintas clases de dominio (indicando en su caso una relación de uso con el estereotipo <<validates>>.*

Un ejemplo de diagrama para los ejercicios planteados en los boletines de laboratorio sería (hemos omitido las generalizaciones hacia BaseEntity para simplificar el diagrama):

DP1 2020/21
PetExtension

Grupo: G2-08



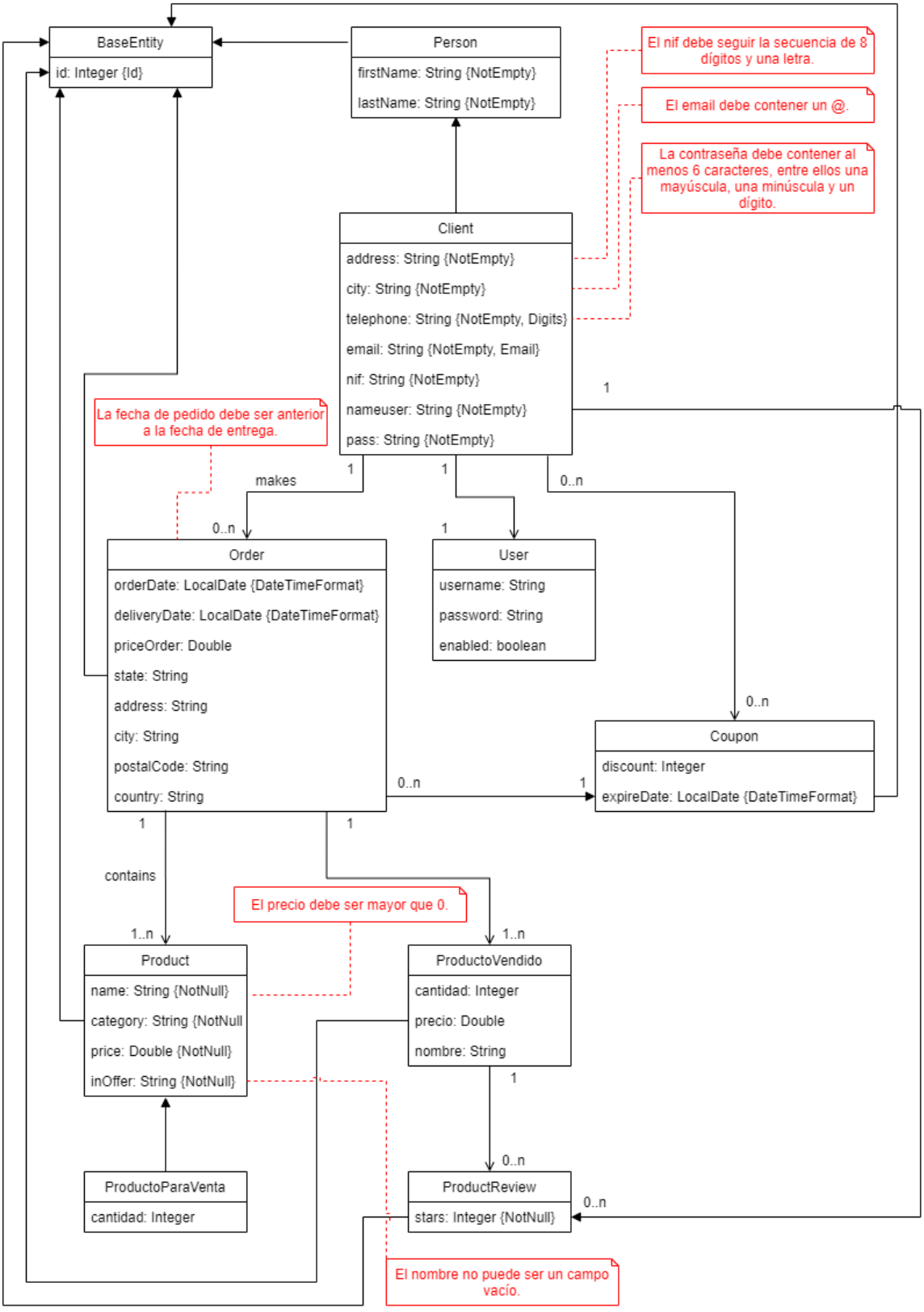


Figura 1. Diagrama de dominio/diseño tienda.

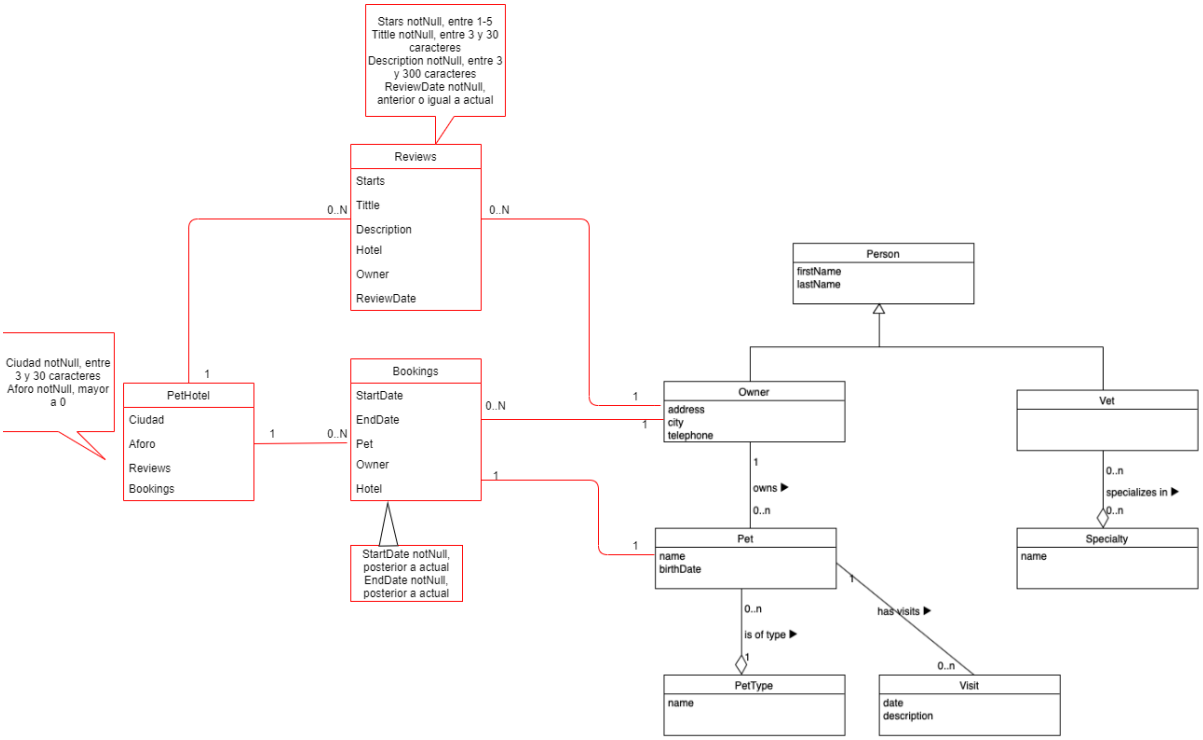


Figura 2. Diagrama de dominio/diseño hotel.

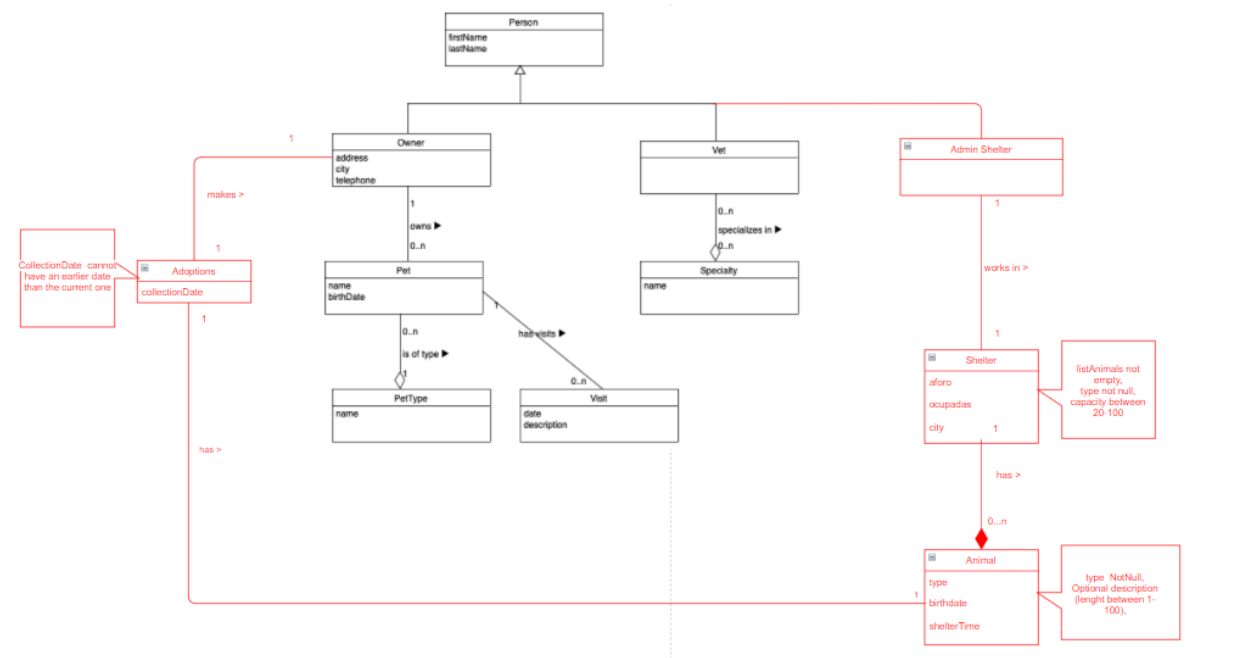


Figura 3. Diagrama de dominio/diseño refugio.

Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

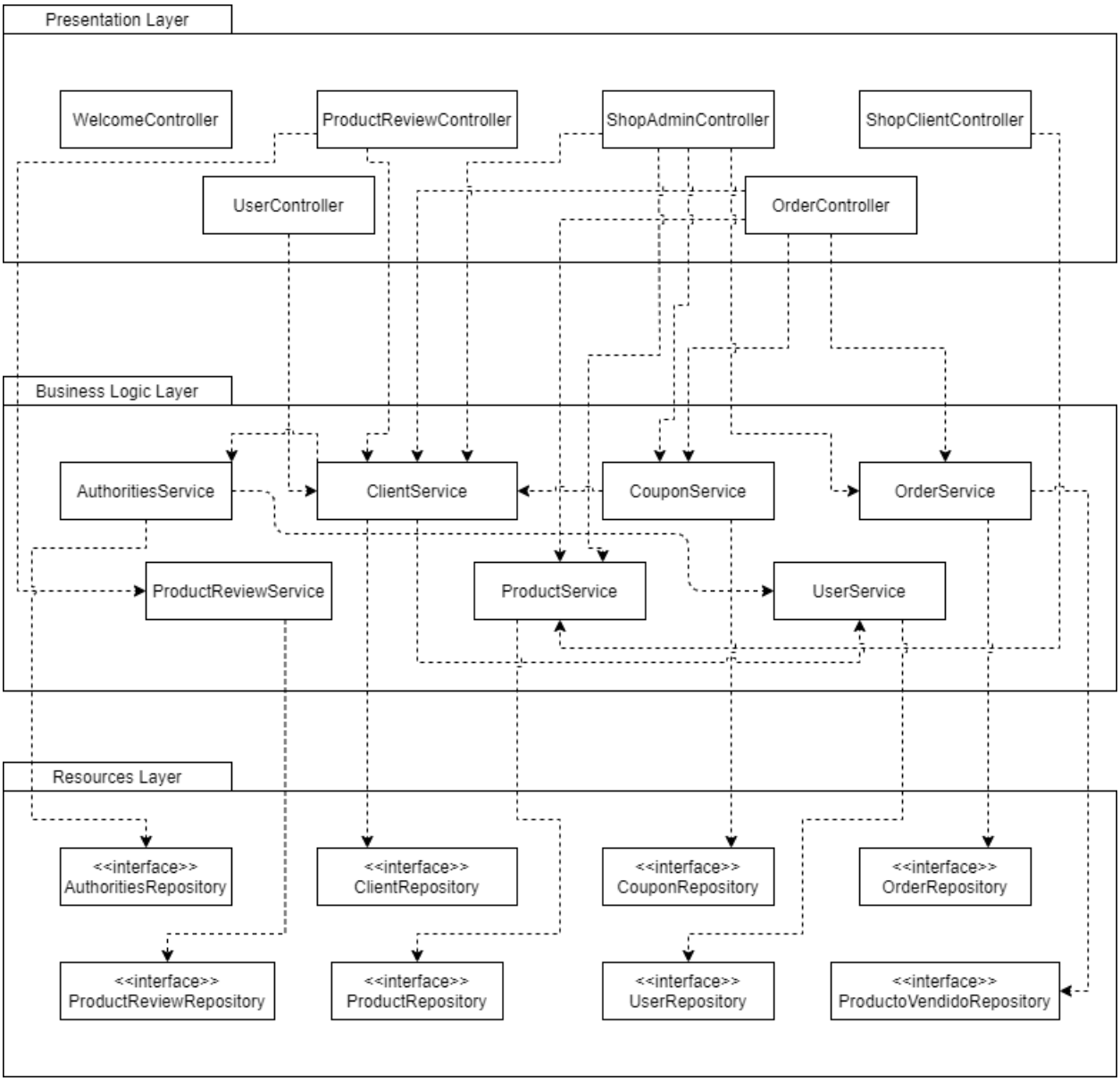


Figura 4. Diagrama de capas tienda.

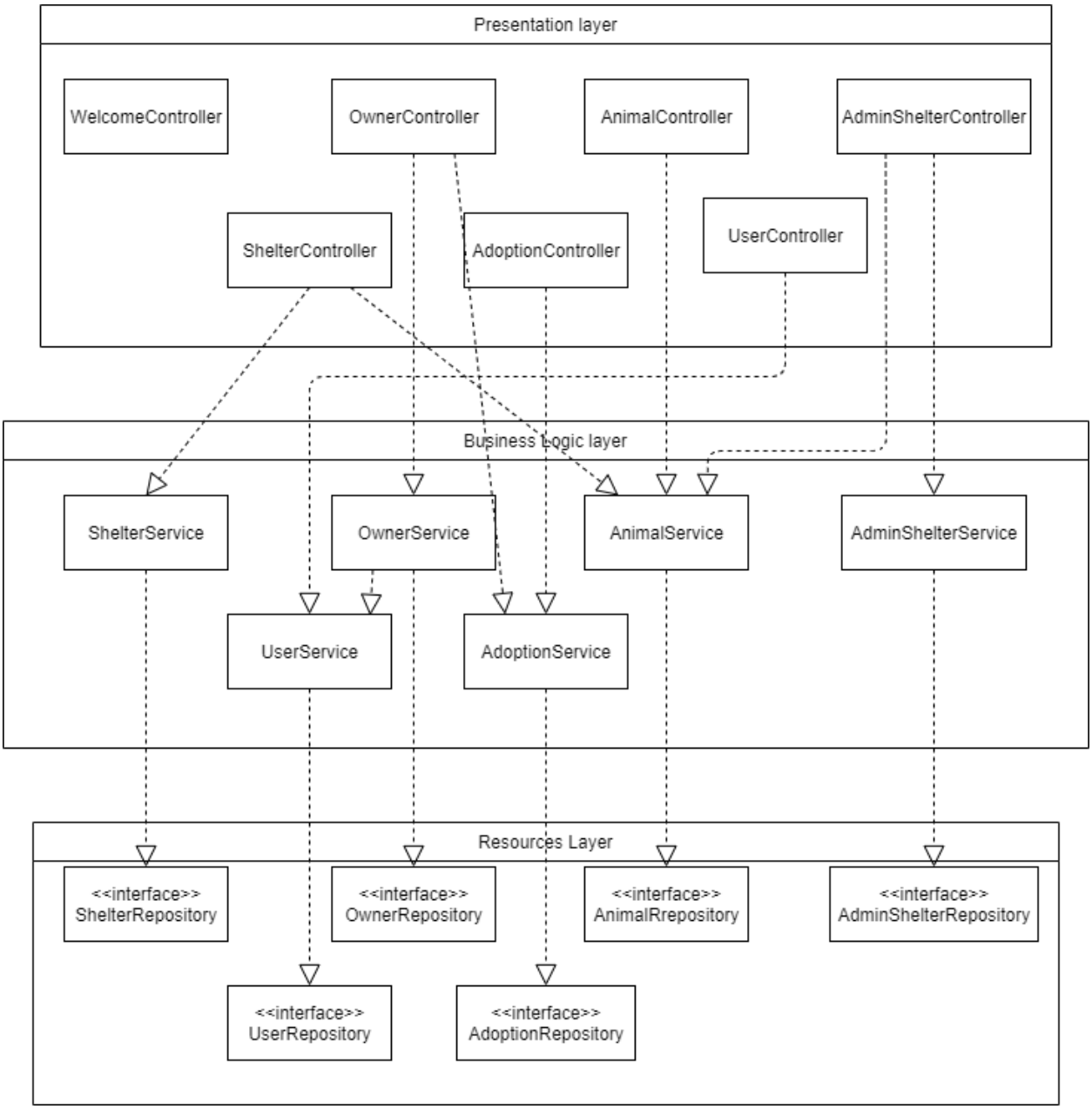


Figura 5. Diagrama de capas shelter.

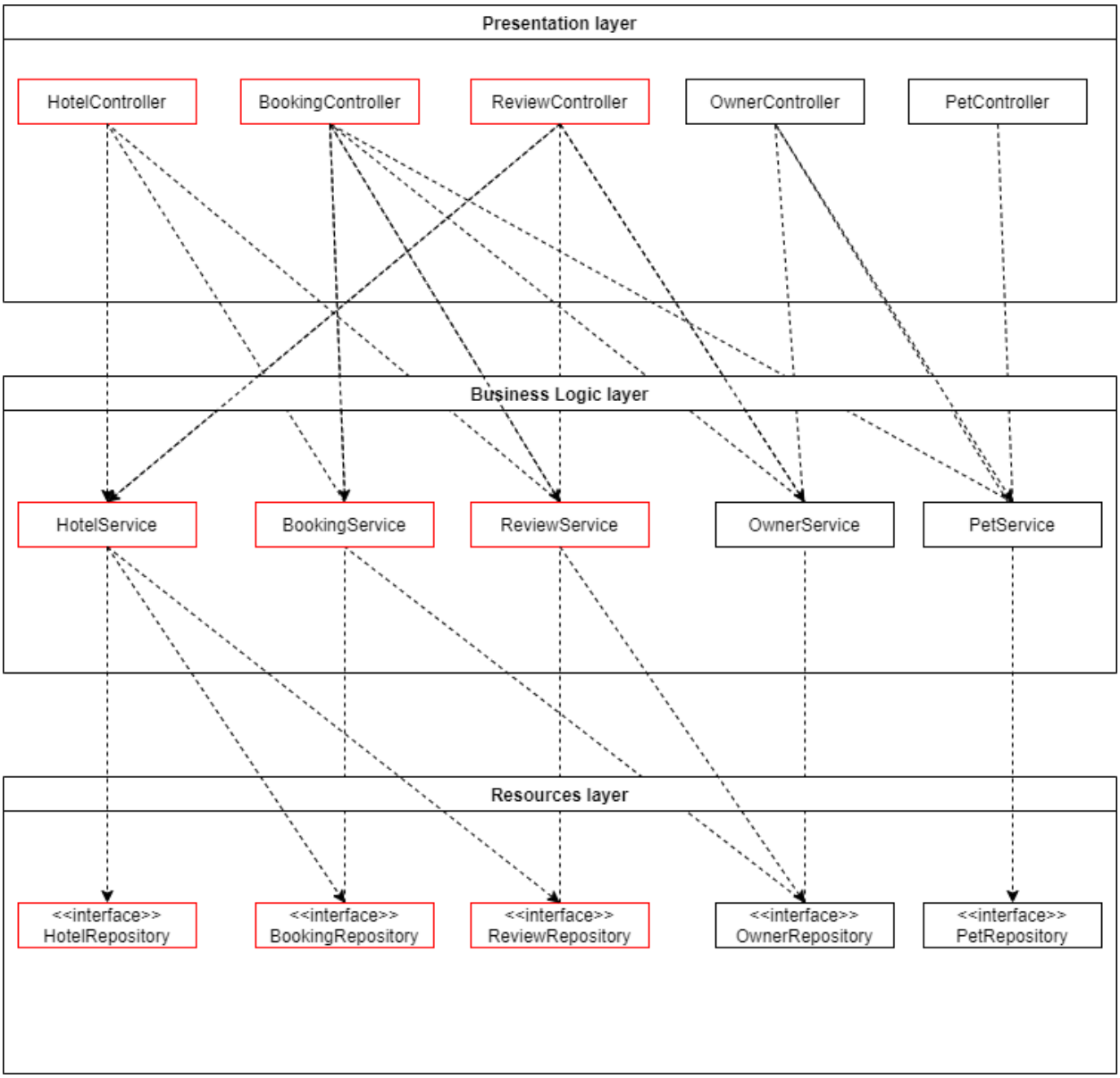


Figura 6. Diagrama de capas hotel.

Patrones de diseño y arquitectónicos aplicados

En esta sección se especifica el conjunto de patrones de diseño y arquitectónicos aplicados durante el proyecto. Para especificar la aplicación de cada patrón puede usar la siguiente plantilla:

Patrón: <MVC>

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

El patrón modelo vista controlador (MVC) se ha aplicado en todo el proyecto para dividir las vistas que verá el usuario en la interfaz gráfica, los controladores, que es donde los desarrolladores crean las

funcionalidades de la aplicación, y el modelo, que es donde se describen los tipos que se usarán en la aplicación(Nombre de atributos, restricciones y creación de tablas)

Clases o paquetes creados

Paquete models, para las entidades

Paquete Web, para los controladores

Paquete WEB-INF, para los archivos jsp(vistas)

Ventajas alcanzadas al aplicar el patrón

Este patrón tiene como ventaja separar la lógica de negocio y las interacciones del usuario, que facilita el manejo de errores. Otra gran ventaja es que se puede dividir fácilmente el trabajo entre los miembros del equipo de desarrolladores. Si necesitamos cambiar algo en la lógica de la aplicación, al estar separada de la interfaz de usuario, no habría que modificar ambas partes, ya que son independientes.

Decisiones de diseño

En esta sección describiremos las decisiones de diseño que se han tomado a lo largo del desarrollo de la aplicación que vayan más allá de la mera aplicación de patrones de diseño o arquitectónicos.

Decisión X

Descripción del problema:

Describir el problema de diseño que se detectó, o el porqué era necesario plantearse las posibilidades de diseño disponibles para implementar la funcionalidad asociada a esta decisión de diseño.

Alternativas de solución evaluadas:

Especificar las distintas alternativas que se evaluaron antes de seleccionar el diseño concreto implementado finalmente en el sistema. Si se considera oportuno se pueden incluir las ventajas e inconvenientes de cada alternativa

Justificación de la solución adoptada

Describir porqué se escogió la solución adoptada. Si se considera oportuno puede hacerse en función de qué ventajas/inconvenientes de cada una de las soluciones consideramos más importantes.

Ejemplo:

Decisión 1: Importación de datos reales para demostración

Descripción del problema:

Como grupo nos gustaría poder hacer pruebas con un conjunto de datos reales suficientes, porque resulta más motivador. El problema es que al incluir todos esos datos como parte del script de inicialización de la base de datos, el arranque del sistema para desarrollo y pruebas resulta muy tedioso.

Alternativas de solución evaluadas:

Alternativa 1.a: Incluir los datos en el propio script de inicialización de la BD (data.sql).

Ventajas:

- Simple, no requiere nada más que escribir el SQL que genera los datos.

Inconvenientes:

- Ralentiza todo el trabajo con el sistema para el desarrollo.
- Tenemos que buscar nosotros los datos reales

Alternativa 1.b: Crear un script con los datos adicionales a incluir (extra-data.sql) y un controlador que se encargue de leerlo y lanzar las consultas a petición cuando queramos tener más datos para mostrar.

Ventajas:

- Podemos reutilizar parte de los datos que ya tenemos especificados en (data.sql).
- No afecta al trabajo diario de desarrollo y pruebas de la aplicación

Inconvenientes:

- Puede suponer saltarnos hasta cierto punto la división en capas si no creamos un servicio de carga de datos.
- Tenemos que buscar nosotros los datos reales adicionales

Alternativa 1.c: Crear un controlador que llame a un servicio de importación de datos, que a su vez invoca a un cliente REST de la API de datos oficiales de XXXX para traerse los datos, procesarlos y poder grabarlos desde el servicio de importación.

Ventajas:

- No necesitamos inventarnos ni buscar nosotros los datos.
- Cumple 100% con la división en capas de la aplicación.
- No afecta al trabajo diario de desarrollo y pruebas de la aplicación

Inconvenientes:

- Supone mucho más trabajo.
- Añade cierta complejidad al proyecto

Justificación de la solución adoptada

Como consideramos que la división es capas es fundamental y no queremos renunciar a un trabajo ágil durante el desarrollo de la aplicación, seleccionamos la alternativa de diseño 1.c.