DP1 2020-2021

Documento de Diseño del Sistema

Proyecto Gestión de taxis rurales

< https://github.com/gii-is-DP1/dp1-2020-g2-3 >

Miembros <en orden alfabético por apellidos>:

- Cuevas Gallardo, José Manuel
- Díaz Correa, Vicente
- Hernández Rodríguez, Iván
- Macías Portillo, José Antonio
- Nold Cardona, Elena
- Rivas Llamas, Manuel

Tutor: José Antonio Parejo

GRUPO G2-3

Versión 3.0

10/01/2021

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
10/12/2020	V1	• Creación del documento • Añadido diagrama de dominio/diseño	3

Contents

Esta es una plantilla que sirve como guía para realizar este entregable. Por favor, mantén las mismas secciones y los contenidos que se indican para poder hacer su revisión más ágil.

Introducción

En esta sección debes describir de manera general cual es la funcionalidad del proyecto a rasgos generales (puedes copiar el contenido del documento de análisis del sistema). Además puedes indicar las funcionalidades del sistema (a nivel de módulos o historias de usuario) que consideras más interesantes desde el punto de vista del diseño realizado.

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>>.*

DP1 2020/21
<Nombre Proyecto> Grupo: GX-XX

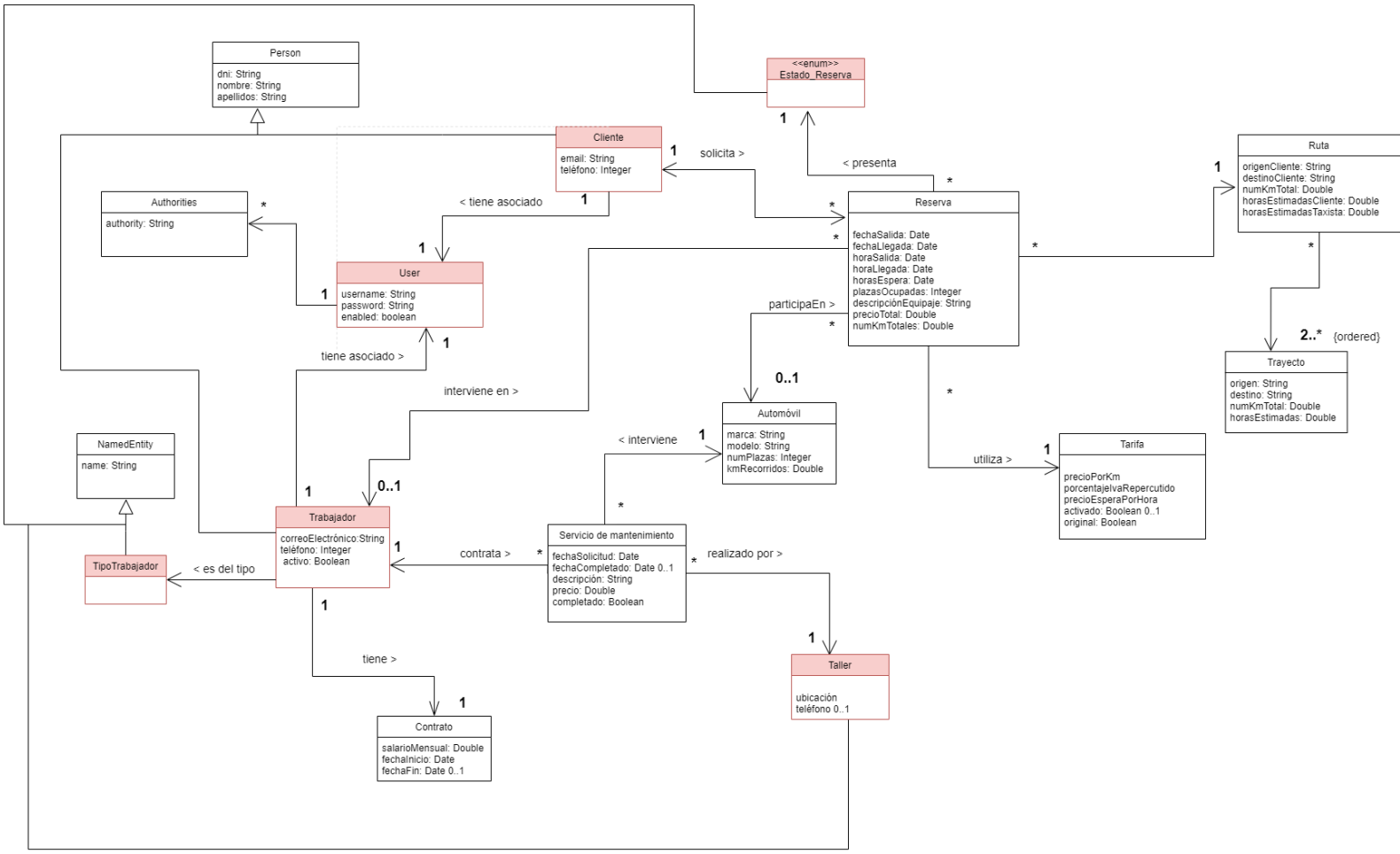
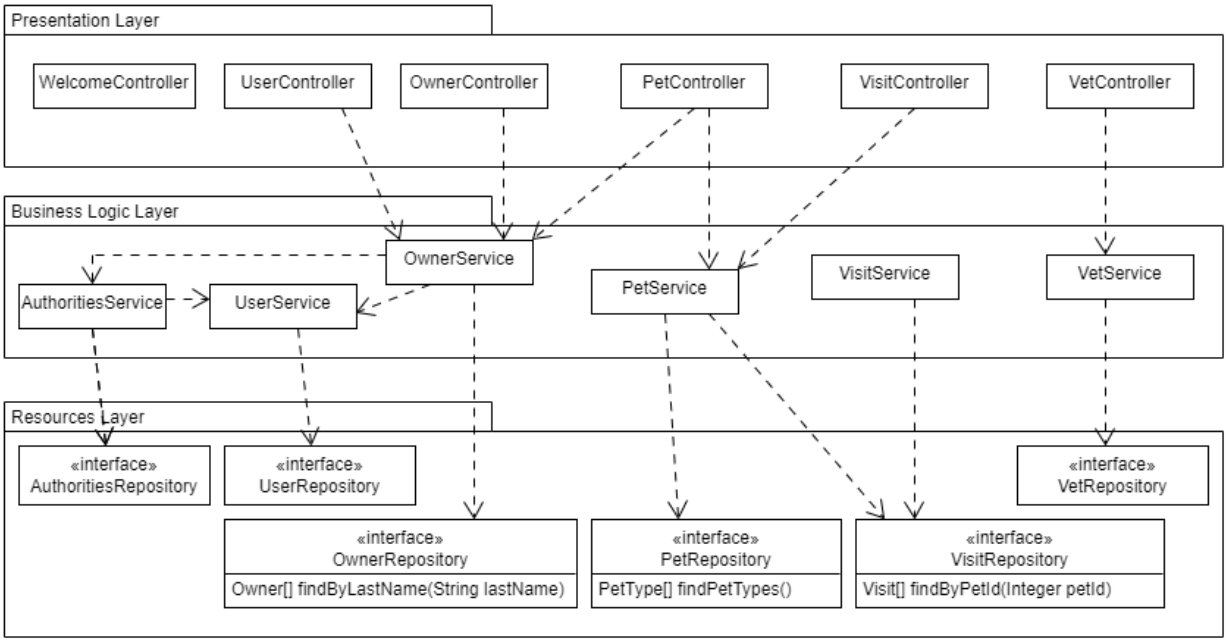


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

En esta sección debe proporcionar un diagrama UML de clases que describa el conjunto de controladores, servicios, y repositorios implementados, incluya la división en capas del sistema como paquetes horizontales tal y como se muestra en el siguiente ejemplo:



El diagrama debe especificar además las relaciones de uso entre controladores y servicios, entre servicios y servicios, y entre servicios y repositorios.

Tal y como se muestra en el diagrama de ejemplo, para el caso de los repositorios se deben especificar las consultas personalizadas creadas (usando la signatura de su método asociado).

Patrones de diseño y arquitectónicos aplicados

En esta sección de especificar 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: <Nombre del patrón>

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

Describir las partes de la aplicación donde se ha aplicado el patrón. Si se considera oportuno especificar el paquete donde se han incluido los elementos asociados a la aplicación del patrón.

Clases o paquetes creados

Indicar las clases o paquetes creados como resultado de la aplicación del patrón.

Ventajas alcanzadas al aplicar el patrón

Describir porqué era interesante aplicar el patrón.

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 puden incluir las vestajas 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 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 require nada más que escribir el SQL que genere 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 lo 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.