DP1 2020-2021

Documento de Diseño del Sistema

Proyecto MyGym

https://github.com/gii-is-DP1/MyGym

Miembros:

* Manuel Ales Rogríguez
* Jose Manuel Lobato Troncoso
* Manuel Outeiriño Barneto
* Borja Vera Casal

Tutor: Irene Bedilia Estrada Torres

GRUPO G3-05

Versión 1

13/01/2021

# 

# Historial de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción de los cambios** | **Sprint** |
| 13/01/2021 | V1 | * Creación del documento | 3 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[**Historial de versiones**](#_heading=h.1zbtetngp8rn) **2**

[Introducción](#_heading=h.30j0zll) **4**

[Diagrama(s) UML:](#_heading=h.1fob9te) **4**

[Diagrama de Dominio/Diseño](#_heading=h.3znysh7) 4

[Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)](#_heading=h.2et92p0) 5

[Patrones de diseño y arquitectónicos aplicados](#_heading=h.tyjcwt) **5**

[Patrón: <Nombre del patrón>](#_heading=h.v3lxv9nn9q7b) 5

[Tipo: Arquitectónico | de Diseño](#_heading=h.ajh17xrz773f) 5

[Contexto de Aplicación](#_heading=h.7jqz8ixdozuk) 5

[Clases o paquetes creados](#_heading=h.x4egv8mzo8io) 5

[Ventajas alcanzadas al aplicar el patrón](#_heading=h.mrn178u01lh) 6

[Decisiones de diseño](#_heading=h.3dy6vkm) **6**

[Decisión 1](#_heading=h.1t3h5sf) 6

[Descripción del problema:](#_heading=h.4d34og8) 6

[Alternativas de solución evaluadas:](#_heading=h.2s8eyo1) 6

[Justificación de la solución adoptada](#_heading=h.17dp8vu) 6

[Decisión 2](#_heading=h.pyotm0b4019o) 7

[Descripción del problema:](#_heading=h.sfv00hq09w91) 7

[Alternativas de solución evaluadas:](#_heading=h.4d2ex46l3nb5) 7

[Justificación de la solución adoptada:](#_heading=h.mg4fcpfbcv24) 7

[Decisión 3](#_heading=h.c1dr8x8szmwa) 8

[Descripción del problema:](#_heading=h.6xw02lxjvqej) 8

[Alternativas de solución evaluadas:](#_heading=h.3ei6zh8ea1j8) 8

[Justificación de la solución adoptada](#_heading=h.bunv628m8rn) 8

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

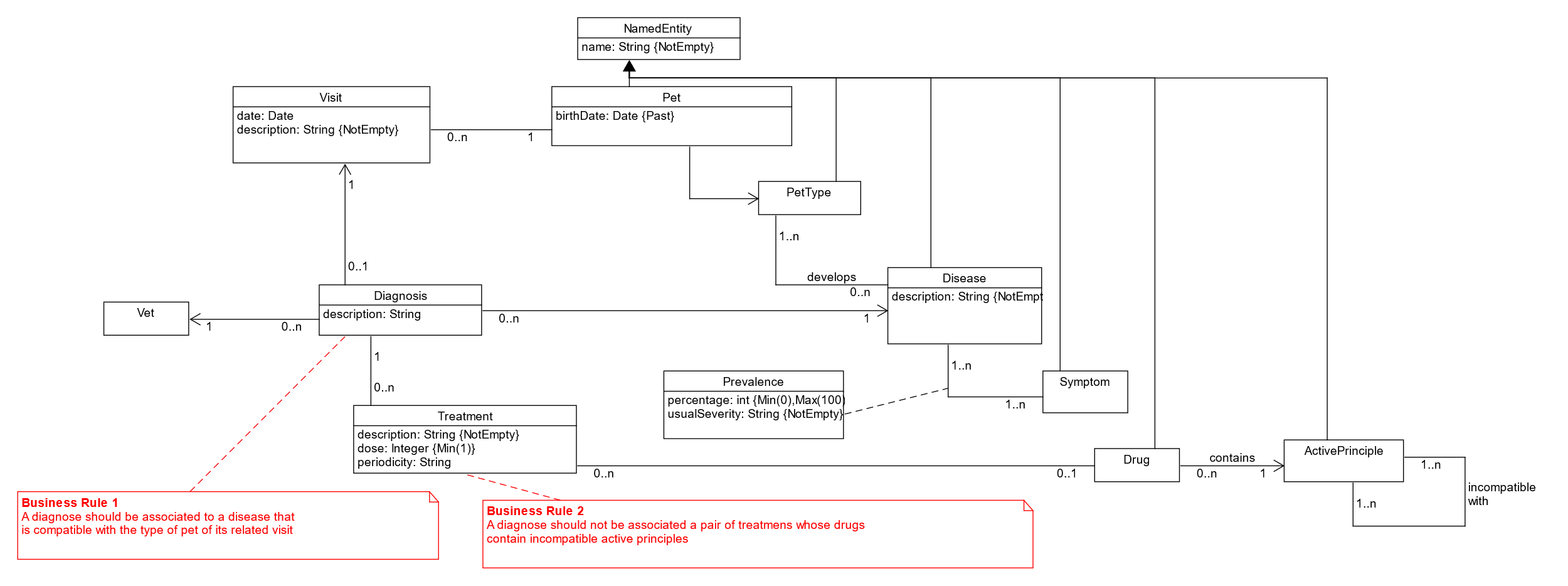
*En esta sección debes describir de manera general cual es la funcionalidad del proyecto a rasgos generales (puedes copiar el contenido del documento de análisis del sistema). Además puedes indicar las funcionalidades del sistema (a nivel de módulos o historias de usuario) que consideras más interesantes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

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

## 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:

Diagram

Description automatically generated

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: MVC

### Tipo: Arquitectónico

### Contexto de Aplicación

Se habilita un paquete donde se declaran todos los controladores de las distintas vistas definidas en los distintos ficheros *jsp* y que gestionan el modelo de la aplicación (“org.springframework.samples.petclinic.model”).

Cada controlador contiene un mapeo con las distintas direcciones desde las que se puede gestionar su modelo asociado, indicando que vista debe de utilizarse según el resultado de cada operación.

### Clases o paquetes creados

La distribución se las distintas clases se dispone de la siguiente manera:

* Paquete de controladores: org.springframework.samples.petclinic.web
  + Ejemplo: UserController, WorkoutController, ActivityController, etc.
* Paquete de modelos: org.springframework.samples.petclinic.model
  + Ejemplo: User, Workout, Activity, etc.
* Directorio de vistas: src/main/webapp/WEB-INF/jsp
  + Ejemplo: vistas del modelo User: users/listadoUsuarios.jsp, users/detalleUsuario.jsp, etc.

### Ventajas alcanzadas al aplicar el patrón

Se trata de un patrón que favorece una cohesión alta (cada vista puede centrarse en cumplir un único objetivo) y un bajo acoplamiento (cada vista puede ser completamente independiente del resto) lo que permite dar soporte a múltiples vistas con un mismo controlador y modelo.

## Patrón: *Layers*

### Tipo: Arquitectónico de Diseño

### Contexto de Aplicación

Se ha aplicado este estilo arquitectónico para la definición de los servicios implementados en la aplicación. Se han establecido un total tres capas:

* Capa de acceso a datos o recursos (repository): define todas las operaciones que se pueden realizar para cada modelo (CRUD, consultas específicas, etc) de forma atómica.
* Capa de lógica de negocio (service): da soporte a toda la lógica de negocio que soporta la aplicación haciendo uso de los repositorios.
* Capa de presentación (controllers): Haciendo uso de la capa de servicio se encarga de representar en la vista la información necesaria.

### 

### Clases o paquetes creados

Los paquetes asociados a cada capa son los siguientes:

* Capa de recursos: org.springframework.samples.petclinic.repository
* Capa de lógica de negocio: org.springframework.samples.petclinic.service
* Capa de presentación: org.springframework.samples.petclinic.web

### Ventajas alcanzadas al aplicar el patrón

A pesar de comprometer el rendimiento y de complicar el mantenimiento/desarrollo favorece muchísimo la cohesión además de permitir una división más clara de las responsabilidades de cada capa y de reducir el acoplamiento. Las capas son independientes entre sí por lo que su reutilización por ejemplo para implementar un acceso a datos mediante microservicios no tendría por qué afectar al negocio de la aplicación ni mucho menos a la presentación de la información.

# 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 1

### Descripción del problema:

Hay que definir qué entidad va a determinar el rol de un usuario dentro de una sesión y el tipo de relación que va a tener esta.

### Alternativas de solución evaluadas:

*Alternativa 1.a:* Utilizar la entidad “Authorities” ya incluída en la aplicación para determinar los distintos roles/permisos que puede tener un usuario en una sesión.

**Ventajas**:

* Al ser un listado de permisos permite diversificar las distintas acciones que se pueden realizar en la aplicación.

**Inconvenientes**:

* Requiere un mapeo más exhaustivo.
* La gestión de los permisos a la hora de crear/editar un usuario es más compleja.
* Hay que agrupar los permisos por rol/tipo de usuario.

### 

*Alternativa 1.b:* Utilizar una entidad “TipoUsuario” que determine el tipo de usuario que se está autenticando contra el sistema y le asigne un único rol (el propio tipo).

**Ventajas**:

* Simplifica la gestión de los permisos de un usuario.
* Al ser un único permiso

**Inconvenientes**:

* Las acciones que se pueden realizar no pueden estar muy diversificadas.

### Justificación de la solución adoptada

Según el modelo de la aplicación los usuarios están relacionados con un tipo de usuario. Por simplicidad en diseño y desarrollo y porque los requisitos de la aplicación no requieren un control exhaustivo de los permisos de cada acción se decide optar por la alternativa 1.b.

## Decisión 2

### Descripción del problema:

Un usuario tiene asignado una rutina, y esta consta de un entrenamiento por cada día de la semana. El entrenamiento está formado por una lista de ejercicios.

Se tiene la necesidad de personalizar los entrenamientos y ejercicios para cada usuario.

### Alternativas de solución evaluadas:

*Alternativa 2.a:* Los monitores crearán entrenamientos y ejercicios que serán utilizados como plantillas para después ser asignados a los usuarios. Diremos que estos entrenamientos y ejercicios son genéricos. Al asignar una rutina a un usuario asignaremos una copia del entrenamiento genérico (junto a sus ejercicios, también copiados) para que puedan ser personalizados según las necesidades del usuario.

**Ventajas**:

* Al crear la rutina del usuario, partimos de un entrenamiento genérico con ejercicios asignados que podemos personalizar para el caso concreto de cada usuario.
* Este grado de personalización permite separar los entrenamientos genéricos de los personalizados.

**Inconvenientes**:

* La posibilidad de tener registros de rutinas con entrenamientos y ejercicios idénticos (registros duplicados).
* El número de registros en las tablas.
* Baja cohesión de la información: no se podrá editar de forma común un entrenamiento o ejercicio.

### 

*Alternativa 2.b:* Crear rutina desde cero. Al asignar entrenamientos a la rutina, tendríamos que asignar un entrenamiento vacío y añadir los ejercicios al mismo, sin tener la posibilidad de seleccionar un entrenamiento prediseñado.

**Ventajas**:

* No tener la necesidad de crear entrenamientos y ejercicios genéricos.

**Inconvenientes**:

* Tener que crear rutinas y ejercicios partiendo de cero, sin tener la posibilidad de usar uno genérico que nos ahorre el tener que seleccionar los días de entrenamientos y sus respectivos ejercicios uno a uno.

### Justificación de la solución adoptada:

Creemos que para la usabilidad de la aplicación y para facilitar el trabajo de los monitores la alternativa 2.a es la más conveniente para nuestra aplicación. De esta forma, un monitor puede asignar las rutinas de un usuario según sus necesidades de forma más rápida y eficaz.

## Decisión 3

### Descripción del problema:

La versión de Bootstrap que incorpora la plantilla del proyecto está anticuada y su versión actual incluye features que pueden facilitar el desarrollo del proyecto.

### Alternativas de solución evaluadas:

*Alternativa 1.a:* Utilizar MDBootstrap (una implementación del patrón Material Design con Bootstrap.

**Ventajas**:

* Diseño más innovador y usable.
* Se unifican los componentes UI con los de Bootstrap en una misma librería por lo que se puede prescindir de Bootstrap-UI.
* Se aprovechan todas las features de la versión más reciente de Bootstrap.

**Inconvenientes**:

* Hay que adaptar los layouts y componentes que vienen con la aplicación a la nueva tecnología.

### 

*Alternativa 1.b:* Actualizar la versión de Bootstrap y Bootstrap-UI a su versión compatible más reciente..

**Ventajas**:

* Se aprovechan todas las features de la versión más reciente de Bootstrap.

**Inconvenientes**:

* Revisar los breaking changes y adaptar las vistas a las nuevas versiones.

*Alternativa 1.c:* Seguir usando las versiones de Bootstrap y Bootstrap-UI incluídas en la plantilla del proyecto.

**Ventajas**:

* No hay que crear un nuevo diseño.
* Se pueden reutilizar todas las plantillas incluídas en los ejemplos.

**Inconvenientes**:

* Debido a que la versión está anticuada, cuando nos surjan problemas y busquemos en foros, no encontraremos soluciones para dicha versión.

### Justificación de la solución adoptada

Elegimos la alternativa 1.a para así aprovechar las features de la última versión y trabajar con un diseño más innovador que aumente la motivación del equipo.