UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

DISEÑO DE SISTEMAS

Trabajo Práctico Anual

“Sistema de Gestión Energética”

Grupo: 04

Integrantes:

- Matias Vivone
- Martín Javier Gauna
- Gustavo Matías Di Peppe
- Gonzalo Giliberti
- Juan Martin Conde

Fecha de entrega: 15/07/2018

Profesor: Martín Agüero

Ayudante a cargo: Alejandro Leoz

Repositorio: https://github.com/MartinGauna/SGE-G4

Branch: entrega2

Commit ID: ab1dad07d6210971be830232be8a95a425783b82

Trabajo Práctico Anual

“Sistema de Gestión Energética”

Registro de cambios

Fecha	Modificaciones
28/04/2018	Entrega 0
25/05/2018	Entrega 1
14/07/2018	Entrega 2

Entrega 0

Modificaciones

Se creó un diagrama de clases, se empezaron a programar las mismas, y se hizo un diagrama de casos de uso.

Se agregaron el detalle para el caso de uso "Consultar Saldo aproximado a pagar".
Se modificó el diagrama de clases, dejando 1 sola clase para manejar las categorías.
Se agregó una descripción de la arquitectura.

Diagrama de Clases:

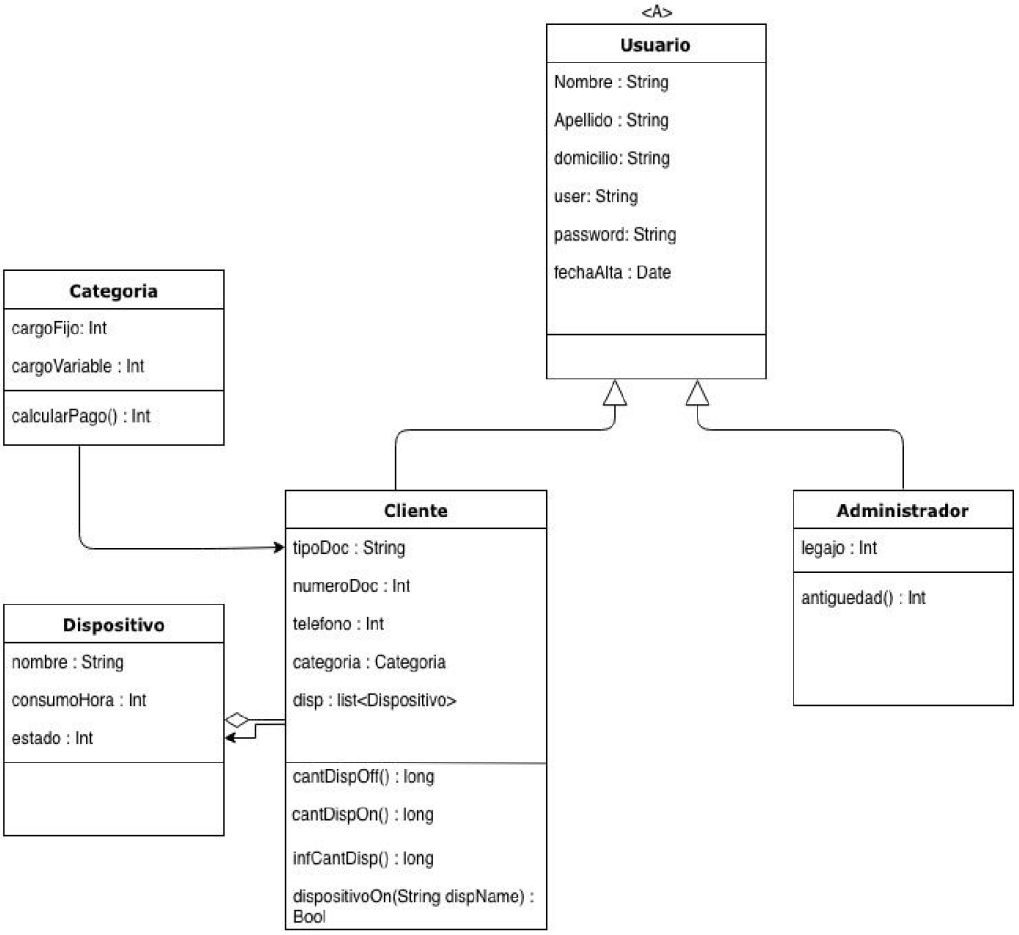
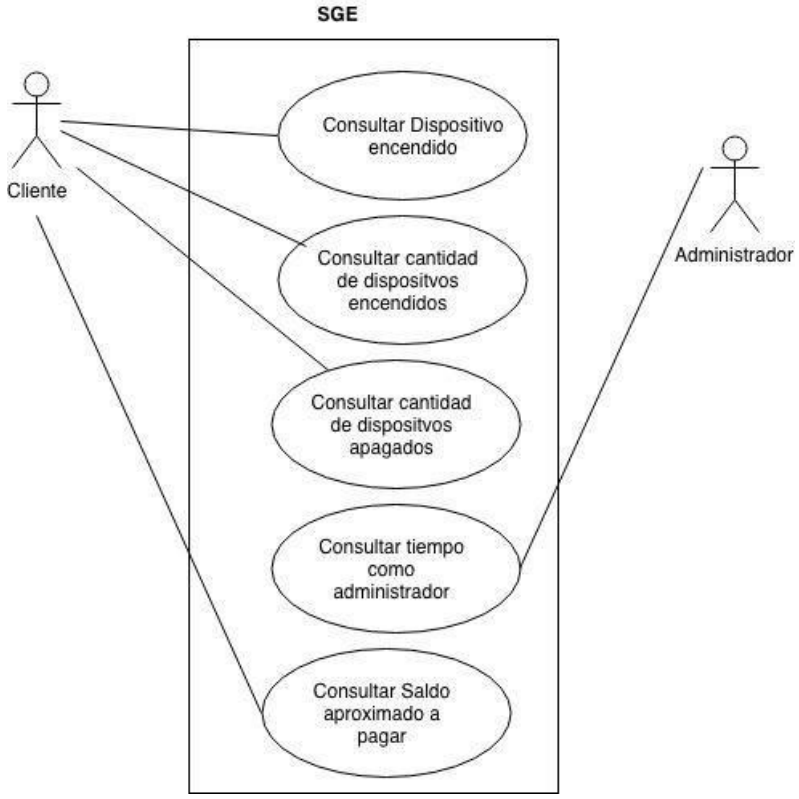


Diagrama de casos de uso:



Detalle del caso de uso "Consultar Saldo aproximado a pagar":

Nombre de caso de uso	**Consulta de saldo aproximado a pagar**	
Requisito asociado	Ninguno	
Objetivo	Obtención de valor aproximado de saldo a pagar.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un cliente realice la consulta del saldo aproximado a pagar	
Actores	Cliente	
Precondiciones	El cliente debe poseer una categoría asociada (R1-R9)	
Secuencia Normal	**Paso**	**Acción**
	1	El cliente solicita al sistema consultar el saldo aproximado a pagar
	2	El sistema obtiene el valor de kWh en el periodo actual.

	3	De acuerdo a la categoría asociada al usuario, se realiza el cálculo de acuerdo a los valores de cargo fijo y cargo variables.
	4	Se muestra por pantalla el resultado del cálculo de saldo a pagar
Condición de fin exitosa	El cliente cuenta con el valor aproximado de saldo a pagar.	
Condición de fin fallida	El cliente no obtiene el valor aproximado de saldo a pagar.	
Excepciones	**Paso**	**Acción**
	1	Si el usuario no posee categoría asociada, se maneja la excepción informando la misma por pantalla. A continuación, el caso de uso termina.

Diagrama de secuencia de "Consultar Saldo aproximado a pagar":

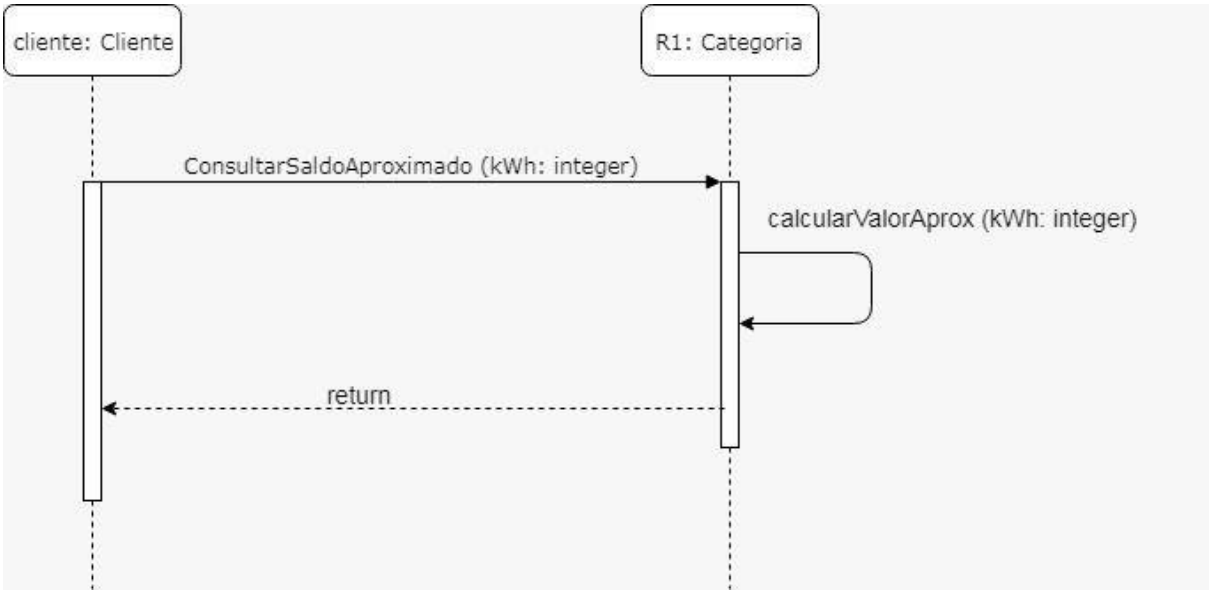
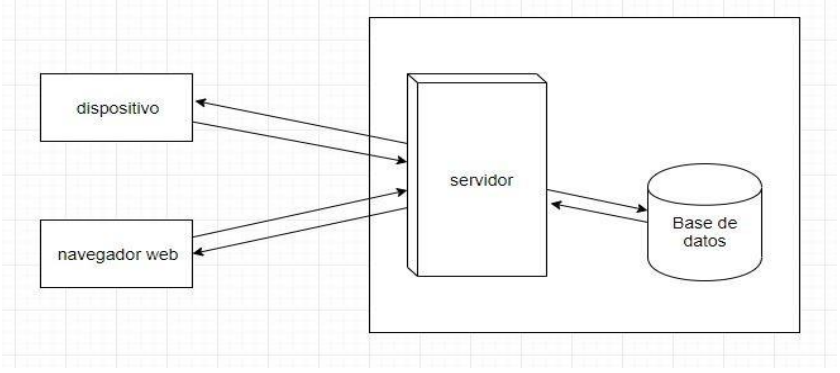


Diagrama de Arquitectura:



La arquitectura del proyecto será Cliente-Servidor. Donde un cliente puede ser un navegador o un dispositivo inteligente/adaptador. Dado que el proyecto está pensado para capital federal se analizará la posibilidad de tener más de un server con un balanceador de carga debido al gran número de habitantes en ciertas zonas de la ciudad, sobre todo el microcentro.
Nuestra solución contará con 3 capas:

- Front-end: Será una aplicación web, hecha en Javascript, JQuery y bootstrap.
- Back-end: Utilizaremos Java como tecnología de desarrollo y Nhibernate para la comunicación con la base de datos.
- Base de Datos: Hemos decidido utilizar MySQL como motor de base de datos. Estamos considerando la posibilidad de usar MongoDB pero de momento vamos a empezar con MySQL.

Requerimientos no funcionales

1) Estética (UX)

a) Los elementos de la interfaz deberán verse y comportarse tal como fueron pensados para la interacción del usuario, así como la rotulación de los elementos acordes al modelo mental y lenguaje de usuario que fueron propuestos y testeados por el/los diseñador/es.
b) Mediante el uso de performance tracking (implementado con Google Analtics) se analizará las funcionalidades más usadas y el mayor porcentaje de abandono.
c) Se analizará, mediante la medición de clicks en call to actions, que las acciones consideradas importantes por el cliente, según su requerimiento, tengan el tráfico esperado. Asegurando así, una buena comprensión y conducción en la experiencia de usuarios.

2) Performance

a) Se asegurará una velocidad de rendimiento/optimización del sitio web mayor a 85 puntos (benchmark sugerido) a través de su medición con PageSpeed Insights
b) Se asegurará una velocidad de carga del sitio web menor a 1 segundo a través de su medición con PageSpeed Insights
c) Se medirá la tasa de defectos: número de defectos / LOC, FP
d) Se medirá la confiabilidad: número de fallas / N horas de operación

3) Seguridad

a) Los diferentes roles de usuarios que interactúen con la aplicación deberán comportarse tal como se especifica en los requerimientos.
b) El sitio web estará hospedado bajo el protocolo HTTPS cumpliendo con todas las cláusulas de seguridad que dicho standard conlleva.

4) Durabilidad

a) Al comienzo de cada sprint (entrega) se realizará una refactorización del sistema garantizando que su diseño y arquitectura implementada en sprints anteriores permita una escalabilidad óptima lo cual conllevará a una alta cohesión y un bajo acoplamiento.

Entrega 1

Modificaciones

Se actualizó el diagrama de clases de acuerdo con el enunciado.

Se crearon las nuevas clases y sus constructores.

Se programaron los compartimientos de las nuevas clases de acuerdo con el enunciado.

Se generaron los tests necesarios para comprobar el buen funcionamiento de los métodos.

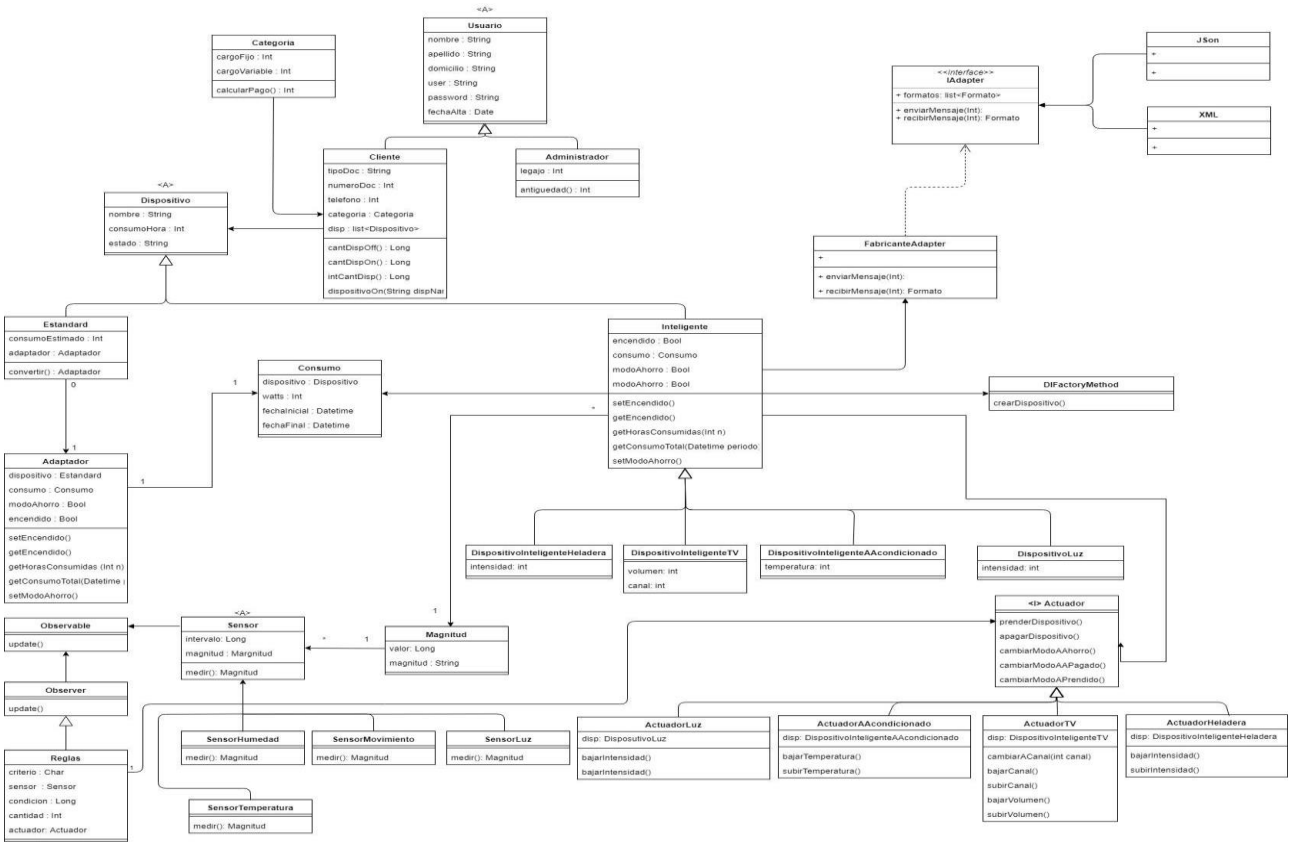
Se transformo la clase Dispositivo

Se hicieron diferentes fixes acordados en la revisión.

Patrones de diseño usados:

- Factory Method (adaptado): creamos una clase *DIFactoryMethod* que de acuerdo con el nombre que le envía el dispositivo ejecuta un case y genera la instancia de la clase específica con sus actuadores correspondientes.
- Observer: Para la implementación de Reglas creamos un Observer que se fija si el sensor indicado ha cambiado de valor, de ser así, considera implementar la Regla dependiendo de la condición.

Diagrama de Clases actualizado:



Entrega 2

Zonas y transformadores

Las zonas se cargan a través de un JSON y conoce su longitud, latitud y la lista de Transformadores a los que estará asociados.
Los transformadores también se crean con JSON. Para mantener solo los activos (que serán enviados por el ENRE en el formato mencionado) creamos una propiedad estática para hacer un getAll(). Luego desde el JSON Parser antes de importar todos los transformadores nuevos limpiamos los existentes:

```
List<Transformador> transformadoresViejos = Transformador.getAll();
    for (Transformador t : transformadoresViejos)
    {t = null;}
```

Esto solo borra los que estan en memoria, en un futuro habra que hacer un refactor para borrar los que esten en la base de datos.

Test unitarios

- ZonaTest →
    - parseJSON(): para probar el JSON parser y los getter y setters de Zona.
    - getConsumoTotal(): para el getConsumo por zona que sumariza todos los transformadores de la zona.

Tipos y subtipos de Dispositivos Inteligentes

Los tipos de dispositivos inteligente son subclases y a su vez las subcategorias de los dispositivos inteligentes, por ejemplo, DispositivoInteligente → DispositivoInteligenteLuz → DispositivoInteligenteLuz11W

En este caso:

- DispositivoInteligenteLuz conoce su consumoMinimo y consumoMaxico
- DispositivoInteligenteLuz11W conoce el Consumo y si es de BajoConsumo.

Test unitarios

- JsonParserTest →
    - loadDispositivosInteligentesJSON(): esto probará el correcto funcionamiento del FactoryMethod con los tipos y subtipos de Dispositivos Inteligentes.

Hogar eficiente, Simplex y Automatización

Se importó la librería Simplex propuesta por la catedra. Se desarrolló un Adapter para desacoplar la solución y poder simplificar en un futuro adoptar otra librería.
Para la automatización del apagado de dispositivos con Simplex se usaron las clases java.util.Timer y java.util.TimerTask y así creamos una función automatizacionAhorroAutomatico() que apaga los dispositivos revisando la función simplex cada x milisegundos (configurable) si el Cliente en cuestión tiene la propiedad ahorroAutomatico en True.

Test unitarios:

- SimplexSolverTest: testeará que la libreria Simplex, propuesta por la cátedra, se haya importado correctamente
- AdapterSimplexTest: comprobará el adapter que interactúa entre el usuario y el SimplexSolver. Se presenta el escenario de un cliente con 3 dispostivos y se genera el reporteConsumoEficiente();
- ClienteTest → ahorroAutomatico() antes de configurar la automatización probaremos que pasando un cliente se pueden apagar todos los dispositivos que consumieron más que lo que el simplex sugiere, para luego reutilizar esta función automatizándola en intervalos de tiempo.
- BackgroundProcessesTest → automatizacionAhorroAutomatico() chequea el Simplex cada 5 segundos y apaga los dispositivos que corresponda (test hecho durante 20 segundos con Thread.Sleep)

Diagrama de clases actualizado: