

# Arquitectura Modelo 4+1 Vistas - Body Fitness Gym.

<<versión 0.5>>

Historial de Versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 18/11/2017 | 0.3 | Diagramas Vista Lógica | Gabriel Amaya |
| 23/11/2017 | 0.5 | Diagramas Vista de Desarrollo | Gabriel Amaya |

Contenido

[Arquitectura Modelo 4+1 Vistas - Body Fitness Gym. 1](#_Toc499324977)

[1. Introducción 4](#_Toc499324978)

[1.1. Propósito 4](#_Toc499324979)

[1.2. Alcance 5](#_Toc499324980)

[1.3. Glosario de términos 5](#_Toc499324981)

[1.4. Organización del documento 5](#_Toc499324982)

[2. Representación de la Arquitectura. 5](#_Toc499324983)

[3. Objetivos y Restricciones 7](#_Toc499324984)

[4. Vista de Casos de Uso 8](#_Toc499324985)

[5. Vista Lógica 8](#_Toc499324986)

[5.1. Diseño de subsistemas 8](#_Toc499324987)

[5.1.1. Módulo de usuarios 8](#_Toc499324988)

[5.1.2. Módulo de Servicios 12](#_Toc499324989)

[5.1.3. Módulo de Contabilidad 14](#_Toc499324990)

[5.1.4. Módulo de Suscripciones 17](#_Toc499324991)

[5.2. Acotaciones Generales 19](#_Toc499324992)

[6. Vista de Desarrollo o Despliegue 20](#_Toc499324993)

# Introducción

Este documento provee información detallada sobre la arquitectura para el diseño y desarrollo de una aplicación web para la administración de la empresa BODY FITNESS GYM LTDA. Tiene como guía el documento [nombre del documento], el cual especifica los requisitos tanto funcionales como no funcionales del sistema para el análisis del sistema desde diferentes vistas, las cuales permiten visualizar diferentes componentes del software, sus interacciones y actores involucrados. Se tiene por objetivo que este documento sirva de guía para el entendimiento del sistema a todos los interesados y/o involucrados en el mismo (*stakeholders*, **desarrolladores, y directores de grupo**).

## Propósito

Se describe detalladamente la visión completa de la arquitectura del sistema, usando diferentes vistas arquitectónicas: **vista lógica, vista de procesos, vista de desarrollo, vista física y vista de escenarios (+1) siguiendo el Modelo de Vistas de Arquitectura 4+1** con el objetivo de resaltar aspectos que conciernen a cada vista y permiten identificar las diferentes dimensiones del sistema, así como roles, procesos y tareas involucradas.

## Alcance

La visión arquitectónica que engloba este documento comprende únicamente a BODY FITNESS GYM LTDA como sistema. Mediante el Modelo de Vistas de Arquitectura 4+1 hace un análisis de alto nivel de las capacidades del software a desarrollar en cada una de las vistas con las limitantes dadas por el documento de requisitos especificados y las reglas del negocio consignadas en el documento análisis del sistema, por lo que características menores o no incluidas en dicho documento son obviadas o consideradas irrelevantes para el desarrollo del software.

## Glosario de términos

Véase Glosario General de Términos [aquí el nombre del documento].

## Organización del documento

La estructura de este documento está basada en la plantilla provista por para el artefacto *Software Architecture Document* del proceso de desarrollo de software elaborado por RUP [2].

En la sección 2 se hace una breve introducción a lo que trata el Modelo de Vistas de Arquitectura 4+1 vistas en lo que respecta a su definición y alcance, con el ánimo de ofrecer un entendimiento claro y conciso de cada una de las vistas a analizar dentro del sistema BODY FITNESS GYM.

En la sección 3 se hace una revisión de los objetivos y restricciones del sistema en lo que concierne a los requisitos funcionales y no funcionales, de los cuales, para este documento, se mencionan los que tienen mayor impacto dentro del funcionamiento general del sistema.

# Representación de la Arquitectura.

El modelo de vistas múltiples, organiza una descripción de la arquitectura de software utilizando cinco vistas concurrentes, las cuales permiten aproximar de manera aislada los intereses de los diferentes *stakeholders* de la arquitectura: los usuarios finales, los desarrolladores, entre otros; y manejar de manera separada los requerimientos funcionales y no funcionales [1]. Arquitectos capturan sus decisiones de diseño en cuatro de las vistas y utilizan la quinta vista para ilustrar y validarlas. El modelo propone las siguientes perspectivas o vistas [1]:

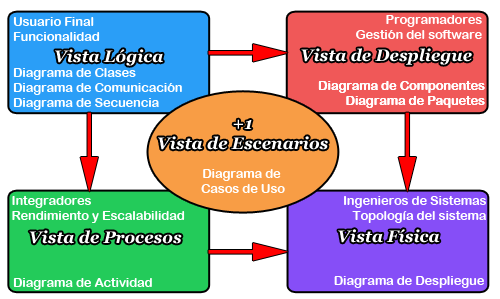


Figura . Modelo 4+1 Vistas. Fuente: [2]

* **Vista Lógica:** En esta vista se representa la funcionalidad que el sistema proporcionara a los ***usuarios finales.*** Es decir, se ha de representar lo que el sistema debe hacer, y las funciones y servicios que ofrece. Está profundamente influenciada por las reglas del negocio, en el cual se identifican módulos específicos denominados paquetes que interactúan con otros dentro del sistema [2].
* **Vista de Desarrollo o Despliegue**: En esta vista se muestra el sistema desde la perspectiva de ***un programador*** y se ocupa de la gestión del software; o en otras palabras, se va a mostrar cómo está dividido el sistema software en componentes y las dependencias que hay entre esos componentes. Para completar la documentación de esta vista se pueden incluir los diagramas de componentes y de paquetes de UML.
* **Vista de Procesos**: describe el diseño de concurrencia y aspectos de sincronización. Especifica las líneas de mando que ejecutan cada operación en cada una de las clases señaladas en la vista lógica [3] Se representa desde la perspectiva de un ***integrador de sistemas***, el flujo de trabajo paso a paso de negocio y operacionales de los componentes que conforman el sistema [2].
* **Vista Física**: La vista física representa el sistema desde el punto de vista de un ingeniero de sistemas. Se refiere a la topología de los componentes de software en la **capa física**, así como a las conexiones físicas entre estos componentes. Esta vista también se conoce como la vista de despliegue [4].
* **Vista de Escenarios (+1):** Esta vista es representada por los casos de uso  software y va a tener la función de unir y relacionar las otras 4 vistas, esto quiere decir que desde un caso de uso se puede ver cómo se van ligando las otras 4 vistas, con lo que se tiene una trazabilidad de componentes, clases, equipos, paquetes, etc., para realizar cada caso de uso. Para completar la documentación de esta vista se pueden incluir los diagramas de casos de uso de UML.[2]

# Objetivos y Restricciones

Como objetivo primordial del software a desarrollar para BODY FITNESS GYM se tiene que, en forma general, se cree un sistema que permita la administración de diferentes módulos que se manejan a la fecha en la empresa, entre los que se incluyen módulos contables y registros de usuarios de forma transaccional, con operaciones tipo *CRUD*, todo lo anterior orientado a un entorno *Web*. [AQUÍ FALTARÍA UN POCO MÁS]

Examinando el documento de análisis de requisitos se tiene que los siguientes son requisitos que tienen un impacto directo en la arquitectura seleccionada

* RF\_01\_01: El sistema debe permitir realizar registro, edición, eliminación de alumnos (CRUD) por parte del administrador.
* RF\_01\_03: Permite al administrador registrar una salida de dinero en el gimnasio, con su respectiva descripción.
* RF\_01\_04: Permite al administrador mostrar un balance de los movimientos de dinero ya sea diario, semanal, mensual, trimestral o anual.
* RNF\_01: El programa debe estar desarrollado para un entorno web. Debe poder usarse en navegadores Google Chrome y Mozilla Firefox
* RNF\_09: Los datos concernientes a pagos y listas de usuarios se deberán poder exportar a documentos de texto plano (.txt), a documento de formato portable (.pdf), y a programas de hoja de cálculo (.xlsx)

# Vista de Casos de Uso

# Vista Lógica

En esta vista se describe la estructura ya la funcionalidad del sistema. Para facilitar la comprensión del mismo, se divide en subsistemas que corresponden a módulos o paquetes lógicos bien definidos e interrelacionados con otros subsistemas. Tras un análisis del sistema, se encontraron tres grandes módulos o paquetes lógicos que se describen a continuación

## Diseño de subsistemas

Para el diseño de los subsistemas se hizo uso de **patrones de diseño** (creacionales, estructurales y de comportamiento) para crear una estructura lógica más refinada basada en dichos patrones.

### Módulo de usuarios

Este subsistema es el encargado de administrar los usuarios que existen en el sistema. Según la lógica del negocio, en el sistema deben existir usuarios que pueden cumplir múltiples roles. El dominio de dichos roles está limitado a: **estudiante, entrenador y administrador**  donde el administrador es el que tiene el control total sobre el sistema, tanto dentro del módulo que lo contiene como de los demás módulos que existen el sistema. El **diagrama de clases** que representa este módulo se muestra a continuación

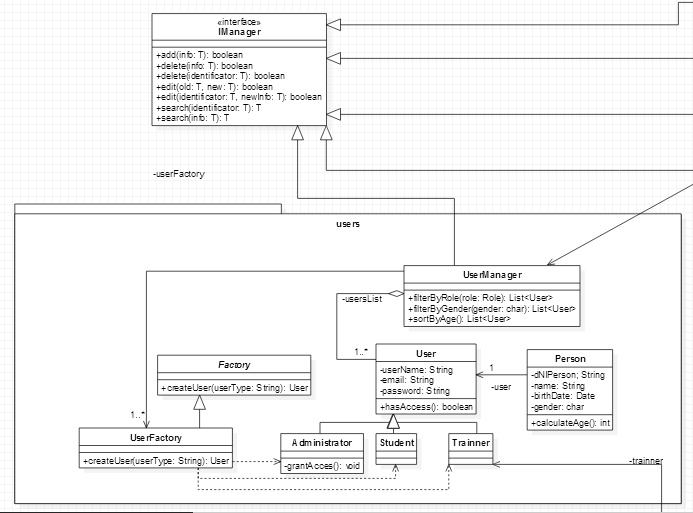


Figura . Diagrama de clases – Módulo Usuarios. Fuente: Autores

En este módulo se hizo uso del patrón de diseño creacional *Factory Method*. La aplicación de este patrón se ve representada en el módulo de usuarios. Dado que en este módulo solo hay una jerarquía, donde los diferentes usuarios (Administrador, Estudiante y entrenador) extienden de un padre común denominado *User*, resulta fácil delegar su creación a una factoría cuyo método de creación se parametriza para obtener el tipo de usuario requerido. Este patrón facilita el trabajo a la clase UserManager para la creación de usuarios, delegando dicha función a una clase asociada que se encarga específicamente de dicha tarea.

Como acotación, este módulo hace uso de la interfaz IManager que provee de métodos tipo CRUD que implementa la clase UserManager para cumplir con sus funciones respectivas; dicha interfaz no está contenida dentro del paquete lógico de usuarios ya que es común a otros módulos.

Las operaciones en este módulo se ilustran a un alto nivel mediante el siguiente **diagrama de secuencia**

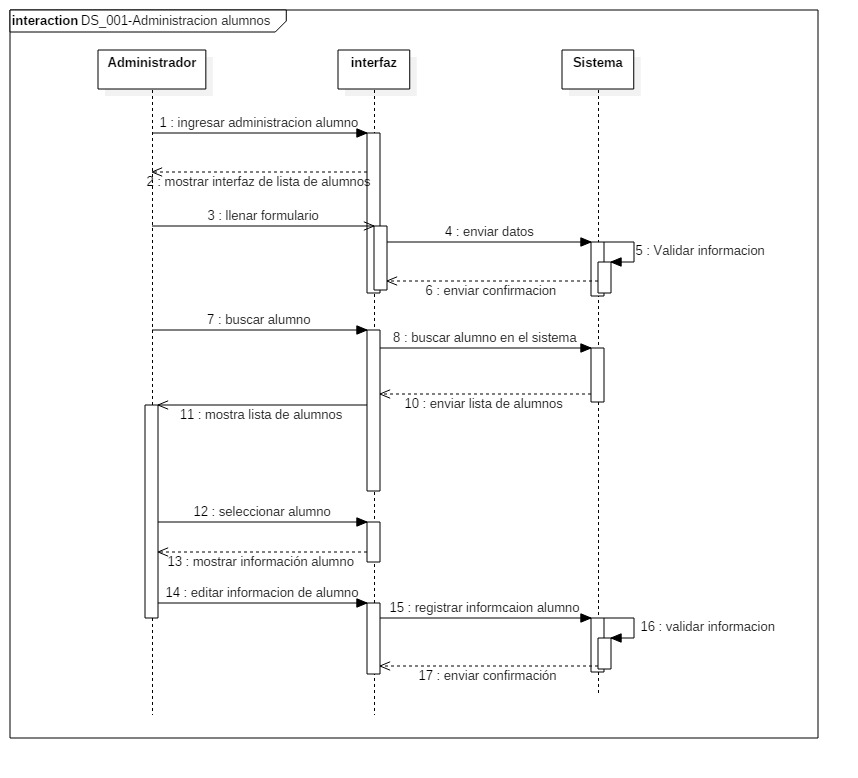


Figura . Diagrama de secuencia – Módulo Usuarios. Fuente: Autores

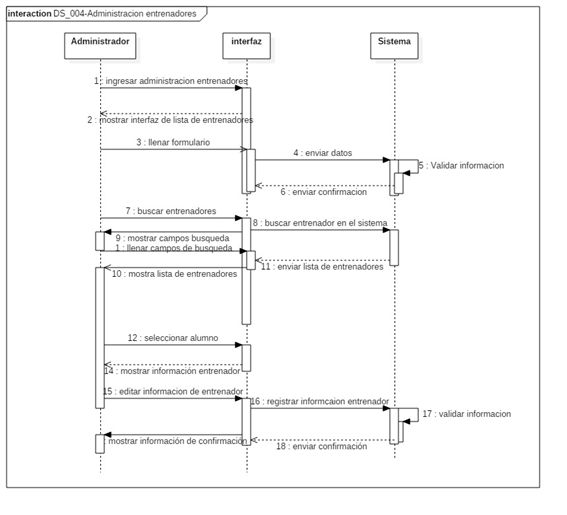


Figura. Diagrama de secuencia – Módulo Usuarios. Fuente: Autores

El procedimiento interno del sistema para la creación y registro de un nuevo usuario en el sistema se muestra en el siguiente **diagrama de comunicación**

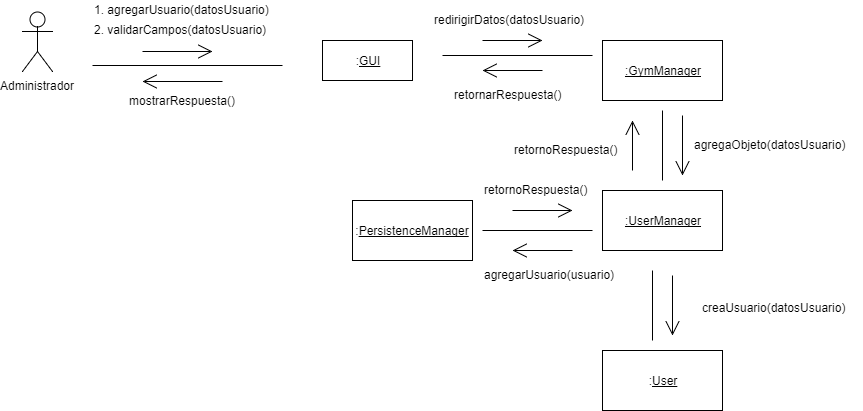


Figura. Diagrama de Comunicación. Módulo usuarios

La figura [#] hace más entendible la comunicación entre clases para el proceso de creación y almacenamiento de usuarios. El administrador, a través de la **GUI,** envía los parámetros del usuario a la interfaz lógica GymManager, la cual delega la tarea a UserManager para crear el nuevo usuario y almacenarlo comunicándose con PersistenceManager. De regreso, la interfaz lógica devuelve una respuesta de éxito/fracaso del proceso a la GUI para mostrar al administrador.

### Módulo de Servicios

Este subsistema es el encargado de administrar los programas que ofrece el negocio, entre estos se incluye operaciones tipo CRUD sobre estos y sobre los horarios con las sesiones de tales programas y los entrenadores que las dirigen. Tales horarios, según las reglas del negocio son actualizados semanalmente. Como se mencionó anteriormente, la única persona que tiene la facultad para realizar control sobre este módulo es el usuario administrador, los otros usuarios son actores pasivos. El **diagrama de clases** que representa este módulo se muestra a continuación

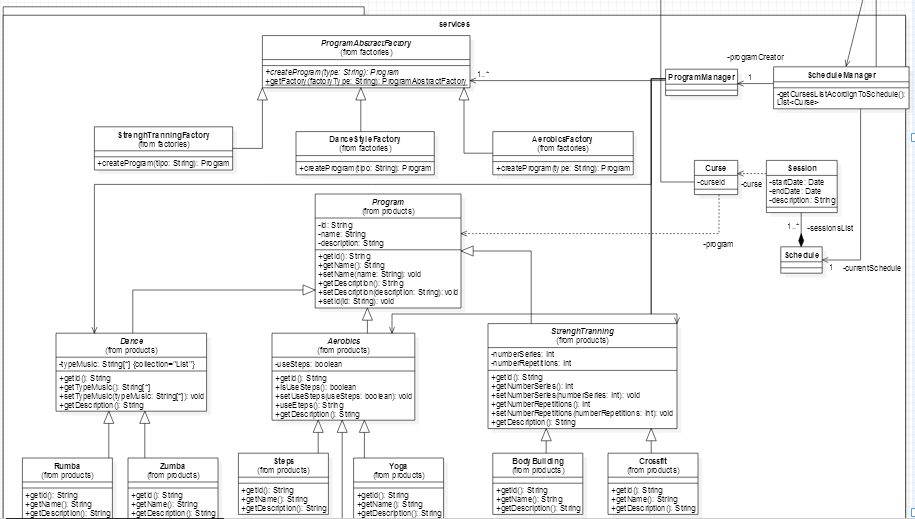


Figura . Diagrama de clases – Módulo Usuarios. Fuente: Autores

En este módulo se hizo uso del patrón de diseño creacional *Abstract Factory.*La aplicación de este patrón surge de la necesidad de crear varias familias de objetos, en este caso, las factorías son: la de entrenamientos de fuerza, estilos de baile, y aeróbicos, cada una asociada a una grande familia de productos que puede crear. Esto permite que se mantenga una independencia de cómo los productos son creados, delegando dicho trabajo a las factorías que crean cada uno de los tipos de programas que existen en el sistema. Dado que, según la descripción del negocio, el sistema no sufrirá alteraciones mayores, no hay que lidiar con el potencial problema de modificar las fábricas abstractas y concretas.

Este subsistema tiene interacción directa con el módulo de usuarios mediante la conexión entre la clase Entrenador y la clase Curso, donde cada curso debe ser dirigido por al menos un entrenador.

Las operaciones en este módulo se ilustran a un alto nivel mediante el siguiente **diagrama de secuencia [AQUÍ FALTA EL DIAGRAMA DE SECUENCIA QUE NO SE HA HECHO]**

### Módulo de Contabilidad

Este subsistema es el encargado de llevar control sobre las transacciones que hace el negocio. Estas transacciones pueden ser tanto **ingresos** como **egresos**. Los ingresos representan el usufructo de las suscripciones pagadas por los estudiantes, los egresos representan simplemente salidas de dinero que, según las reglas del negocio, deben ser fundamentadas y autorizadas por el administrador. El **diagrama de clases** que representa este módulo se muestra a continuación

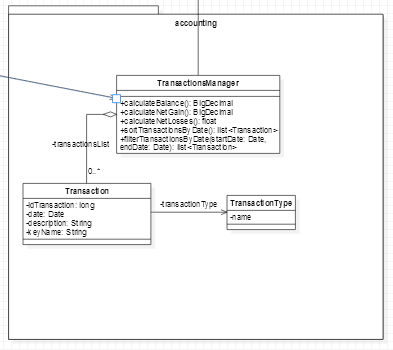


Figura . Diagrama de clases – Módulo Contabilidad. Fuente: Autores

Debido a la simplicidad de este módulo, no se hizo uso de ningún patrón de diseño para su estructuración. Por el contrario, su funcionalidad radica en la clase TransactionsManager que además de hacer las operaciones de CRUD, de acuerdo a los requisitos funcionales, debe realizar informes periódicos (semanales, quincenales, mensuales, trimestrales, anuales) acerca de ganancias, pérdidas, permitir consultas refinadas acerca de atributos de transacciones.

Las operaciones en este módulo se ilustran a un alto nivel mediante el siguiente **diagrama de secuencia**

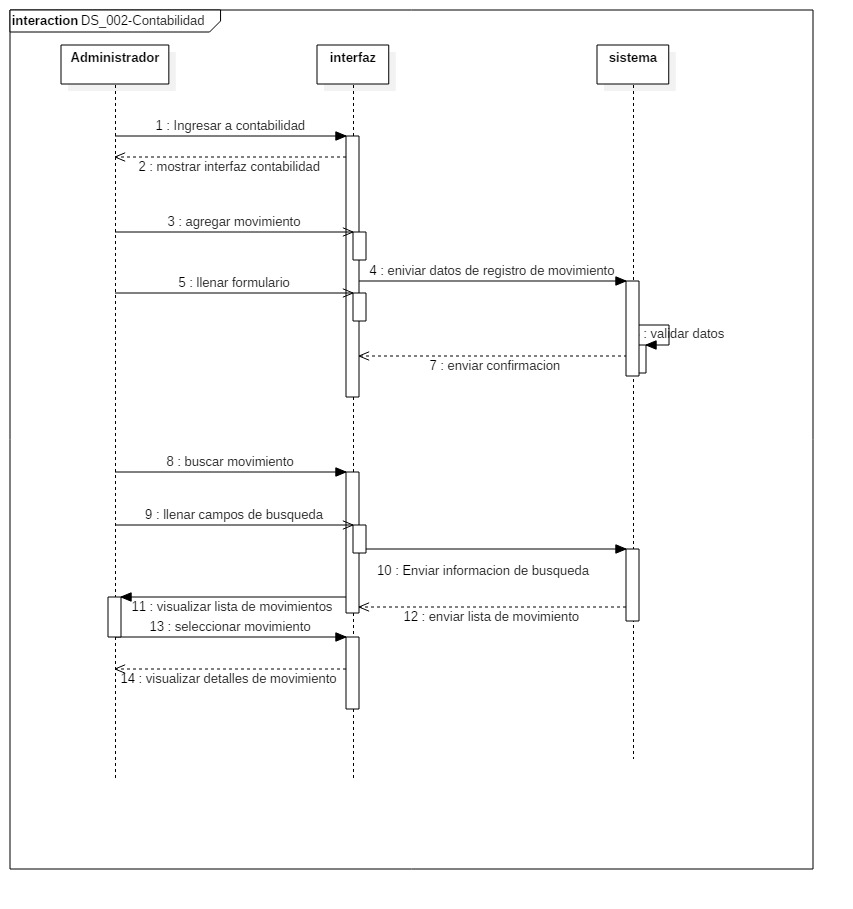


Figura . Diagrama de clases – Módulo Usuarios. Fuente: Autores

Este diagrama es una representación de alto nivel de la secuencia de acciones entre los diferentes actores y componentes involucrados, aquí se muestran las acciones fundamentales que debe hacer el administrador a través de TransactionsManager.

[AQUÍ FALTA AGREGAR DIAGRAMA DE SECUENCIA PARA APLICAR LOS FILTROS Y LAS OPERACIONES QUE HACE ESTE MANAGER]

A nivel de comunicación, el registro de nuevos movimientos es similar al de otros módulos, así, el administrador ingresa los datos del movimiento a través de la GUI, hacia la interfaz lógica GymManager, la cual delega dicha operación de TransactionManager y crea el nuevo movimiento, el cual almacena PersistenceManager y retorna una respuesta de éxito/fracaso en la operación. El siguiente **diagrama de comunicación** resume lo dicho.

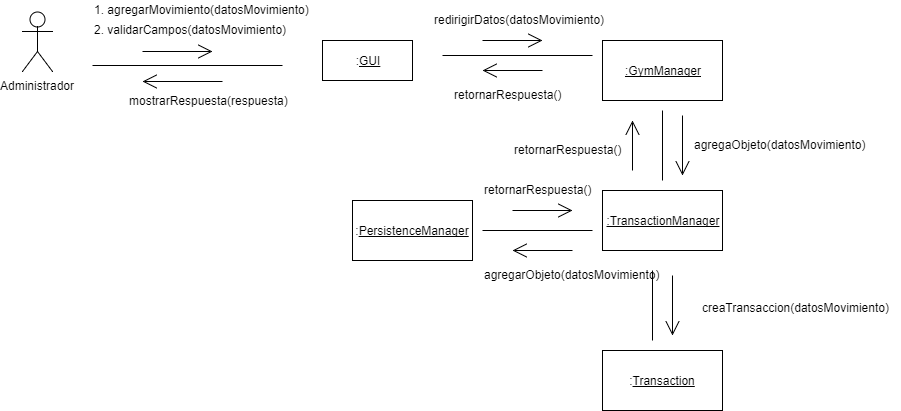


Figura. Diagrama de secuencia – Registro Movimiento – Módulo de Contabilidad. Fuente: Autores

De igual manera, al momento de hacer una consulta parametrizada sobre los datos de este módulo, el administrador puede generar un informe con los datos obtenidos. Para ello, el administrador define unos parámetros para la salida del informe, esto respecto a los datos a mostrar y el formato de los archivos de salida, teniendo en cuenta los requisitos del sistema. A continuación su correspondiente diagrama de comunicación

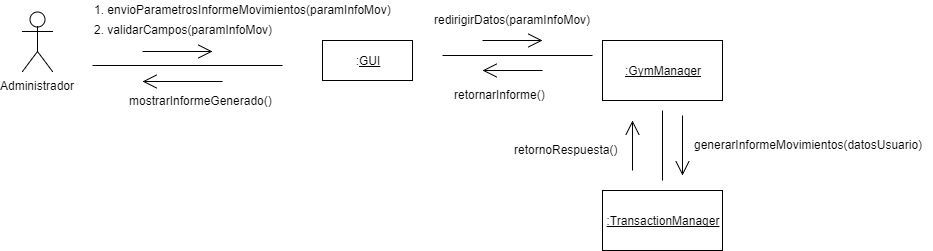


Figura. Diagrama de Comunicación – Generación Informes – Módulo Contabilidad. Fuente: Autores

En la figura # se puede apreciar que el **scope** de la generación de reportes, desde la vista lógica, llega hasta la invocación de TransactionManager. Esto se debe a que la generación física del informe requiere comunicación con dispositivos hardware de salida (ej. Impresora), la cual no se trata en esta vista.

### Módulo de Suscripciones

Este subsistema es el encargado de llevar control sobre las *suscripciones* que realizan para el acceso a diferentes programas por parte de los estudiantes. Como se mencionó anteriormente, estas acciones son supervisadas por el administrador del sistema. El **diagrama de clases** que representa este módulo se muestra a continuación

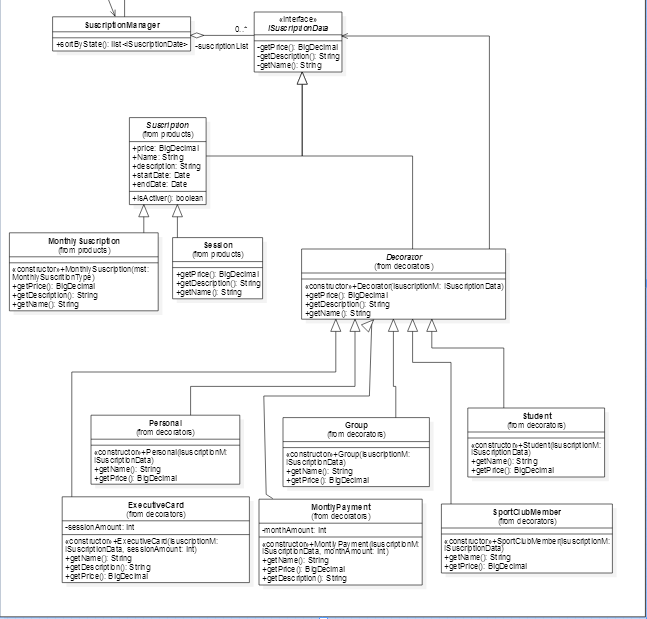


Figura. Diagrama de clases – Módulo Suscripciones. Fuente: Autores

En este módulo se hizo uso del patrón de diseño estructural *Decorator*. La aplicación de este patrón es debido a que El sistema parte de suscripciones base de tipo mensualidad y sesión con unos atributos base específicos, sin embargo, de acuerdo a determinados planes y tipos servicios adquiridos, a estas suscripciones se les pueden agregar cualidades específicas robusteciéndolas, “decorándolas” de manera dinámica, creando una gran cantidad de combinaciones, escenario que es perfectamente posible respondiendo a la lógica del negocio. De igual manera, en el sistema, evita que sobre este módulo se creen jerarquías de clases complejas que dificulten la consecución del mismo objetivo.

Las operaciones en este módulo se ilustran a un alto nivel mediante el siguiente **diagrama de secuencia**

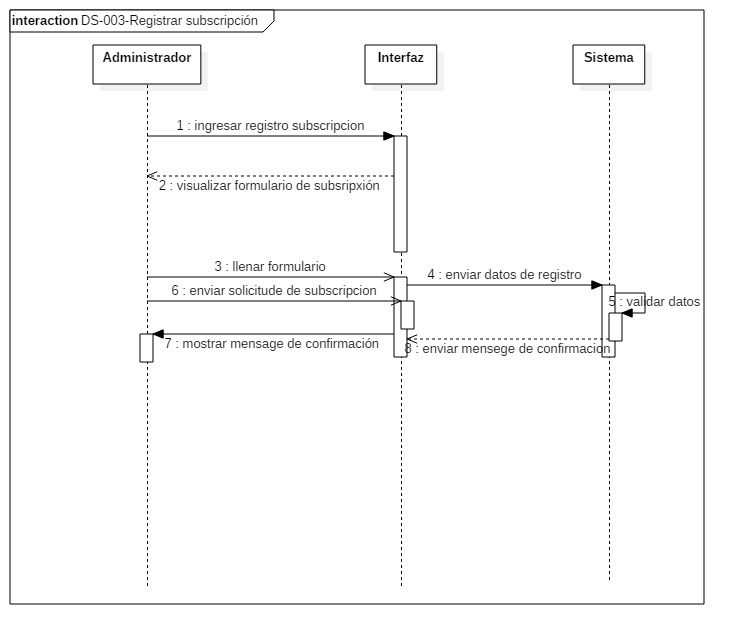


Figura 8. Diagrama de secuencia – Módulo Suscripciones. Fuente: Autores

[FALTA EDITAR NOMBRES Y AGREGAR OTROS PROCEDIMIENTOS]

A nivel de comunicación entre clases en este módulo para el registro de suscripciones, se tiene el siguiente diagrama

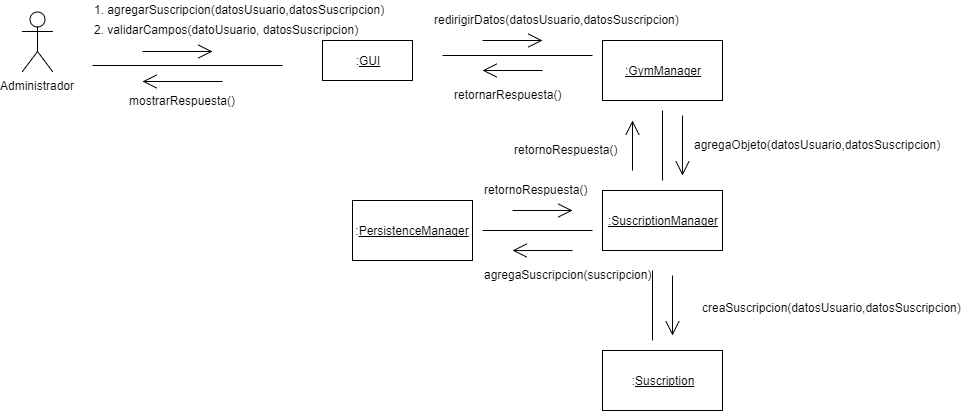


Figura 8. Diagrama de comunicación – Registro Suscripción - Módulo Suscripciones. Fuente: Autores

## Modelo de Datos

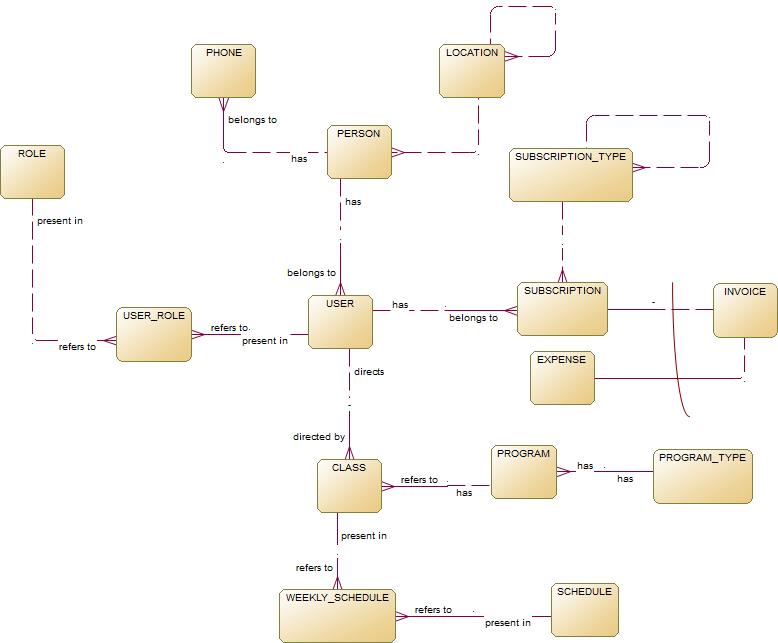


Figura. Modelo conceptual de datos. Fuente: Autores

## Acotaciones Generales

* El patrón de diseño *Template* es sutilmente usado en todos los módulos, puesto que es la base para su estructuración de acuerdo a los requisitos del sistema así como para facilitar su escalabilidad y reutilización.
* El patrón *Template* se ve representado más que todo en el módulo de usuarios del sistema. Si bien todos los usuarios poseen características similares, cada uno interactúa de manera distinta en el sistema y tiene diferentes responsabilidades sobre el mismo. A ello aducen los roles que se le asignan al usuario en el momento en el que ingresa al registro del sistema y que le permiten o no participar de manera activa o pasiva en los diferentes módulos. Otro caso particular se presenta en la administración de dichos módulos, los cuales tienen responsabilidades específicas y tratan de cosas totalmente distas, pero que en general comparten acciones similares en lo que respecta a la manipulación de datos (CRUD).
* La aplicación de este patrón se ve representada en la jerarquía superior del sistema. Dado que se mantienen una serie de módulos específicos los cuales son administrados por unos Managers, se ve la necesidad de utilizar una fachada mediante la cual se puedan realizar llamadas a cada uno de los managers sin la necesidad de crear instancias separadas en tiempo de ejecución. Asimismo, de forma parametrizada, se permite el direccionamiento de las acciones a cada uno de los managers de manera dinámica mediante la fachada, ya que esta tiene sus referencias. A nivel del sistema, permite que una clase tenga el control total en cada uno de sus módulos, así como elevar la abstracción del proceso de realizar acciones simples (CRUD) sin tener que preocuparse por lo que sucede en niveles inferiores de la jerarquía.

# Vista de Desarrollo o Despliegue

Esta vista se encarga de mostrar el sistema desde la perspectiva del desarrollador en una forma estática. Subdivide el sistema en componentes y analiza las relaciones que hay entre dichos componentes. A diferencia de la vista lógica, el alcance de los diagramas mostrados a continuación no se limita a los componentes propios del negocio sino que también aborda subsistemas o componentes externos con los que se interactúa.

Esta perspectiva es visible mediante el diagrama de componentes del sistema.

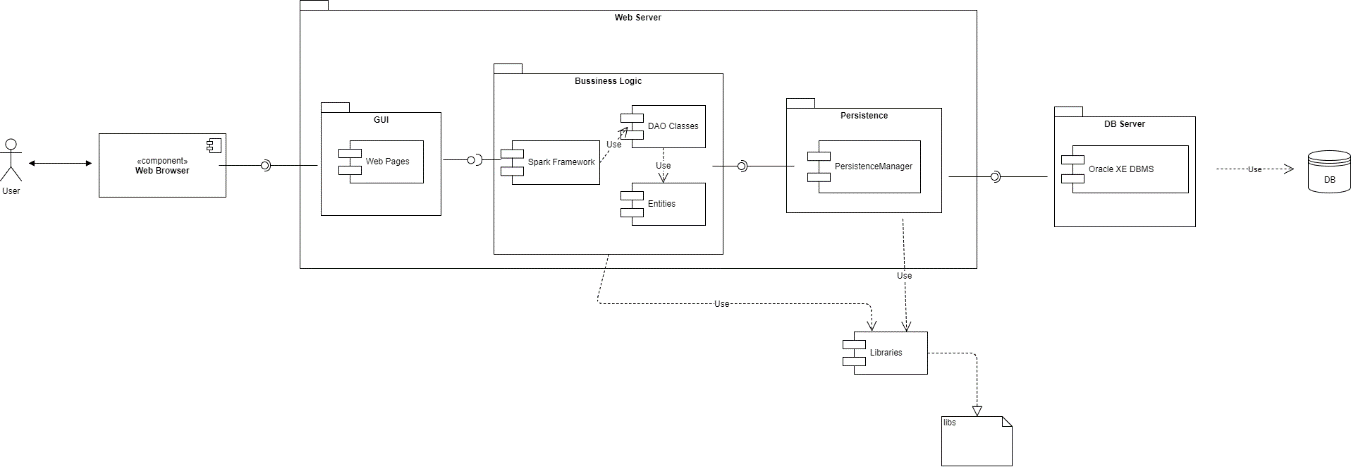


Figura. Diagrama de componentes – Arquitectura C-S & MVC. Fuente: Autores

El anterior diagrama permite esbozar la estructura básica de una **arquitectura Cliente-Servidor** en conjunción con una **arquitectura MVC** a alto nivel. El ***core*** del sistema radica en el componente del servidor web, donde recepciona la información a la GUI a través de protocolo ***HTTP***, de este componente a la lógica del negocio a través de servicios web ***RESTful***, este último al DBMS a través de sus respectivos ***drivers*** y el ***framework Hibernate****.*

En complemento a los diagramas de componentes mostrados, una perspectiva a un mayor nivel, eliminando la minucia, se encuentra el **diagrama de paquetes**, este diagrama permite obtener un panorama más amplio en la estructura del sistema en unidades lógicas mayores denominadas paquetes. Por esto último es que es posible identificar partes del sistema externas a la capa de lógica del negocio, mostrando tanto el componente visual que comprende la trinidad **HTML5-CSS3-JavaScript** y librerías JS anexas, como el componente de persistencia de datos, compuesto de ***DAOs*** que interactúan por medio de **frameworks** con el **DBMS**



Figura. Diagrama de paquetes – Paquete UI. Fuente: Autores

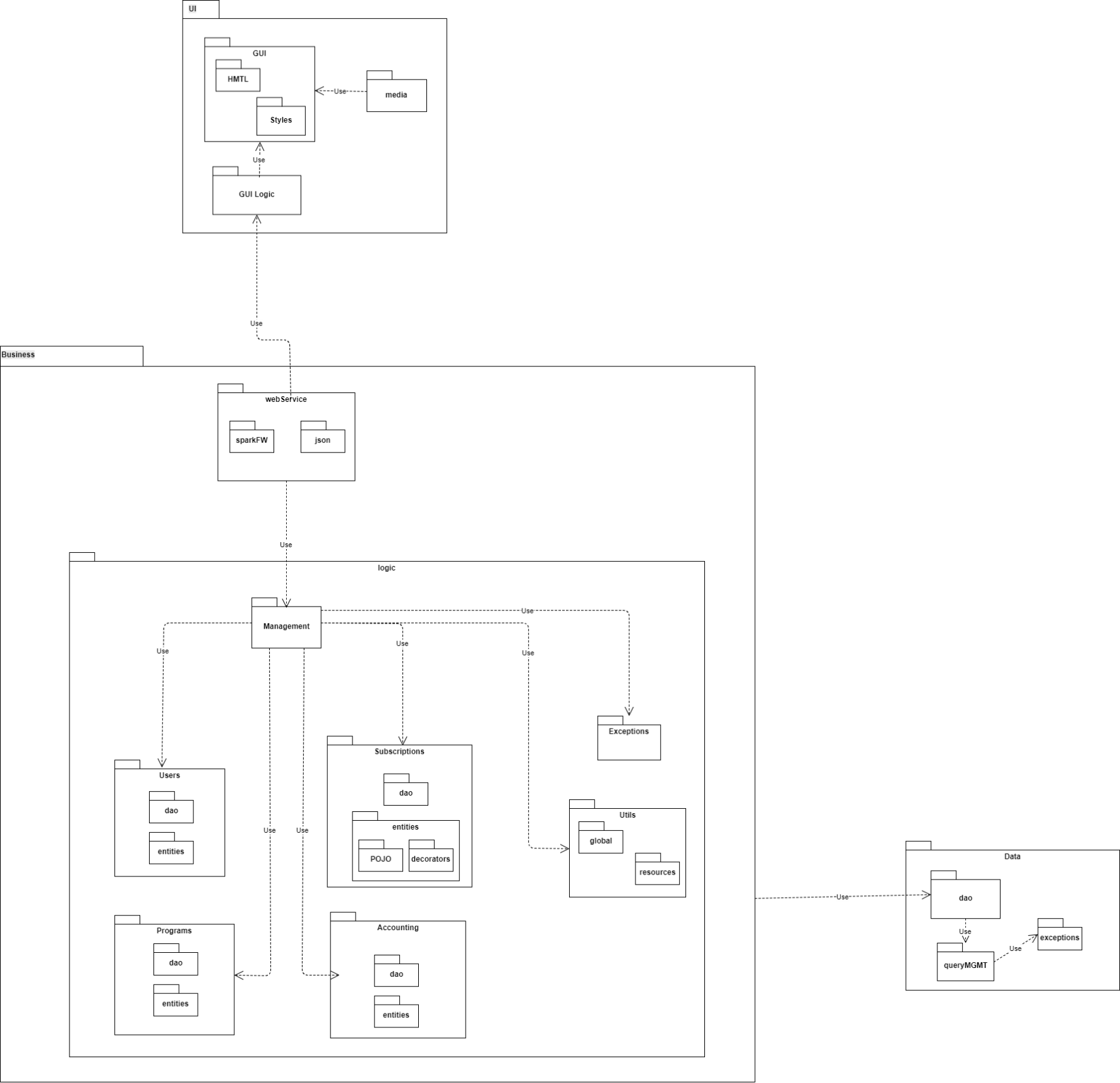


Figura. Diagrama de paquetes – Paquete Business. Fuente: Autores



Figura. Diagrama de paquetes – Paquete Data. Fuente: Autores

**Referencias**

[1] P. Kruchten, «Architectural blueprints—The “4 1” view model of software architecture», *IEEE Software*, vol. 12, págs. 42-50.

[2] <https://jarroba.com/modelo-41-vistas-de-kruchten-para-dummies/>

[3] <http://docente.ucol.mx/almoradi/public_html/Respaldo/resumen3.htm>

[4] <https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kontio05-2>