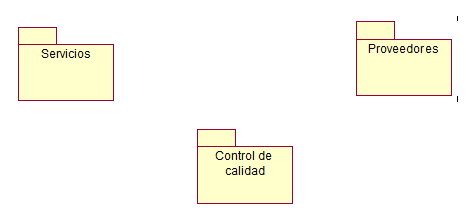
**ESTRUCTURA DE DISEÑO**

[Describe the design from the highest level. This is commonly done with a diagram that shows a layered architecture.]

**Sub-Sistemas**

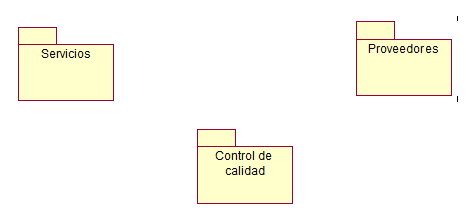
**Proveedores**

El sub-sistema proveedores, se encarga del registro, búsqueda y modificación de proveedores para el acopio respectivo de leche.



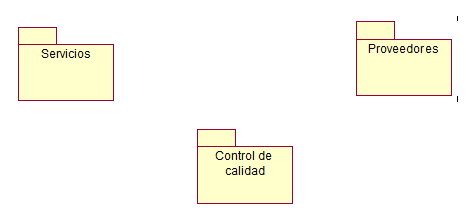
**Control de calidad**

El subsistema control de calidad se encarga del registro de incidencias en el acopio de la leche, para medir su calidad en la recepción por cada proveedor, comparando con un indicador, la cual nos informará si tiene la pureza aceptada o es un producto con agua.



**Servicios**

El subsistema servicios, se encarga del registro de pagos y prestamos realizados a los proveedores, para lo cual se tiene en cuenta la liquidación recibida por gloria o nestle, y las incidencias presentadas en el control de calidad a su producto.



**Patrones**

**[XVC]**

El patrón desarrollado se basa en el patrón de diseño MVC , que se rige bajos sus fundamentos y sus estándares que usa, el patrón XVC deslinda la capa modelo para acoplar una capa de implementación de lógica, que puede ser determinado por el programador denotando así en la capa X, mas patrones de desarrollo o mejora en el.

**PSR-4:**

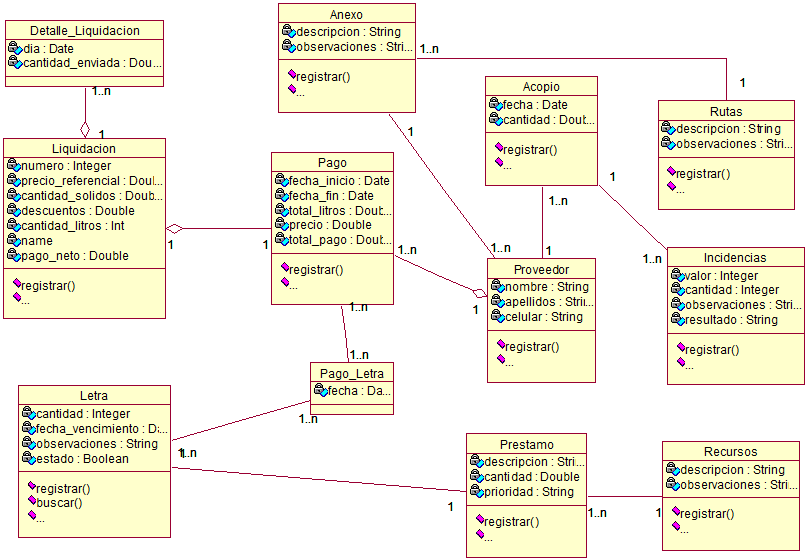
Este PSR describe una especificación para carga automática de clases de rutas de archivos. Es totalmente interoperable, y se puede utilizar en adición a cualquier otra especificación de carga automática, incluyendo PSR-0 . Este PSR también describe dónde colocar los archivos que se cargan automáticamente de acuerdo a la especificación.

Ejemplo:

\ <NameSpaceName> (\ <SubNamespaceNames>) \* \ <NombreClase>

**Repository**

El patrón repositorio o repository pattern está íntimamente relacionado con el acceso a datos y nos permite tener una abstracción de la implementación de acceso a datos en nuestras aplicaciones, de modo que nuestra lógica de negocio no conozca ni esté acoplada a la fuente de datos. En pocas palabras esto quiere decir que el repositorio actúa como un intermediario entre nuestra lógica de negocio y nuestra lógica de acceso a datos para que se centralice en un solo punto, y de esta forma se logre evitar redundancia de código. Y como dije antes al ser una abstracción del acceso a datos nos permite desacoplar y testear de una forma más sencilla nuestro código, ya que al estar desacoplado podemos generar pruebas unitarias con mayor facilidad. Adicional al estar centralizado en un solo punto de acceso, podemos reutilizar nuestra lógica de acceso a datos desde cualquier cliente, es decir desde otras aplicaciones que tengamos en nuestro entorno corporativo, incluso desde una app móvil si exponemos los repositorios a través de Apis por ejemplo.



[For other scenarios that must be described to convey an appropriate amount of information about how the requirement behavior will be realized, describe how instances of the design elements collaborate to realize the requirement. When using UML, you can do this with collaboration diagrams (sequence or communication).]