Sistema de Gestión

Este documento detalla la arquitectura de la aplicación creada para la resolución del trabajo práctico. La misma requiere crear una aplicación para la gestión de trámites y consultas de la materia.

Se detallan las siguientes vistas del modelo 4 + 1:

* [Casos de uso](#_Vista_de_Casos)
* [Desarrollo](#_Vista_de_Desarrollo)
* [Lógica](#_Vista_Lógica)
* [Proceso](#_Vista_de_procesos)
* [Física](#_Vista_física)

Como la idea es proveer un panorama amplio de la arquitectura, los diagramas y explicaciones no tienen un gran nivel de detalle, sino que muestran información general sobre la arquitectura y el diseño.

# Vista de Casos de Uso

En esta sección se mencionan los casos de uso principales de la aplicación.

1. Configurar cuentas de e-mail de la materia.
2. Administrar datos de la materia (crear, modificar y eliminar).
3. Procesamiento de e-mails bajo determinadas reglas:
   * Alta de alumnos en la base de datos y/o google spreadsheat.
   * Persistencia de archivos adjuntos en e-mails.
   * Responder ante e-mails con formato desconocido.
   * Generar nuevos temas de discusión.
   * Generar el alta de grupos en la base de datos.
4. Consultar temas de discusión.
5. Asignación de temas de discusión.
6. Monitoreo de estado de los temas de discusión.
7. Ver estadísticas (alumnos inscriptos, grupos asignados, TPs recibidos, e-mails recibidos, estados de temas de discusión, número de e-mails que no cumplen con las reglas).

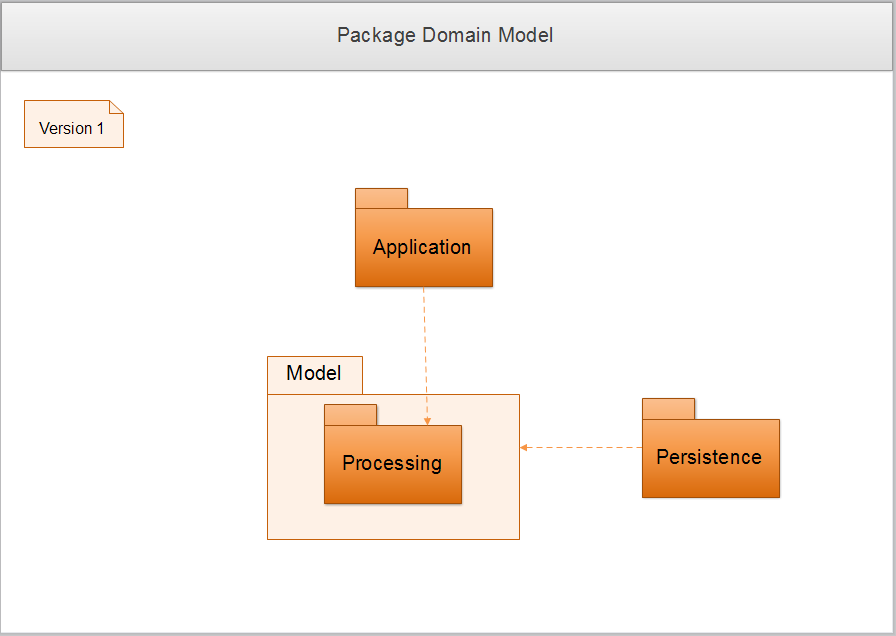
# Vista de Desarrollo

Para el desarrollo de la aplicación se utilizara el lenguaje de programación C#, posiblemente con código embebido de Phyton para la lectura de e-mails. Esta decisión se baso en parte, la facilidad que tiene Phyton para la lectura de e-mails. El equipo de desarrollo eligió utilizar C# ya que considero que de esta forma el desarrollo de la aplicación tomaría menos tiempo, y la calidad sería mayor.

## Diagrama de paquetes

Los siguientes son los diagramas de paquetes resultantes pensados para la aplicación (sin incluir los paquetes de pruebas).

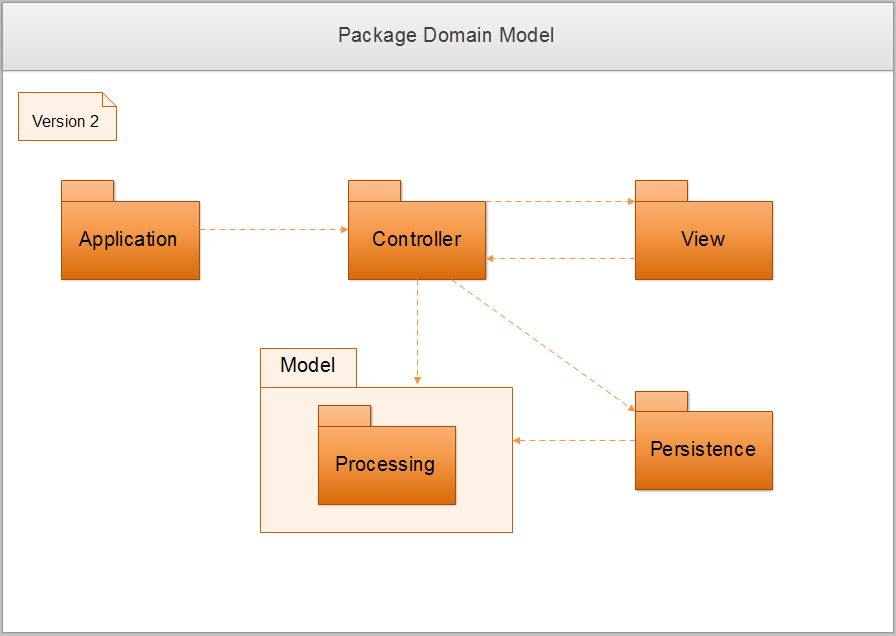
En una primera instancia se realizara una aplicación con vista por consola, este es el diagrama de paquetes:



El contenido de cada paquete se explica a continuación:

* **Models:** Contiene a las clases que representan al dominio de la aplicación.
* **Application:** Contiene clases necesarias para iniciar la aplicación (como el Bootstrapper).
* **Persistence:** Contiene las clases necesarias para guardar los datos generados por el procesamiento de e-mails y recuperarlos.
* **Processing:** Contiene las clases que realizan validaciones sobre los e-mails a partir de reglas y aplica las acciones a los mismos.

En una segunda instancia se realizara una interfaz grafica para la consulta y las configuraciones. Este es el diagrama de paquetes:



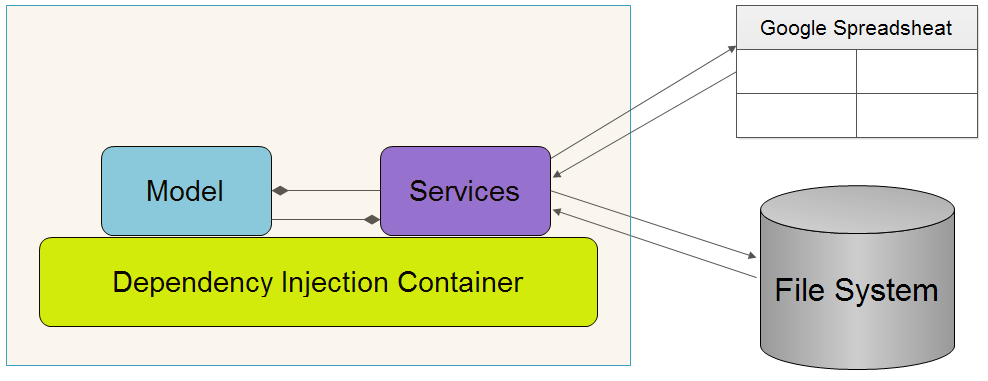
El contenido de cada paquete se explica a continuación:

* **View:** Contiene las interfaces de las vistas y la implementación de las mismas.
* **Controllers:** Contiene a la implementación de los controladores para cada vista y las interfaces de los mismos.
* **Models:** Contiene a las clases que representan al dominio de la aplicación.
* **Application:** Contiene clases necesarias para iniciar la aplicación, como la ventana principal (que se compone con las vistas) y el Bootstrapper.
* **Persistence:** Contiene las clases necesarias para guardar los datos generados por el procesamiento de e-mails y recuperarlos.
* **Processing:** Contiene las clases que realizan validaciones sobre los e-mails a partir de reglas y aplica las acciones a los mismos.

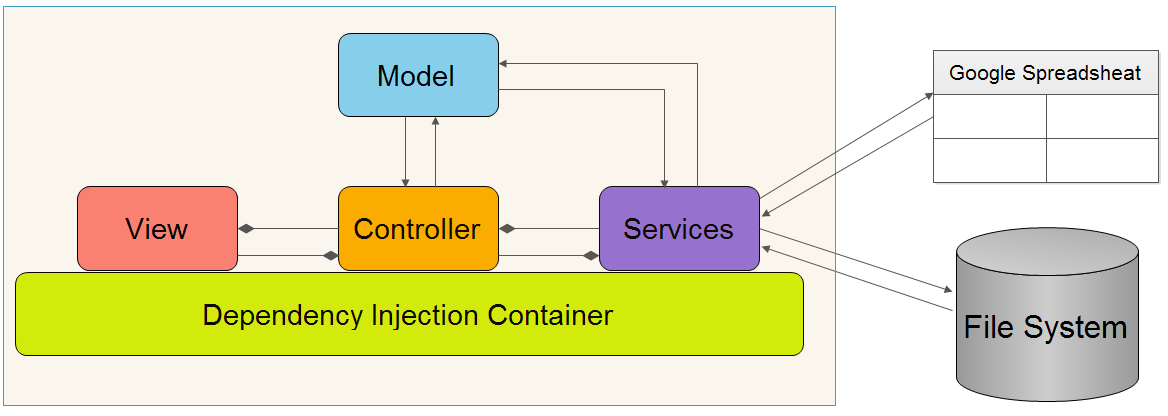
# Vista Lógica

Las siguientes figuras proveen una visualización abstracta del diseño en capas de la aplicación para ambas versiones.

Versión 1:



Versión 2:



Desde el comienzo del proceso de desarrollo se determino utilizar el patrón [Model View Controller](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) para la segunda versión donde proveeremos una GUI para las consultas y administración de datos de la materia (la primer versión resolverá esto a través de la consola de comandos). Las razones para ello fueron:

* El patrón permite abstraer a lógica visual y la de control, simplificando cada componente y clarificando sus responsabilidades.
* Mediante el desarrollo orientado a interfaces en vez de implementaciones concretas permite realizar pruebas unitarias con "[objetos mock](http://en.wikipedia.org/wiki/Mock_object)".
* Comunicación desacoplada, comunicando a la vista y controlador utilizando interfaces o el patrón [Observer](http://en.wikipedia.org/wiki/Observer_pattern).
* Reutilización de vistas, como el caso de la vista de Atributos, la cuál se permitió desarrollar una única vista que sirva para administrar los atributos de las entidades y relaciones.

Adicionalmente, como uno de los objetivos del desarrollo era lograr el desacoplamiento de componentes, cada clase recibe en su constructor interfaces, no implementaciones abstractas. Para facilitar la construcción de estos objetos y poder modificar de manera simple y rápida la implementación a utilizar, se decidió utilizar un [DependencyInjectionContainer](http://en.wikipedia.org/wiki/Dependency_injection), [PicoContainer](http://picocontainer.codehaus.org/). El mismo se configura mediante un [Bootstrapper](http://en.wikipedia.org/wiki/Bootstrapping).

# Vista de procesos

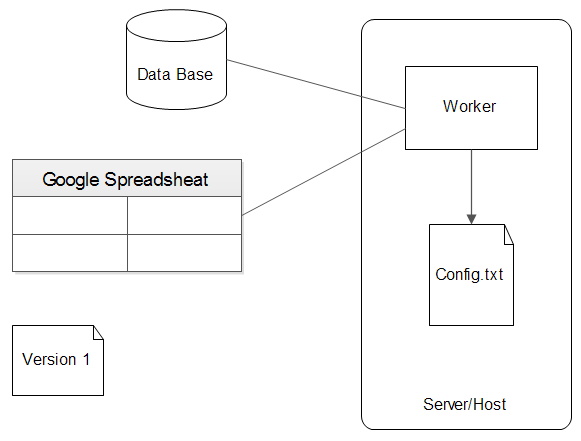
La aplicación cuenta con:

* Un thread en un loop infinito que corre en un servidor. Este thread se encarga de levantar, procesar y persistir los resultados.
* Al realizar consultas se ejecutara otro thread en un host que se encargara de ejecutar las acciones indicadas por el usuario a través de la interfaz provista obteniendo los datos de la base de datos del servidor.

# Vista física

La aplicación se distribuirá de la siguiente forma:

Versión 1:



Versión 2:

