Universidad ORT Uruguay

Facultad de Ingeniería

OBLIGATORIO 1

Entregado como requisito para la obtención del crédito de la materia Arquitectura de Software

Integrantes: Mauro Caruso - 174922.

Ignacio Dolan - 165442

Marcos Guichón - 173252

Docentes: Gaston Musques

Tommas Piaggio.

2017

Contenido

[1. Introducción 2](#_Toc486542832)

[1.1. Propósito 2](#_Toc486542833)

[1. Antecedentes 2](#_Toc486542834)

[2.1. Propósito del sistema 2](#_Toc486542835)

[2.2. Requerimientos funcionales 2](#_Toc486542836)

[2.3. Requerimientos no funcionales 4](#_Toc486542837)

[2.4. Restricciones 6](#_Toc486542838)

[3. Documentación de la arquitectura 7](#_Toc486542839)

[3.1. Vista de módulos 7](#_Toc486542840)

[3.1.1 Vista de descomposición 7](#_Toc486542841)

[3.1.1.1 Representación primaria 7](#_Toc486542842)

[3.1.1.2 Decisiones de diseño 7](#_Toc486542843)

[3.1.1.3 Guía de variabilidad 8](#_Toc486542844)

[3.1.2 Vista de usos 8](#_Toc486542845)

[3.1.2.1 Representación primaria 8](#_Toc486542846)

[3.1.2.2 Decisiones de diseño 8](#_Toc486542847)

[3.1.2.3 Guía de variabilidad 15](#_Toc486542848)

[3.1.3. Vista de Componentes y conectores 16](#_Toc486542849)

[3.1.3.1 Representación primaria 16](#_Toc486542850)

[3.1.4. Vista de despliegue 17](#_Toc486542851)

# Introducción

El documento tiene como objetivo la presentación de la arquitectura general del sistema EnviosYa.

El mismo se puede divide en 2 secciones fundamentales, una que explica de que trata el sistema, sus requerimientos funcionales y no funcionales.

La otra sección presenta diversos aspectos importantes del sistema de forma tal que se reflejan muchas de las decisiones importantes tomadas en el diseño del mismo a través de distintas vistas como son las vistas de módulos, componentes y conectores y asignación.

# Propósito

El propósito del presente documento es proveer una especificación completa de la arquitectura de EnviosYa.

# Antecedentes

# Propósito del sistema

EnviosYa es una compañía ficticia que se encarga de realizar envíos.

El sistema posee 3 actores principales y se detallan a continuación:

* Cliente: Es quien realiza la solicitud de un envió
* Cadete: Es quien se encarga de realizar el pedido
* Destinatario: Es quien recibe él envió

# Requerimientos funcionales

|  |  |  |
| --- | --- | --- |
| **Requerimiento** | **Descripción** | **Actor** |
| 1- Mantenimiento de Cadetes y Vehículos | El sistema deberá permitir registrar la información de los cadetes (CI, Nombre, Apellido, Correo Electrónico), sus Vehículos (Matrícula, Descripción) y asociarlos. También se deberá poder modificar y borrar a los cadetes y sus vehículos. Las cédulas de los cadetes no pueden ser repetidas y se deberá validar que el correo electrónico sea válido. | Usuario |
| 2- Mantenimiento de Clientes | El sistema deberá permitir registrar la información de los clientes (CI, Nombre, Apellido, Correo Electrónico, información sobre medios de pago). También se deberá poder modificar los datos y borrar a los clientes. Las cédulas no pueden ser repetidas, y se debe validar que el correo electrónico sea válido. | Usuario |
| 3 - Registrar una solicitud de envío | El sistema deberá permitir que los clientes puedan solicitar envíos a destinatarios. Para ello deberán poder crear un Envío (ID, descripción, Ubicación del Cliente que Envía, Ubicación Cliente Destinatario, Forma de pago, comisión, Foto Paquete). El sistema debe verificar que el cliente y destinatario sean usuarios registrados y responder con los datos de los 4 cadetes más cercanos que puedan recoger el envío. El cliente selecciona el cadete que desea y el sistema le asigna el envío notificándole los datos del envió que debe realizar.  NOTA: para calcular el costo del envío en base a la foto del paquete, se deberá consumir un servicio REST provisto por los profesores a través de Aulas en los próximos días. | Usuario |
| 4 - Registrar recepción de envío | El destinatario confirma la recepción del envío y el sistema notifica a ambos usuarios los datos del envío, de su recepción y la información sobre el cadete que realizó el traslado. La notificación debe ser por correo electrónico, y el correo debe incluir un link para que ambos usuarios puedan calificar el servicio y al cadete que realizó el traslado. | Sistema |
| 5 - Calificar el servicio y el Cadete | Utilizando el link recibido por correo ambos usuarios deben poder calificar al servicio y al cadete. Para ambos casos el usuario puede valorar utilizando una escala de 1 a 5 estrellas y agregar un comentario. | Usuario |
| 6 - Listar los envíos pendientes de calificación de un cliente | Luego de loguearse al sistema, el cliente accede a la lista de envíos realizados.  El sistema obtiene del módulo Envíos la lista de envíos del cliente, ordenados cronológicamente de forma descendente.  El sistema obtiene del módulo Revisiones la lista de calificaciones en estado “approved” para esos envíos. El sistema compara los resultados anteriores y retorna al usuario la lista de envíos pendientes de calificación. | Usuario |
| 7 - Listar todas las calificaciones de un Cadete | Luego de loguearse al sistema, el usuario accede a consultar un envío realizado y para este consulta la información del Cadete que lo realizó.  El sistema obtiene del módulo Reviews la lista de calificaciones en estado “approved” para el Cadete, ordenadas cronológicamente de forma descendente y la retorna al usuario. | Usuario |
| 8 - Rechazar manualmente una calificación inapropiada a un Cadete | Solo usuarios administradores pueden utilizar esta funcionalidad.  El administrador obtiene la lista de calificaciones de un Cadete (RF7).  Selecciona una calificación y la opción “rechazar”.  El sistema marca la calificación con estado “rejected”. | Usuario administrador |
| 9 - Consultar las calificaciones no aprobadas | Solo usuarios administradores pueden utilizar esta funcionalidad.  El administrador accede a la opción de consultar calificaciones (no visible para otros usuarios). Se le presenta la posibilidad de filtrar su consulta por estado e intervalo de fechas de registro de las calificaciones.  El sistema obtiene del módulo de Reviews la lista de calificaciones, aplicando los filtros seleccionados y la retorna al usuario. | Usuario administrador |

# Requerimientos no funcionales

|  |  |  |
| --- | --- | --- |
| **Requerimiento** | **Descripción** | **Atributos de calidad** |
| 1 - Configuración del proceso de creación de una Review | La estructura del proceso debe permitir agregar, quitar o intercambiar pasos con el menor impacto posible tanto en tiempo de desarrollo como de despliegue; así como soportar el reintento de los pasos que no lleguen a cumplirse en caso de que el proceso se vea interrumpido por cualquier motivo (por ejemplo fallas propias o de sistemas externos con los que se interopera). | Modificabilidad |
| 2 - Tiempo de respuesta de creación de una Review | El tiempo de respuesta promedio de todos los módulos de back-end considerado aceptable para todas las peticiones que se realizan desde las aplicaciones “cliente” (apps móviles, clientes web) es de 200ms. En particular el proceso de creación de una Review no debe superar los 350ms. | Performance |
| 3 - Usuarios registrados | Todas las funcionalidades descritas sólo pueden ser utilizadas por usuarios debidamente autenticados. Por razones estratégicas del negocio, los usuarios solo pueden registrarse en el sistema utilizando sus cuentas de Gmail, Twitter o Facebook, por lo tanto la autenticación de los usuarios se debe realizar contra esos proveedores. Es decir que para acceder a su cuenta en EnviosYa, el usuario debe loguearse a través de alguno de estos proveedores. | Seguridad |
| 4 - Gestión de errores y fallas | El sistema debe proveer suficiente información, de alguna forma, que permita conocer el detalle de las tareas que realiza. En particular, en el caso de ocurrir una falla o cualquier tipo de error, es imprescindible que el sistema provea toda la información necesaria que permita a los administradores hacer un diagnóstico rápido y preciso sobre las causas. | Disponibilidad |
| 5 - Recuperación y Fallas | Sería deseable que la información producida por el sistema pueda ser reutilizada de forma automática en aquellos casos en que es posible reconstruir alguna tarea que no haya finalizado debidamente (por error, falla o intervención de un administrador). | Disponibilidad |
| 6 - Independencia de “consultas” y “escrituras” | Teniendo en cuenta que se trata de un sistema que soporta una gran demanda de los usuarios, donde la relación entre las operaciones de “consulta de datos” e “ingreso de datos” son del orden de 10 a 1 (es decir, por cada ingreso de datos se realizan 10 consultas), se desea construir la aplicación de forma tal que las modificaciones que afecten las funcionalidades de ingreso de datos (como la creación de una Review) no impliquen la necesidad de re-desplegar los componentes que implementan las funcionalidades de consultas. Es decir, es deseable que los componentes que implementan operaciones de ingreso de datos puedan gestionarse y desplegarse de forma independiente de los componentes que implementan operaciones de consulta de datos. | Performance |
| 7 - Protección de los datos de los clientes. | Los datos de los clientes deben protegerse de accesos no autorizados. Se debe hacer especial énfasis en proteger la información de los medios de pago asociados a los clientes. | Seguridad |
| 8 - Información de auditoría. | El sistema debe registrar información que permita realizar auditorías de acceso de forma de identificar los accesos autorizados y no autorizados al sistema. | Seguridad |
| 9 - Tiempo de respuesta de creación de una solicitud de envío | El tiempo de respuesta promedio de todos los módulos de back-end considerado aceptable para todas las peticiones que se realizan desde las aplicaciones “cliente” (apps móviles, clientes web) es de 200ms. En particular el proceso de creación de una solicitud no debe superar los 200ms para aquellos usuarios que hayan tenido alguna actividad en el sistema en los últimos 15 días. Para aquellos usuarios que no hayan tenido actividad por más de 15 días el tiempo de respuesta no debe superar los 500ms. | Performance |
| 10 - Modificación del cálculo de costos de los envíos. | A futuro se debe poder modificar el mecanismo de cálculo de costo del envío con el menor impacto posible en los módulos del sistema y con el redespliegue del menor número de componentes posibles. | Modificabilidad |

# Restricciones

* La implementación del backend debe desarrollarse en JEE.
* Todas las operaciones deben estar disponibles a través de servicios web.
* La persistencia debe realizarse utilizando JPA.
* Utilizar tecnologías EJB y JMS.
* El servidor de aplicaciones utilizado por el sistema será Glassfish 4.1.1
* El lenguaje de programación a utilizar será JAVA.
* Respetar estándares de codificación de JAVA.
* La API que expone el backend debe ser REST.

# Documentación de la arquitectura

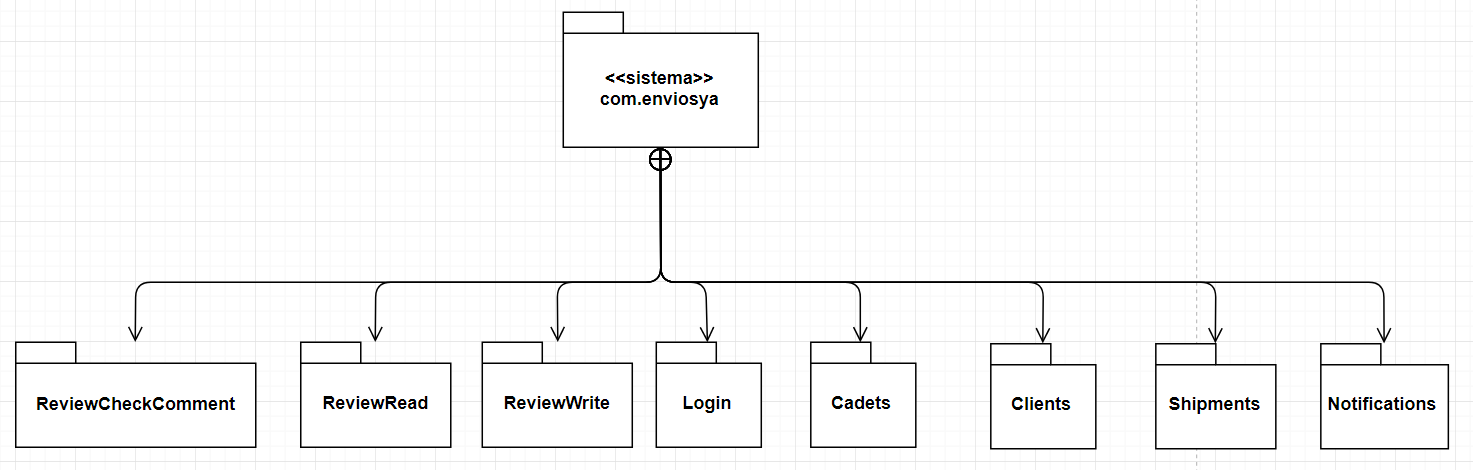
A continuación se refleja el diseño de la arquitectura del sistema por medio de distintas vistas (Descomposición, Módulos, Componentes y conectores)

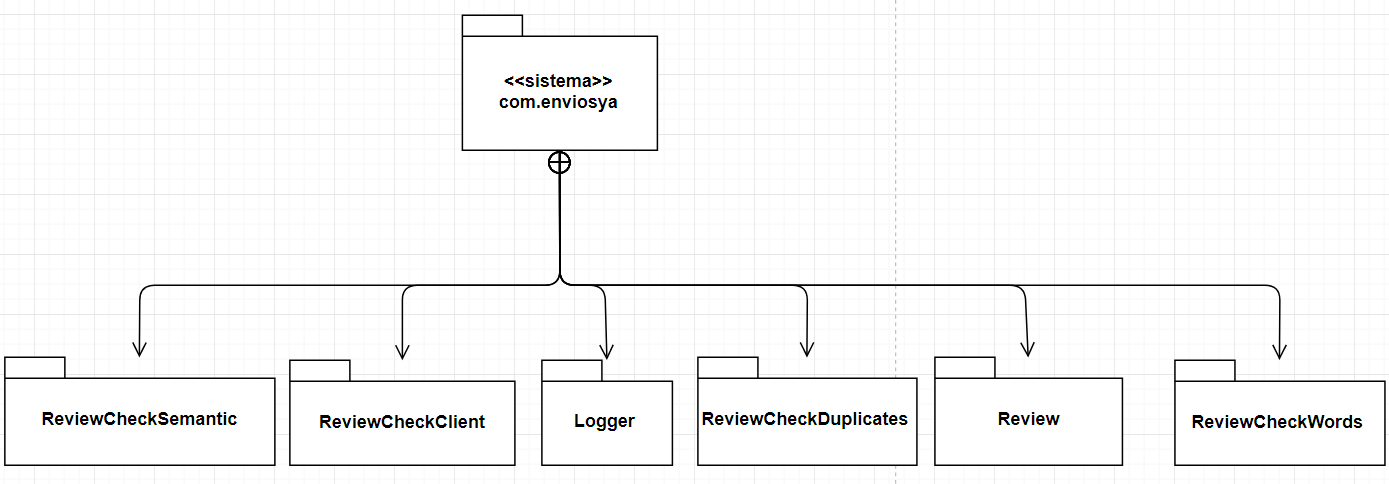
# 3.1. Vista de módulos

# 3.1.1 Vista de descomposición

# 3.1.1.1 Representación primaria

En este diagrama de descomposición se reflejan los paquetes que componen el sistema en su totalidad.





# 3.1.1.2 Decisiones de diseño

La decisión de diseño general se realizó buscando que en un futuro el sistema sea altamente modificable, es decir, que a la hora de añadir, modificar o eliminar funcionalidades del sistema, este reciba el menor impacto posible. Para esto se descompuso la solución en módulos buscando obtener una alta cohesión y un bajo acoplamiento entre ellos.

Esta descomposición se realizó de este modo debido a las funcionalidades que el sistema debe proveer, por esa razón, los módulos están compuestos por conjuntos de clases o paquetes que tienen responsabilidades en común.

# 3.1.1.3 Guía de variabilidad

Teniendo en cuenta el tamaño inicial del sistema, se plantea la solución que se muestra en el la vista de descomposición pero se determina que cada nuevo Feature será agregado como un módulo nuevo.

# 3.1.2 Vista de usos

# 3.1.2.1 Representación primaria

En el siguiente diagrama se muestra cómo interactúan los módulos entre sí y permite determinar cómo impactan los cambios que se realicen.

La convención para los nombres es com.enviosya.

USOS GENERALES

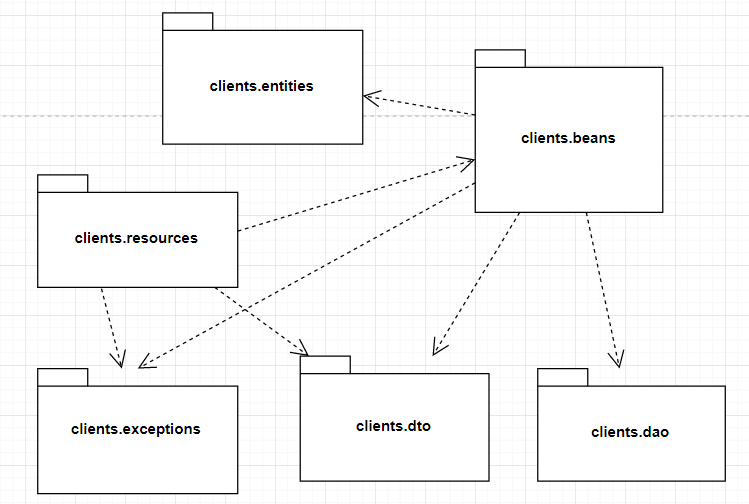
# 3.1.2.2 Decisiones de diseño

En el módulo com.enviosya.cadets se gestionan los cadetes y sus vehículos. En este módulo se permite el ingreso de cadetes, vehículos y asociarlos entre sí.

El módulo com.enviosya.clients permite el ingreso de clientes y sus medios de pagos.

El modulo com.enviosya.shipments es el encargado de crear los envíos.

Módulo Clients:



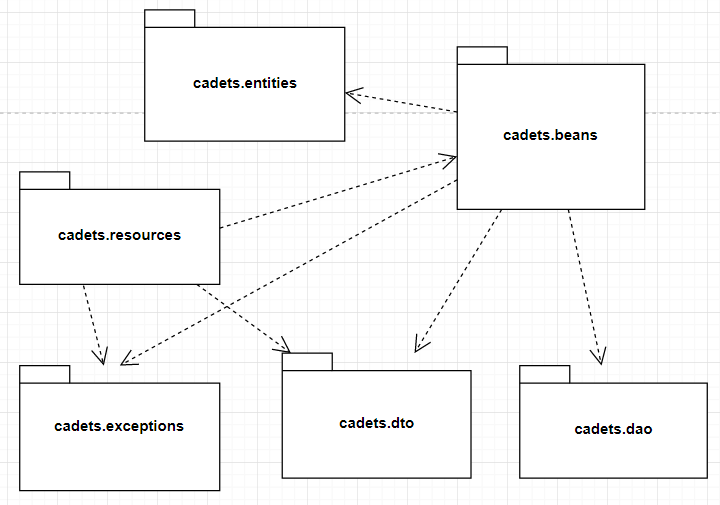
Se realiza la implementación de DTOs o Data transfer object, de manera de evitar que los servicios queden acoplados con el dominio del sistema. Este tipo de objetos presentan características que favorecen la performance ya que son más livianos.

También se utilizan Stateless sessionBeans como por ejemplo el ClientBean, estos se encuentran en el EJB (Enterprise Java Bean) y son los encargados de manejar la lógica de negocios de la aplicación y exponen una interfaz local.

Los paquetes entities contienen clases de tipo Entity como por ejemplo ClientEntity, como los beans, estas también se encuentran en el EJB y son las encargadas de manejar el acceso a datos. Estas se conectan a la base mediante JDBC. Como el ejemplo brindado hace referencia a la clase ClientEntity, el encargado de consumirla siguiendo el hilo del ejemplo seria el ClientBean.

Por otra parte, tenemos los resources, estos se encuentran en el WAR y exponen operaciones a través de web services como por ejemplo ClientResources.

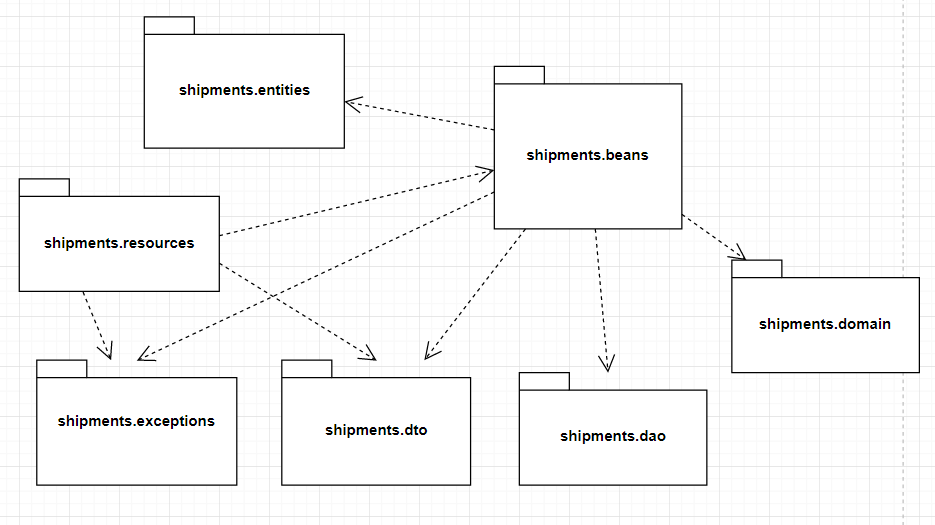
Módulo Cadets:



El diseño de esta aplicación es bastante similar al diseño utilizado para el módulo de Clients por los componentes que la conforman y cómo interactúan entre sí.

En el caso de la aplicación Cadets, ésta se encarga también de gestionar vehículos, es por esto dentro del paquete cadets.bean tenemos dos Session Beans, uno para cadetes y otro para vehículo, esto favorece a la performance ya que se cuenta con varios componentes de menores dimensiones.

Módulo shipments:

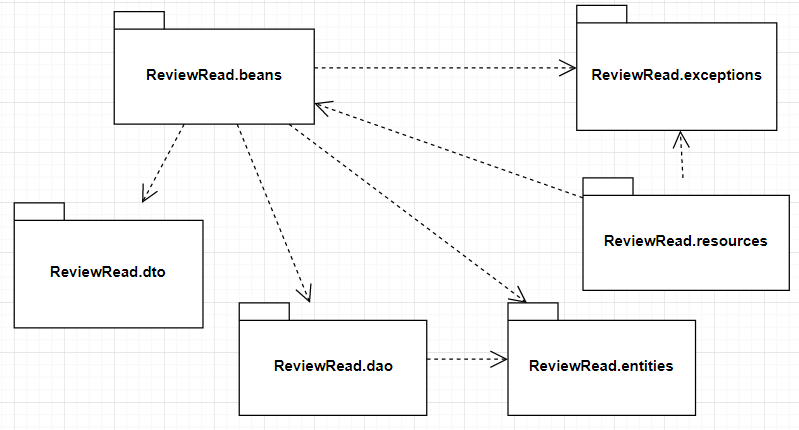


El diseño de esta aplicación es bastante similar al diseño utilizado para el módulo de Clients por los componentes que la conforman y cómo interactúan entre sí.

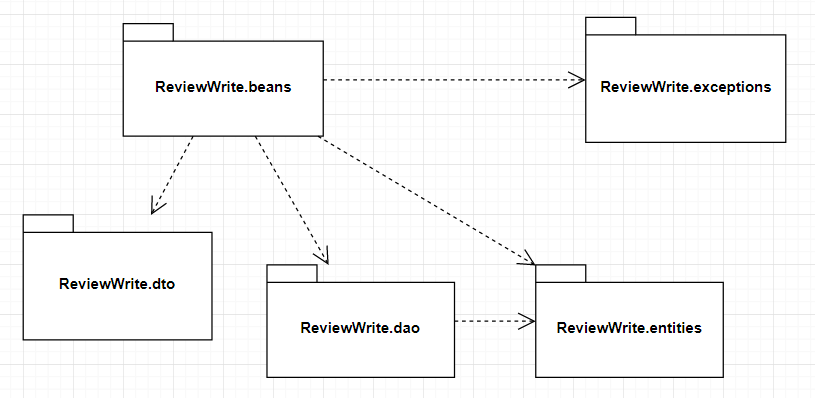
En el caso de la aplicación Shipments, ésta se encarga él envió en sí, esto implica que gestiona 2 clientes (el que solicita el pedido y el que lo recibe) y un cadete que es el encargado de enviar el paquete. Dentro del paquete Shipments.dto tenemos dos dtos, uno llamado InitialShipmentDTO que es el encargado de crear él envió junto con la lista de 4 cadetes disponibles. Por otro lado tenemos el ShipmentDTO que es el que arma el envió con el cadete encargado del mismo.

Dentro de este módulo también podemos apreciar el paquete Shipments.domain, este paquete contiene las clases Package (clase abstracta) y PackageNormal el cual extiende de la antes mencionada. PackageNormal se encarga de implementar el método getCosto que determina como se calcula el costo del paquete a enviar. Esto permite poder agregar más clases que extiendan de Package y puedan sobrescribir el método para agregar nuevas formas de calcular el costo del paquete favoreciendo la modificabilidad.

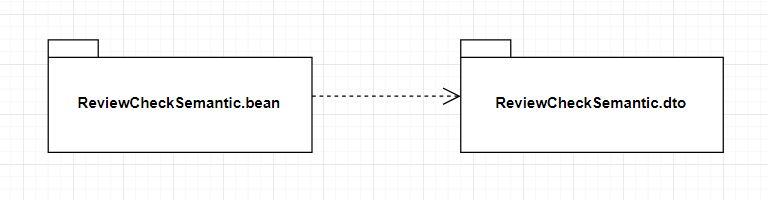
Módulo ReviewRead:



Módulo ReviewWrite:

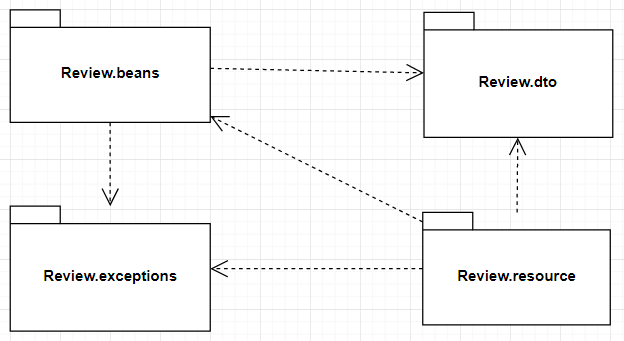


Módulo ReviewCheckSemantic:

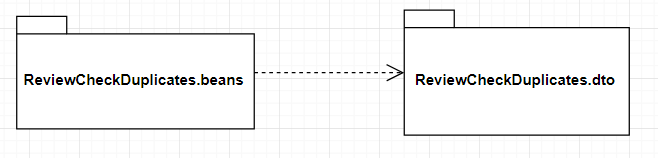


Por razones de tiempo se hizo un mock del modulo ReviewCheckSemantic

Módulo Review:



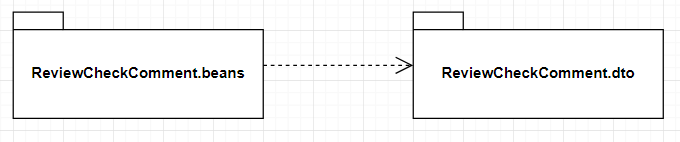
Módulo ReviewCheckDuplicates:



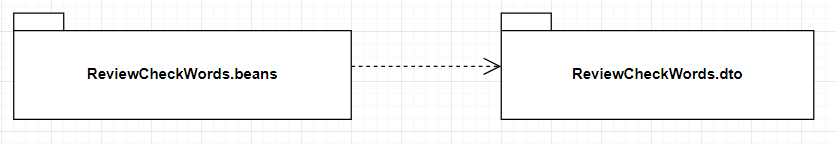
Módulo ReviewCheckClient:



Módulo ReviewCheckComment:



Módulo ReviewCheckWords:



Modulo Login:

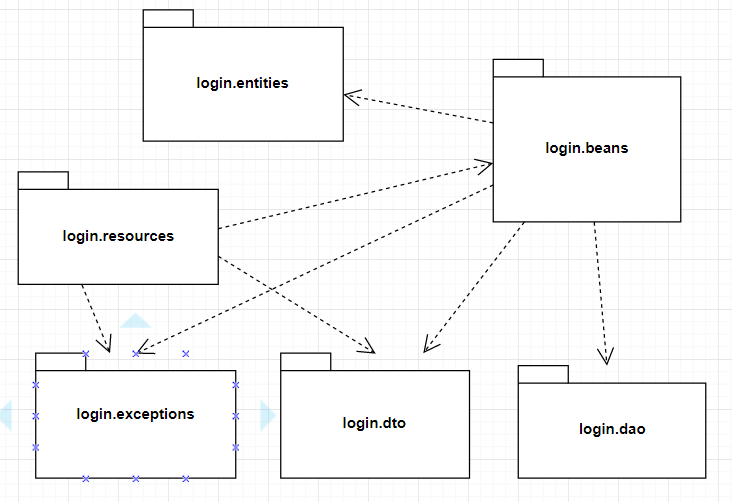
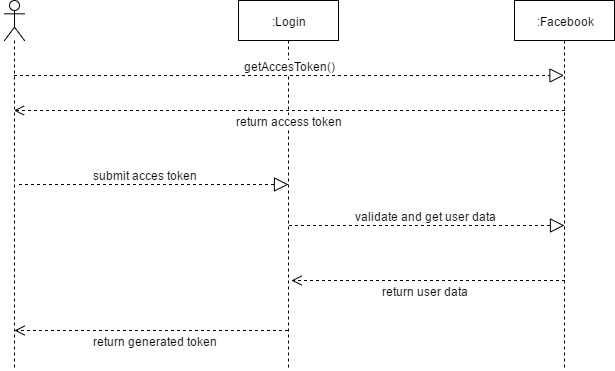


Diagrama de secuencia



Para realizar la autenticación y autorización se realiza el módulo login el cual posee un WAR y un EJB.

Dentro del componente WAR un resource que contiene dos métodos principales, estos son doLogin y verifyToken, el comportamiento de la primera es el siguiente: el loginBean llama al servicio de Facebook con el token dado por el cliente, este es verificado y si es correcto se envía al loginBean los datos asociados a este token (nombre y username). Estos son checkeados en la base de datos, si no existen entonces quiere decir que el cliente no está registrado en el sistema por lo tanto este se agrega a la base y se genera un token único que se almacena y luego se retorna, en caso de que exista, la aplicación simplemente genera un token que actualiza y que luego retorna. Este token es único de cada cliente logueado, es propio de la aplicación y cada vez que quiera realizar una acción dentro del sistema deberá enviar ese token para que sea validado y verificar que el mismo se encuentra logueado. Esta validación la realiza el verifyToken.

El método doLogin solo se ejecuta cuando el cliente intenta loguearse al sistema a diferencia del verifyToken que se llama cada vez que el cliente quiera realizar una operación dentro del sistema.

Por otro lado tenemos el componente EJB, en el cual se encuentra el loginBean que se encara de gestionar la logica de negocios de la aplicacion, el loginDto que es el encargado transportar la información entre los procesos, una clase loginEntity que se encarga del acceso a la base de datos y por ultimo un loginException que se encarga de manejar las excepciones asociadas al login.

Ventajas de login/Register

* Inicialmente el proceso por el cual el cliente obtiene el token por parte del proveedor no se implementa, se asume que el cliente ya tiene este token, esto es una decisión de diseño que se tomó para poder hacer el login mas genérico y que sirva para más de un proveedor. Si en un futuro se quiere realizar la autenticación y autorización con otro proveedor simplemente debemos pasar el token de este nuevo proveedor y el tipo de proveedor con el que queremos realizar el login es decir favorece la extensión.
* Solo depende de servicios externos una vez, esto se da cuando se le pide al proveedor que retorne los datos asociados al token que se envía.

Desventajas de login/Register

* Depende únicamente del cliente obtener el token del proveedor.

# 3.1.2.3 Guía de variabilidad

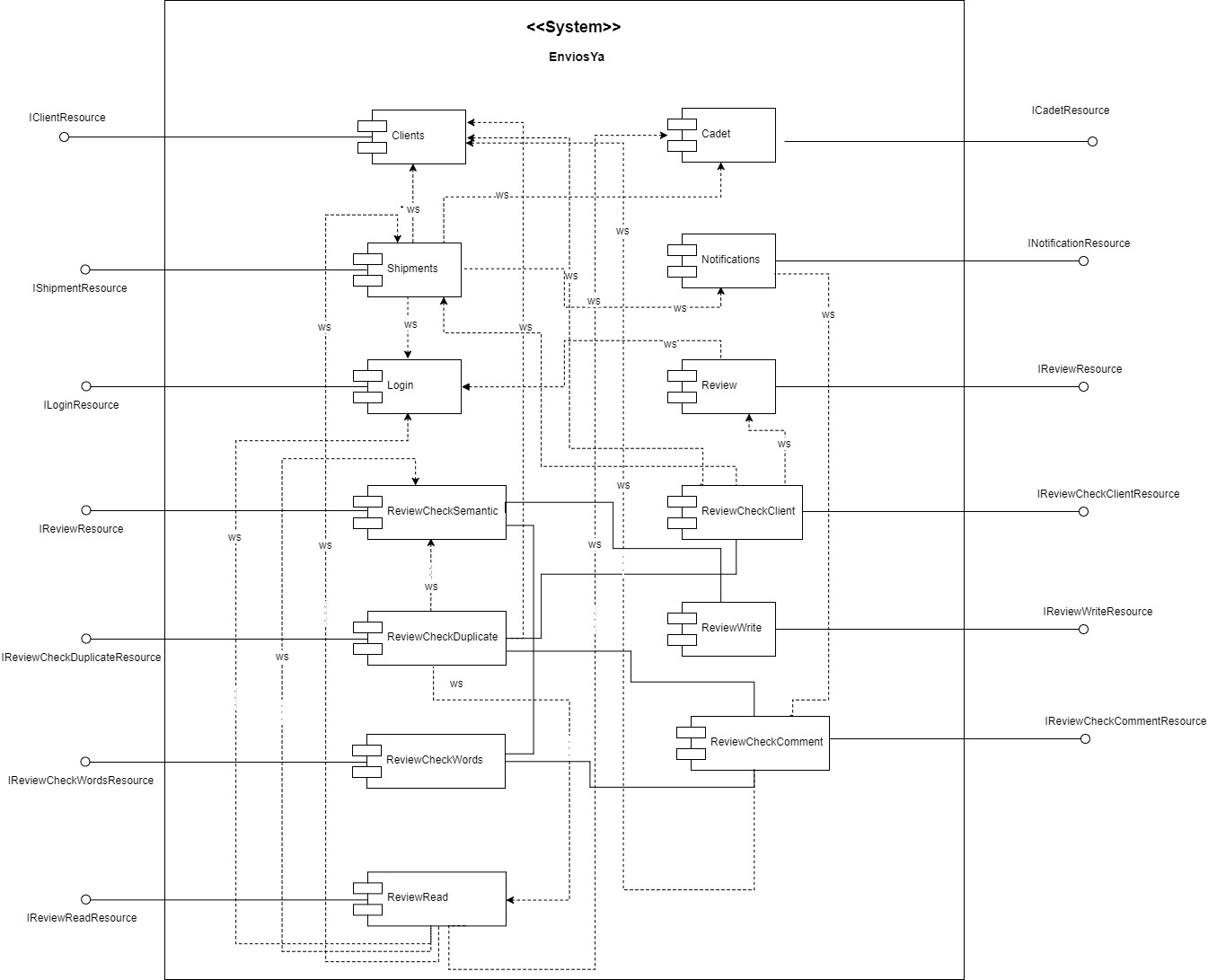
La solución provista hoy en día contiene un paquete DTO por cada módulo implementado, está pensado para un futuro hacer un EAR donde se encuentren todos los componentes a ser consumidos por las aplicaciones.

# 3.1.3. Vista de Componentes y conectores

En la vista de componentes y conectores se puede apreciar lo mas importante al momento de mostrar y comunicar la visión del sistema en tiempo de ejecución.

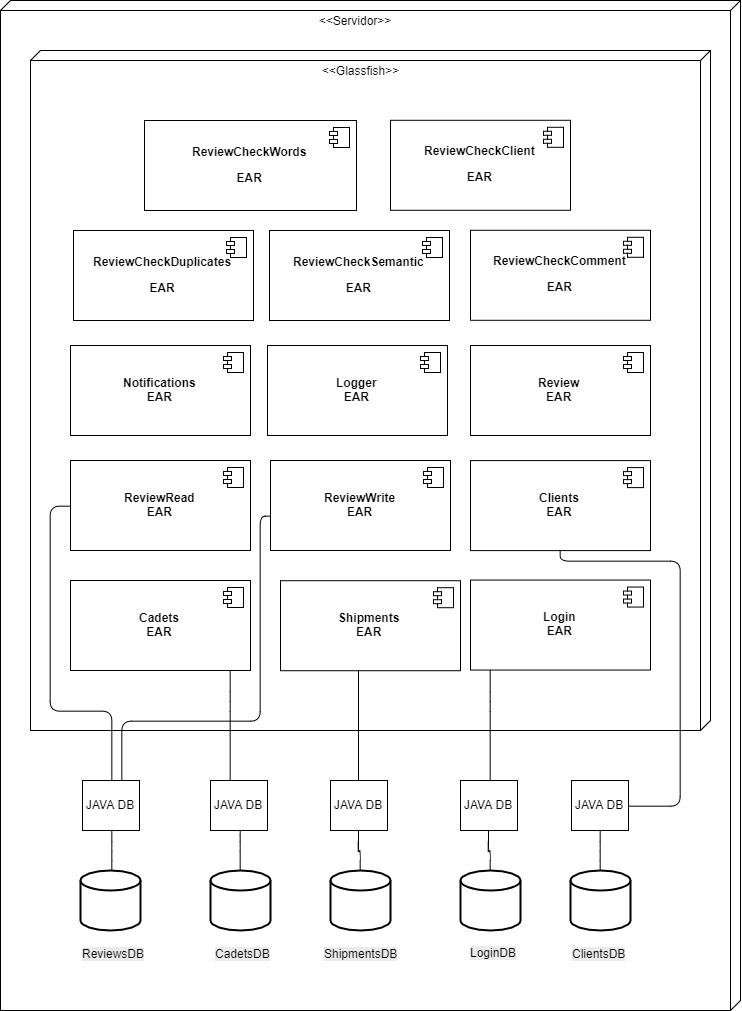
# 3.1.3.1 Representación primaria

En esta vista se pueden ver las relaciones de los Web Services, SessionBeans y Entities de nuestra aplicación.



# 3.1.4. Vista de despliegue

A partir de esta vista se aprecia cómo se comunican los componentes del sistema con la base de datos.



Catálogo de elementos:

EnviosYa:

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **com.enviosya.clients** | Este paquete contiene toda la información asociada a los clientes.  Implementa las reglas de negocio para gestión de los clientes que solicitan envíos, sus datos personales, ubicaciones, etc. |
| **com.enviosya.cadets** | Este paquete contiene toda la información asociada a los cadetes y sus vehículos.  Implementa las reglas de negocio para la gestión de los cadetes que realizan los envíos, sus datos personales, los vehículos que usan, etc. |
| **com.enviosya.shipments** | Este paquete contiene toda la información asociada a los envíos.  Implementa las reglas de negocio para el registro de solicitudes de envío de los clientes. |
| **com.enviosya.review** | Este paquete contiene toda la información asociada a las calificaciones.  Implementa la gestión de las calificaciones que hacen los clientes sobre los envíos y los cadetes. |
| **com.enviosya.notifications** | Este paquete contiene toda la información asociada a las notificaciones.  Implementa y provee a los demás módulos formas de comunicación interna y externa a través de diferentes medios. |
| **com.enviosya.logger** | Este paquete contiene toda la información asociada al Logger. Implementa un log que almacena el detalle de las tareas que realiza el sistema. |
| **com.enviosya.login** | Este paquete maneja la autenticación y autorización de los clientes en el sistema. |

Clients

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **clients.resource** | Expone servicios REST para ser consumidos y posibilitar la gestión de los clientes y los medios de pagos que estos poseen. |
| **clients.beans** | Responsable de la lógica del negocio requerida para satisfacer los requisitos funcionales y no funcionales de la gestión de clientes y sus medios de pagos. |
| **clients.exceptions** | Este módulo contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **clients.dao** | Este módulo contiene utilidades para el acceso al repositorio de datos de clientes. |
| **clients.entities** | Este módulo contiene las entidades que se persistirán en la base de datos. |
| **clients.dto** | Este módulo contiene objetos con información referente a clientes y sus medios de pagos que transportan datos entre procesos. |

Cadets

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **cadets.resources** | Expone servicios REST para ser consumidos y posibilitar la gestión de los cadetes, los vehículos y las relaciones entre ellos. |
| **cadets.beans** | Responsable de la lógica del negocio requerida para satisfacer los requisitos funcionales y no funcionales de la gestión de cadetes y las relaciones (únicamente) que mantienen con los vehículos. |
| **cadets.exceptions** | Este módulo contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **cadets.dao** | Este módulo contiene utilidades para el acceso al repositorio de datos de cadetes y vehículos. |
| **cadets.entities** | Este módulo contiene las entidades que se persistirán en la base de datos.(tanto cadetes como su lista de vehículos) |
| **cadets.dto** | Este módulo contiene objetos con información de cadetes y sus vehículos asociados que transportan datos entre procesos. |

Shipments

|  |  |
| --- | --- |
| Elemento | Responsabilidades |
| **shipments.resources** | Expone servicios REST para ser consumidos y posibilitar la gestión del envió solicitado. |
| **shipments.beans** | Responsable de la lógica del negocio requerida para satisfacer los requisitos funcionales y no funcionales de la gestión de envíos. |
| **shipments.exceptions** | Este paquete contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **shipments.dao** | Este paquete contiene utilidades para el acceso al repositorio de datos de los envíos. |
| **shipments.entities** | Este módulo contiene las entidades que se persistirán en la base de datos. |
| **shipments.dto** | Este módulo contiene objetos con información de envíos que transportan datos entre procesos. |
| **shipments.domain** | Este módulo contiene la lógica para realizar el cálculo del costo del envió del paquete. |

Login

|  |  |
| --- | --- |
| Elemento | Responsabilidades |
| **login.resources** | Expone servicios REST para ser consumidos y posibilitar la creación de usuarios al sistema y gestionar la autenticación, autorización y auditoría de los mismos. |
| **login.beans** | Responsable de la lógica del negocio requerida para satisfacer los requisitos funcionales y no funcionales de la autenticación y autorización |
| **login.exceptions** | Este paquete contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **login.dao** | Este paquete contiene utilidades para el acceso al repositorio de datos de los clientes registrados al sistema. |
| **login.entities** | Este módulo contiene las entidades que se persistirán en la base de datos. |
| **login.dto** | Este módulo contiene objetos con información de clientes que transportan datos entre procesos. |

Notifications

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **notifications.resources** | Expone servicios REST para ser consumidos y posibilitar el envío de mensajes a través de correos electrónicos. |
| **notifications.beans** | Responsable de la lógica del negocio requerida para satisfacer los requisitos funcionales y no funcionales de la gestión de la mensajería. |
| **notifications.domain** | Este módulo tiene la responsabilidad de enviar un mensaje recibido en determinado formato por correo electrónico. |
| **notifications.dto** | Este módulo contiene objetos con información referente a los mensajes que transportan datos entre procesos. |
| **notifications.exceptions** | Este módulo contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |

Review

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **Review.resource** | Expone servicios REST para poder iniciar el proceso de creación de una Review. |
| **Review.beans** | Se encarga de validar que el usuario que esta autenticado y de encolar la review en la cola "QueueValidateClient", para que un responsable la desencola y siga el proceso de creación de Review. |
| **Review.exceptions** | Este módulo contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **Review.dto** | Este módulo contiene un objeto ReviewDto con información referente a la review a ser creada que transporta datos entre procesos. |

ReviewCheckClient

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **ReviewCheckClient.beans** | Se encarga de desencolar la Review de la cola "QueueValidateClient ", validar que el usuario que está haciendo el Review (logueado al sistema) sea el mismo que solicitó el envío y encolar en caso de que se cumpla la condición en la cola "QueueCheckDuplicates". En caso de no cumplirse la validación se le envía un email al usuario explicando el motivo de no creación de la review. |
| **ReviewCheckClient.dto** | Este módulo contiene un objeto ReviewDto con información referente a la review a ser creada que transporta datos entre procesos. Tambiés contiene un objeto ClienteDto y EnvioDto, necesarios para el elemento anterior. |

ReviewCheckDuplicates

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **ReviewCheckDuplicates**  **.beans** | Se encarga de desencolar la Review de la cola " QueueCheckDuplicates ", validar que no exista un Review previo para el mismo envío y encolar en caso de que se cumpla la condición en la cola "QueueCheckComment". En caso de no cumplirse la validación se le envía un email al usuario explicando el motivo de no creación de la review. |
| **ReviewCheckDuplicates**  **.dto** | Este módulo contiene un objeto ReviewDto con información referente a la review a ser creada que transporta datos entre procesos y un objeto ClienteDto que es necesario para que el elemento anterior pueda recuperar el email del cliente y guardarlo aquí. |

ReviewCheckComment

|  |  |
| --- | --- |
| Elemento | Responsabilidades |
| **ReviewCheckComment.beans** | Se encarga de desencolar la Review de la cola "QueueCheckComment ", validar que el comentario de la Review tenga una cantidad mínima de palabras y encolar en caso de que se cumpla la condición en la cola "QueueCheckWords". En caso de no cumplirse la validación se le envía un email al usuario explicando el motivo de no creación de la review. |
| **ReviewCheckComment.dto** | Este módulo contiene un objeto ReviewDto con información referente a la review a ser creada que transporta datos entre procesos y un objeto ClienteDto que es necesario para que el elemento anterior pueda recuperar el email del cliente y guardarlo aquí. |

ReviewCheckWords:

|  |  |
| --- | --- |
| Elemento | Responsabilidades |
| **ReviewCheckWords.beans** | Se encarga de desencolar la Review de la cola " QueueCheckWords", validar que el comentario de la Review no contenga “malas palabras” definidas en un archivo txt que se encuentra en la carpeta c:/listanegra, pudiendo cambiar esta ubicación. Se encarga de encolar en caso de que se cumpla la condición en la cola " QueueCheckSemantic". |
| **ReviewCheckWords.dto** | Este módulo contiene un objeto ReviewDto con información referente a la review a ser creada que transporta datos entre procesos. |

ReviewCheckSemantic

|  |  |
| --- | --- |
| Elemento | Responsabilidades |
| **ReviewCheckSemantic.beans** | Se encarga de desencolar la Review de la cola "QueueCheckSemantic", y determinar un valor de “sentimiento” (positivo, neutro, negativo) y asociarlo a la Review. El sentimiento se determina a través de un llamado a un método de una interfaz, pudiendo cambiar la implementación de este de forma sencilla.  . Se encarga de encolar en caso de que se cumpla la condición en la cola "QueueAddReview". |
| **ReviewCheckSemantic.dto** | Este módulo contiene un objeto ReviewDto con información referente a la review a ser creada que transporta datos entre procesos. |

ReviewWrite

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **ReviewWrite.beans** | Se encarga de desencolar la Review de la cola "QueueAddReview" para dar de alta la Review en la base de datos, en caso que el estado se la review sea "approved", se le notifica por email al usuario que efectivamente su review fue dada de alta con éxito. |
| **ReviewWrite.exceptions** | Este módulo contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **ReviewWrite.dao** | Este módulo se encarga de comunicarse con la base de datos de Review, para dar de alta una Review. |
| **ReviewWrite.entities** | Este módulo contiene las entidad Revision que se persistirán en la base de datos, junto con sus ennumerados. |
| **ReviewWrite.dto** | Este módulo contiene los objetos ReviewDto y ClienteDto, necesarios para el elemento ReviewReading.beans. |

ReviewRead

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **ReviewRead.beans** | Se encarga de resolver la lógica de negocio asociada al proceso de listar todas las calificaciones de un Cadete |
| **ReviewRead.exceptions** | Este módulo contiene excepciones propias del sistema para capturar determinados eventos en la lógica de la aplicación. |
| **ReviewRead.dao** | Este módulo se encarga de comunicarse con la base de datos de Review, para hacer diferentes consultas sobre Review. |
| **ReviewRead.entities** | Este módulo contiene la entidad Review y los enums (Review State y ReviewFeeling) que esta precisa. |
| **ReviewRead.dto** | Se guarda un objeto Dto que tiene la estructura de la respuesta de la consulta " listar todas las calificaciones de un Cadete". También se guardan otras clases Dtos, necesarias para recuperar datos en el bean y tener una estructura para guardarlos. |
| **ReviewRead.resources** | Expone servicios REST para poder ser consumido desde afuera, este servicio expone las operaciones de listar todas las calificaciones de un Cadete y otra operación más para poder obtener una review con un envío y cliente específico. |

Justificación de decisiones de diseño

En la siguiente sección de detallaran las decisiones tomadas con el objetivo de cumplir con los requerimientos no funcionales del sistema y por  defecto con los atributos de calidad.

**RNF1** – Para cumplir con este requerimiento se utilizó el patrón pipe and filters, este trata de un conjunto de componentes separados, cada uno realizando una tarea específica. Mediante la estandarización del formato de los datos que cada componente recibe y retorna y se combinan de modo que uno precede a otro. Esto ayuda a evitar la duplicación de código y facilita la eliminación, el reemplazo o la integración de componentes adicionales en tiempo de ejecución si cambian los requisitos de la tarea que llevan a cabo. Para llevarlo al caso práctico, el proceso de creación de la Review se divide en varios módulos con responsabilidades bien definidas que esperan por la respuesta del componente anterior y que otorga datos procesados al siguiente modulo favoreciendo la modificabilidad.

**RNF2** - Para poder satisfacer este requerimiento se implemento el patrón Pipe-and-Filter, ya que los Pipes almacenan datos durante la comunicación, debido a esto se pueden ejecutar filtros asincrónicamente y concurrentemente. Este patrón es de gran utilidad a la hora de utilizar la interfaz requerida que asigna un valor de “sentimiento” al comentario, el cual tiene una latencia promedio de un segundo. Utilizando Pipe-and-Filter, este servicio puede utilizarse en paralelo sin impactar en la latencia de la funcionalidad. Teniendo en cuenta las tácticas mencionadas en las decisiones de las otras aplicaciones y la utilización del patrón Pipe-and-Filter, la latencia del proceso de creación de una calificación se verá favorecida.

**RNF3** -  Se implementan las tácticas de autenticación y autorización de los clientes con Facebook  para cumplir con este requerimiento y para favorecer la seguridad de la aplicación.

**RNF4** -  Como táctica para poder recuperarse de las fallas del sistema se optó por utilizar el manejo de excepciones. En cada módulo se tiene una clase destinada a manejar las excepciones propias que el sistema arroja de modo de favorecer la disponibilidad requerida en este punto.

**RNF5** - Se implementó un logger ya que el mismo permite reutilizar información de las tareas del Sistema para poder reconstruirlas favoreciendo la disponibilidad del Sistema.

**RNF6** – Para el proceso de creación de una calificación se separó el proceso de escritura y lectura por medio de dos componentes. Tenerlo separado aumenta la disponibilidad de la parte de consultas ya que sería independiente del proceso de creación. La decisión de diseño tomada es aplicable en las otras aplicaciones en la medida que presenten procesos más complejos como el de calificaciones.

**RNF7** – Como táctica para cumplir con este requerimiento no funcional se encriptaron los datos para resistir a los ataques y favorecer la seguridad del sistema. Los datos encriptados fueron los de la tarjeta de crédito de los clientes, más específicamente el número de tarjeta y el CCV que luego de ser encriptados son guardados en la base de datos.

**RNF8** -  Como en el RNF5 el logger también satisface las necesidades de este requerimiento ya que este módulo es el encargado de registrar las tareas que realiza el sistema. El mismo es consumido por todos los módulos restantes del sistema generando una relación de dependencia con cada uno de ellos.

**RNF9** – No implementado. En un futuro se planifica implementar un cache que almacena los datos del cliente con actividad en el sistema en los últimos 15 dias y de esta forma darle prioridad a estos.

**RNF10** – Para cumplir con lo pretendido en este punto, dentro del paquete enviosya.shipments.domain se implementó una clase abstracta llamada Package. Esta clase implementa todos sus métodos a diferencia de uno, este (getCost) es un método abstracto que es implementado por una clase llamada PackageNormal que extiende de Package. La misma sobrescribe el método indicándole la forma de calcular el costo del paquete a enviar. Esta decisión fue tomada para hacer esta sección altamente modificable, es decir, que si se quieren agregar nuevas formas de calcular este costo basta con agregar una nueva clase que sobrescriba este método generando un mínimo impacto sobre la aplicación.