Universidad ORT Uruguay
Facultad de Ingeniería

EnviosYA

DOCUMENTO RESUMIDO: DESCRIPCIÓN DE ARQUITECTURA

Diego Carrión – 172129
Emily Symonds – 172168

Profesor: Gastón Mousqués
Mathías Fonseca

2016

Tabla de contenido

Introducción

En el presente documento se presenta una visión global de la arquitectura del sistema, representando mediante las diferentes vistas, cada una de las cuales ilustran aspectos particulares del software a desarrollar.

1.1 Propósito

El objetivo de este documento es brindar una visión general de la arquitectura del sistema de EnviosYA. No se incluyen detalles de implementación ya que el propósito es presentar las principales decisiones de diseño y sus correspondiente justificación.

2. Antecedentes

2.1 Propósito del sistema

El sistema EnviosYA presenta ser una solución para la gestión y el envío de paquetes entre diferentes actores. En rasgos generales, cualquier persona registrada como usuario (cliente) puede realizar una solicitud de envío. El sistema asigna el envío a uno de los cuatro cadetes más cercanos, quien recoge el paquete y lo lleva a la ubicación destino. Una vez recibido, el destinatario confirma la recepción en el sistema, notificando mediante *email* a los usuarios, con la posibilidad de calificar el servicio y al cadete.

El costo del envío depende de la distancia entre el origen y destino y del tamaño y peso del paquete. Para calcular estos últimos dos parámetros, el cliente debe subir una foto del paquete y la aplicación los calculará automáticamente.

2.2 Requerimientos significativos de Arquitectura

2.2.1 Resumen de Requerimientos Funcionales

ID Requerimiento	Descripción	Actor
RF 1 - Mantenimiento de Cadetes y Vehículos	El sistema registra la información de los cadetes (CI, nombre, apellido, *email*) y sus vehículos. También se puede modificar y borrar a los cadetes y sus vehículos. La cédula de los cadetes no puede ser repetida y se deberá validar que el correo electrónico sea válido.	Administrador
RF 2 - Mantenimiento de	El sistema registra la información de los clientes (CI, nombre, apellido, *email*). También se deberá poder modificar los datos y borrar a los clientes. La cédula	Administrador

Clientes	no puede ser repetida, y se debe validar que el correo electrónico sea válido.	
RF 3 - Registrar una solicitud de envío	El sistema permite que los clientes soliciten envíos a destinatarios. Para ello crean un Envío (ID, descripción, dirección de emisión y recepción, forma de pago, comisión, foto paquete). El cliente selecciona el cadete que desea (dentro de los cuatro más cercanos) y el sistema le asigna el envío notificando los datos del envío a realizar.	Usuario
RF 4 - Registrar recepción de envío	El destinatario confirma la recepción del envío y el sistema notifica vía *email* a ambos usuarios los datos del envío, de su recepción, la información sobre el cadete que realizó el traslado y un link para calificar el servicio y al cadete que realizó el traslado.	Usuario
RF 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
RF 6 - Listar los envíos pendientes de calificación de un cliente	El cliente accede a la lista de envíos realizados ordenados cronológicamente de forma descendente y el sistema obtiene del módulo la lista de calificaciones en estado "*approved*" para esos envíos.	Usuario
RF 7 - Listar todas las calificaciones de un Cadete	El sistema lista de calificaciones en estado "*approved*" para el Cadete, ordenadas cronológicamente de forma descendente.	Usuario
RF 8 - Rechazar manualmente una calificación inapropiada a un Cadete	Solo usuarios administradores pueden utilizar esta funcionalidad, seleccionando una calificación de un cadete y la opción "rechazar". El sistema marca la calificación con estado "*rejected*".	Administrador
RF 9 - Consultar las calificaciones no aprobadas	Solo usuarios administradores pueden utilizar esta funcionalidad. Se presenta la posibilidad de filtrar las calificaciones por estado e intervalo de fechas de registro de las calificaciones.	Administrador

Tabla 1: Requerimientos funcionales

2.2.2 Resumen de Requerimientos No Funcionales

ID Requerimiento	Atributo de Calidad	Descripción
RNF 1 - Configuración del proceso de creación de un Comentario	Modificabilidad / Disponibilidad	Se debe permitir agregar, quitar o intercambiar pasos con el menor impacto posible en tiempo de desarrollo y despliegue; así como soportar el reintento de los pasos que no lleguen a cumplirse en caso de que el proceso se vea interrumpido.
RNF 2 - Tiempo de respuesta de creación de un Comentario	Eficiencia	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” es de 200ms. En particular el proceso de creación de un Comentario no debe superar los 350ms.
RNF 3 - Usuarios registrados	Seguridad	Todas las funcionalidades descritas sólo pueden ser utilizadas por usuarios debidamente autenticados.
RNF 4 - Gestión de errores y fallas	*Testeabilidad*	El sistema debe proveer suficiente información que permita conocer el detalle de las tareas que realiza. En el caso de una falla o cualquier tipo de error, es imprescindible que el sistema provea toda la información necesaria que permita hacer un diagnóstico rápido y preciso sobre las causas.
RNF 5 - Recuperación y Fallas	Disponibilidad	La información producida por el sistema será 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).

RNF 6 - Independencia de “consultas” y “escrituras”	Modificabilidad	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.
RNF 7 - Protección de los datos de los clientes	Seguridad	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.
RNF 8 - Información de auditoría	Seguridad	El sistema debe registrar información que permita realizar auditorías de acceso de forma de identificar los accesos autorizados y no autorizados al sistema.
RNF 9 - Tiempo de respuesta de creación de una solicitud de envío	Eficiencia	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” es de 200ms. 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.
RNF 10 - Modificación del cálculo de costos de los envíos	Modificabilidad	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.

Tabla 2: Requerimientos no funcionales

3. Documentación de la arquitectura

A continuación se presenta la arquitectura del sistema utilizando diferentes tipos de vistas (módulos, componentes y conectores, y asignación, entre otras), cada una de ellas brindando información de forma particular.

Cada punto de vista tiene una representación primaria, junto con un catálogo de elemento que ilustra la representación, las justificaciones de diseño y una guía de variabilidad en aquellas vistas que se consideran pertinentes.

3.1 Vistas de Módulos

3.1.1 Vista de Descomposición

En esta vista se aprecia la estructura del sistema, teniendo un elemento que engloba a los demás elementos llamado "*com.enviosya*".

3.1.2 Representación primaria

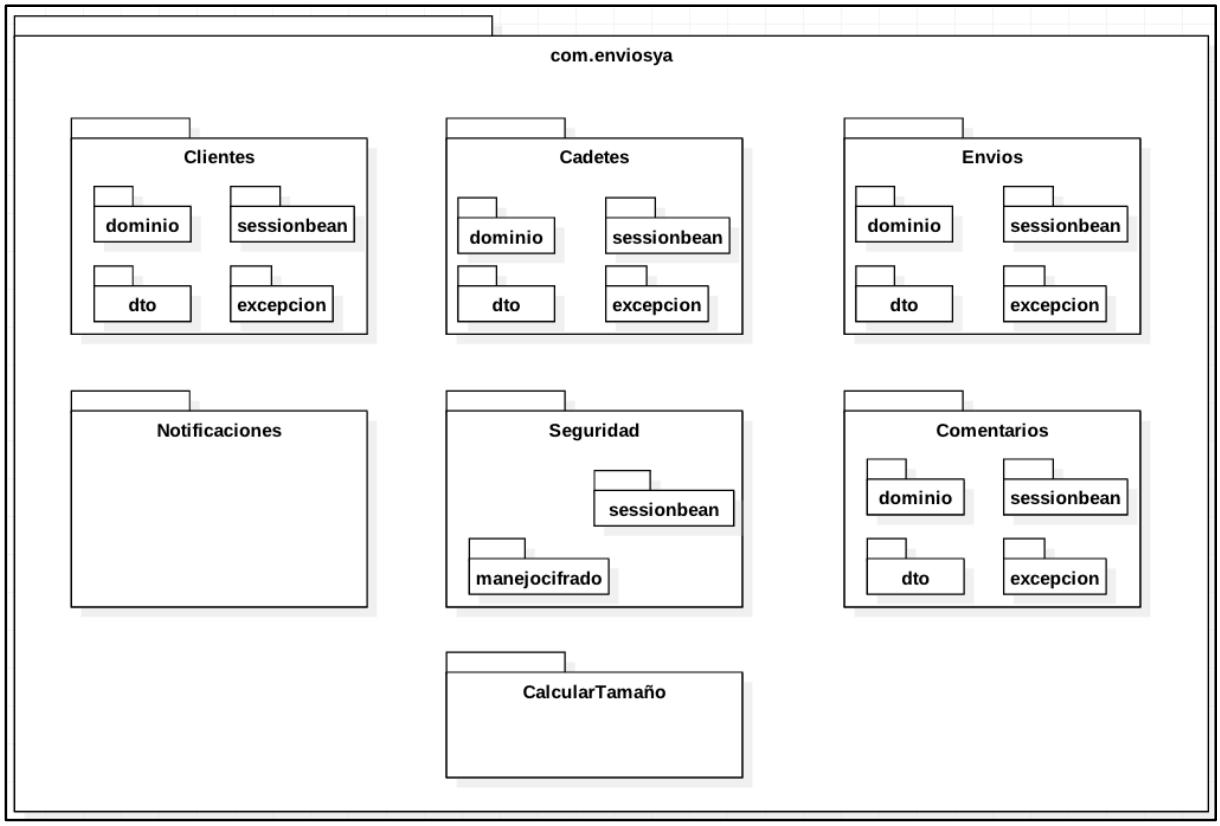

Figura 1: Vista de Descomposición

3.1.3 Catálogo de elementos

Elemento	Responsabilidades
Clientes	Implementa las reglas de negocio para gestión de los clientes que solicitan envíos, sus datos personales, ubicaciones, etc.
Cadetes	Implementa las reglas de negocio para gestión de los cadetes que realizan los envíos, sus datos personales, los vehículos que usan, etc.
Envíos	Implementa las reglas de negocio para el registro de solicitudes de envío de los clientes, medios de pago, preferencias.
Comentarios	Implementa la gestión de las calificaciones que hacen los clientes sobre los envíos y los cadetes.
Notificaciones	Implementa y provee a los demás módulos formas de comunicación interna y externa a través de diferentes medios (mensajería, *email*, telefónica, etc.), que no se persisten pero si se maneja todo lo relacionado al envío de notificaciones a un usuario.
Seguridad	En este módulo se agrupan todas las responsabilidades referentes a la autenticación, tanto de usuarios como administradores. Además del manejo de la seguridad (*tokens*, roles, etc.).
CalcularTamaño	Responsable del 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.

Tabla 3: Catálogo de elementos

3.2 Vista de Uso

En esta vista se presenta cómo los paquetes utilizan a otros paquetes. Es en esta vista donde se visualiza el impacto de cambio entre los distintos módulos del sistema.

Agrupando lógicamente por *feature* es que logramos que el impacto de cambio fuese reducido al feature específico (justificación desarrollada más adelante en el documento).

Se pueden ver algunas dependencias circulares (como por ejemplo entre los módulos Seguridad y Cliente), que se dan cuando un *Web Service* necesita de un *Dto* de otro módulo.

3.2.1 Representación primaria

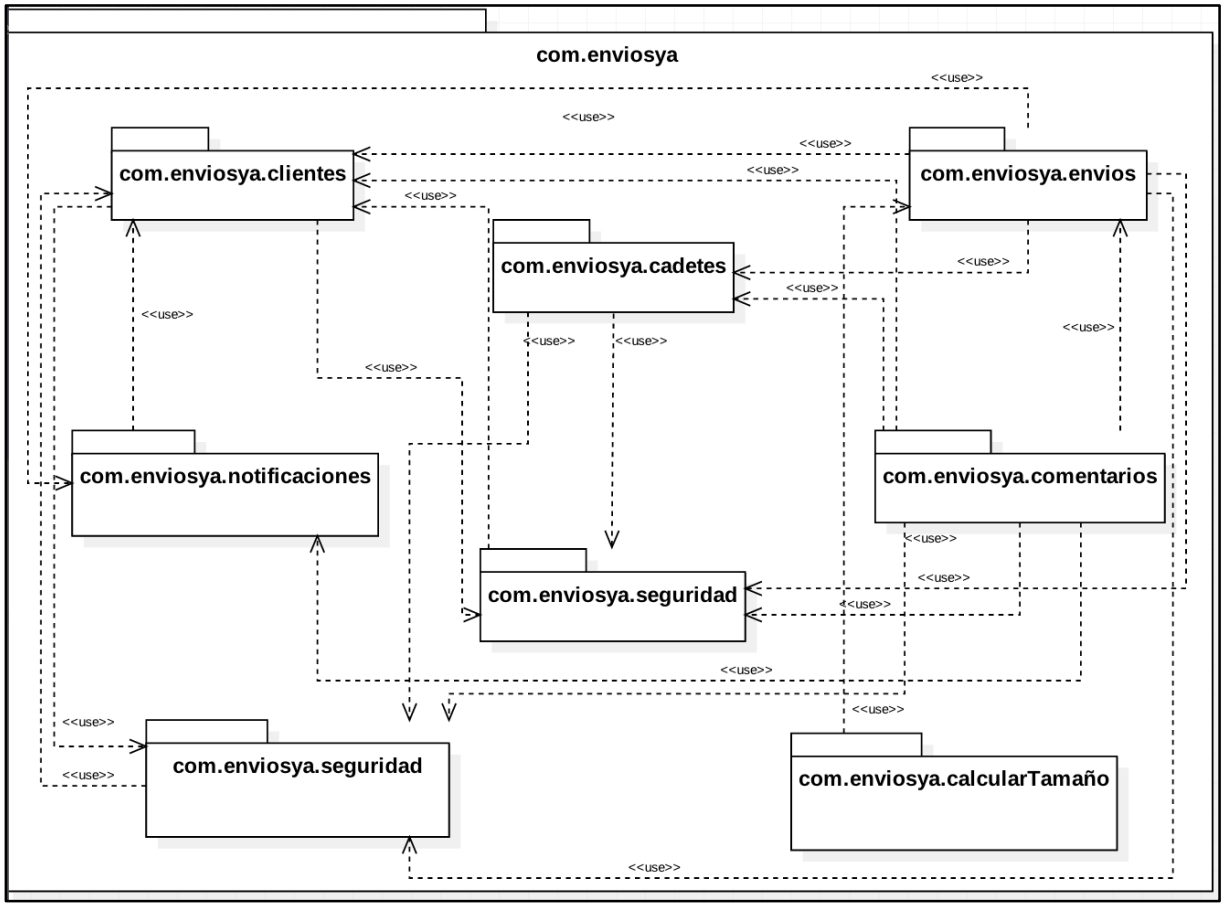

Figura 2: Vista de Uso

3.3 Vista de *Layers*

A continuación se muestra las diferentes capas: servicios, lógica de negocio y acceso a datos.

Para la realización de dicha vista separamos los diferentes módulos, colocando cada *Session Bean*, *Web Service* y *Entity* en la capa pertinente. En el siguiente diagrama se muestran agrupados los *Dtos* para no sobrecargar el diagrama, pero cada uno de los módulos tiene un paquete con sus *Dtos* correspondientes.

A partir de este diagrama se concluye que no existe ninguna capa transversal, además de que no existe ninguna conexión que se sobresalte alguna capa adyacente, así como ninguna conexión que regrese hacia atrás.

3.3.1 Representación primaria

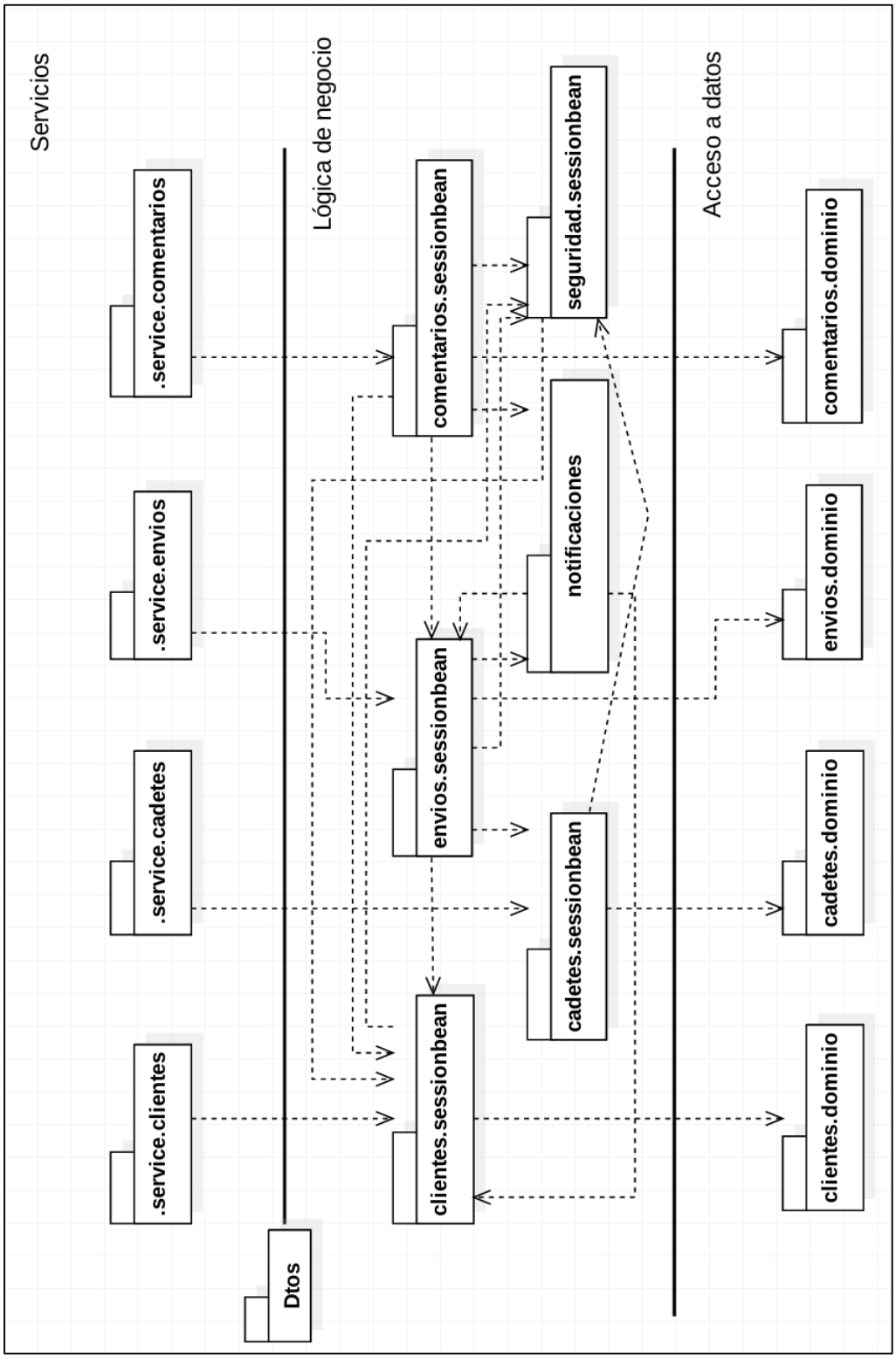

Figura 3: Vista de Layers

3.4 Vista de Datos

En esta vista presentada a continuación se puede visualizar el modelo de entidades (modelo de datos).

3.4.1 Representación primaria

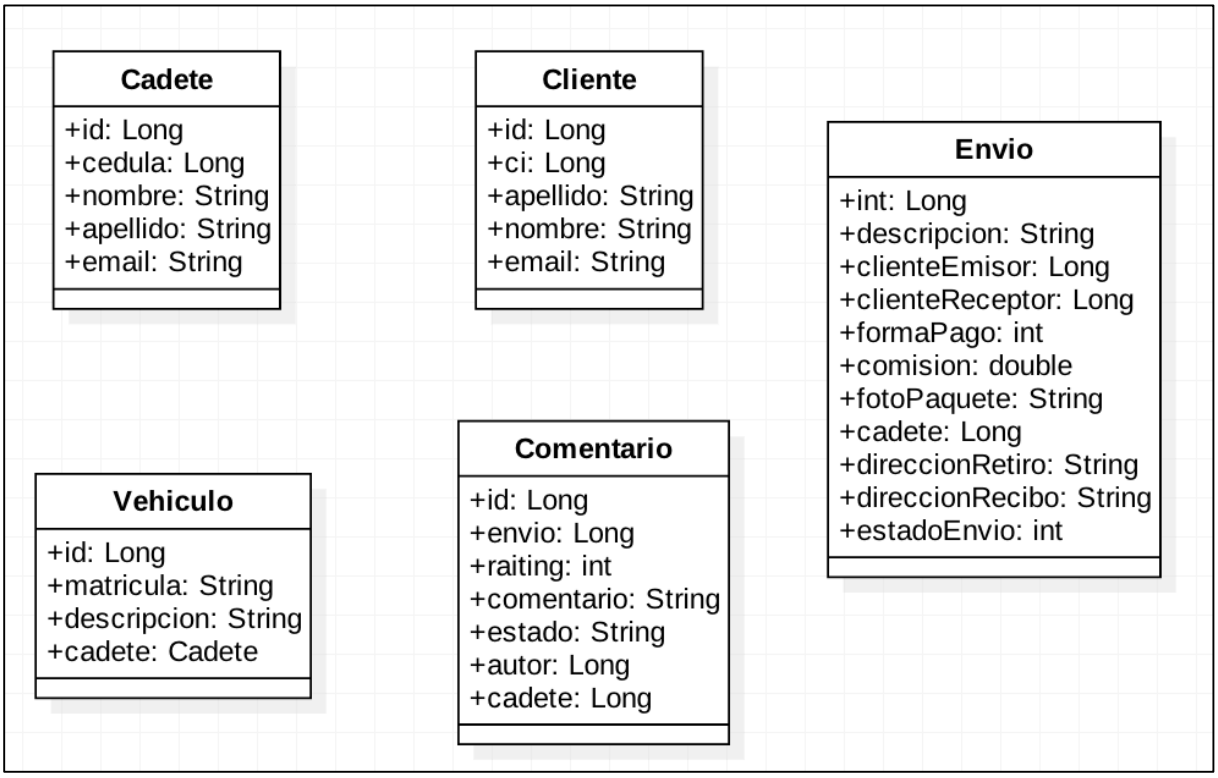

Figura 4: Vista de Datos

3.5 Decisiones de diseño

En primer lugar decidimos priorizar la modificabilidad del sistema, separándolo por módulos altamente cohesivos, implementando la táctica de mantener la coherencia semántica. Como los datos y operaciones de consulta a datos deben poder accederse por separado, optamos por agrupar las responsabilidades por *features*, es decir, por responsabilidad y no por funcionalidad. Además, se apuntó a reducir la dependencia entre módulos al mínimo, haciendo llamadas entre ellos en situaciones imprescindibles.

Dentro de cada módulo, realizamos una separación por funcionalidades, agrupando los componentes en distintos paquetes. Esta separación puede traducirse a un diseño de layers, ya que cada módulo cuenta con un paquete a nivel de interfaz, otro a nivel de lógica y un tercero de acceso a datos. De esta forma, agregar una nueva funcionalidad al sistema es sencillo ya que definir dónde implementarla depende primeramente del módulo al que

aplica y luego de la función que cumple. Este diseño provee un bajo acoplamiento entre módulos dado a que el impacto de un cambio se da dentro del mismo módulo.

4. Vistas de Componentes y conectores

En la vista de C&C (Componentes y conectores) se puede apreciar lo más importante al momento de mostrar y comunicar la visión del sistema en tiempo de ejecución.

4.1 Representación primaria

La siguiente representación se divide en tres esquemas. El primero de ellos, el cliente (*Web Browser*) el cual se conecta a través de interfaces (usuarios, clientes, envíos, etc.). El esquema centra, en el cual se pueden ver las relaciones de los Web Services, Session Beans y Entities de nuestra aplicación. Por último, el esquema compuesto por el componente 'Derby' y la Base de datos.

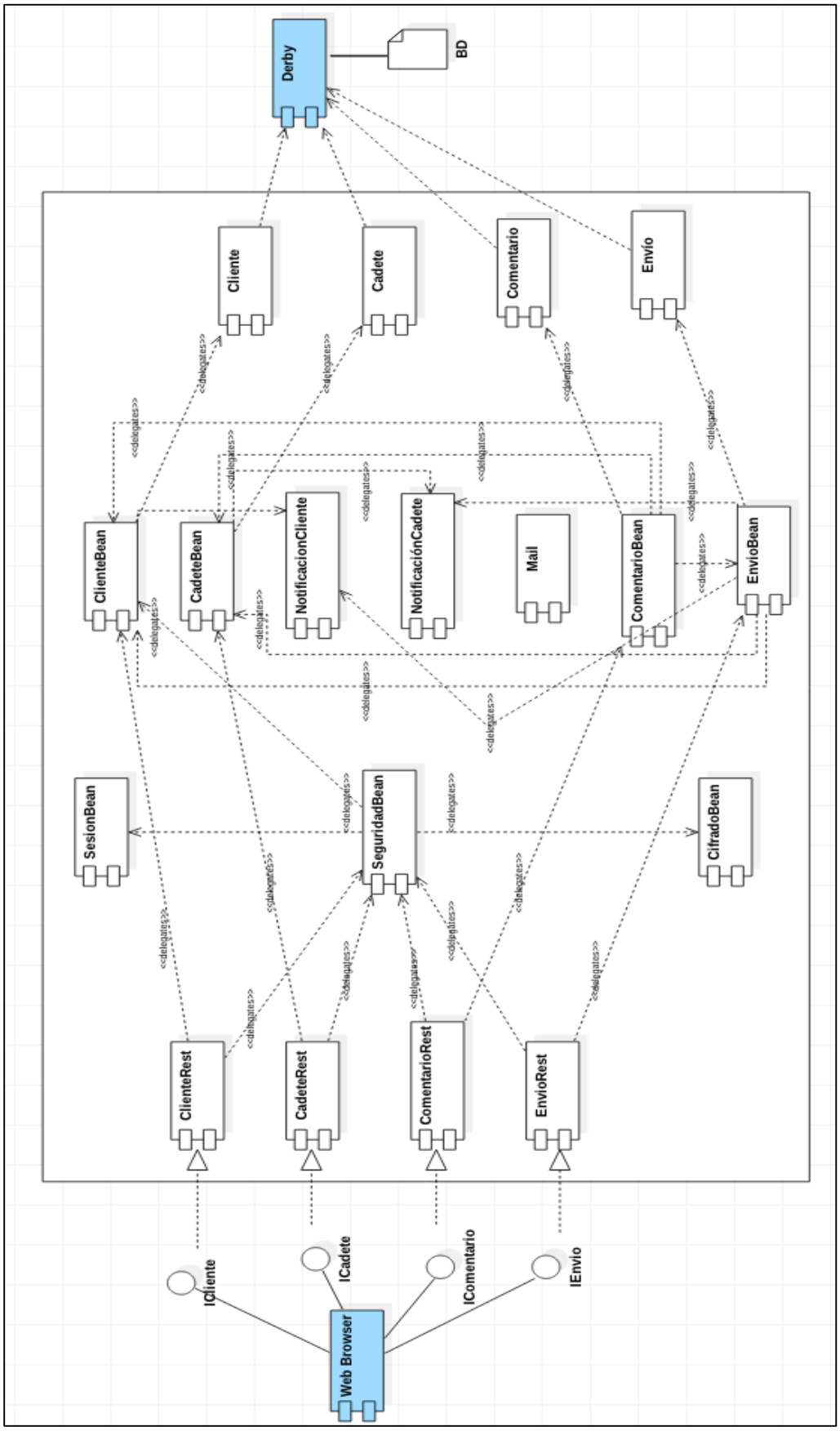

Figura 5: Vista de Componentes y conectores

4.2 Catálogo de elementos

Componente/conector	Tipo	Descripción
Web browser	Componente	Componente o actor que realiza llamados al sistema.
ClienteResource	Web Service	Servicios web de los clientes (crear, modificar, etc.).
CadeteResource	Web Service	Servicios web de los cadetes (crear, modificar, etc.).
EnvioResource	Web Service	Servicios web de los envíos (crear, asignar, listar, etc.).
ComentarioResource	Web Service	Servicios web de los comentarios (crear, listar, etc.).
ClienteBean	Stateless SB	Aquí se encuentra toda la lógica del cliente.
CadeteBean	Stateless SB	Aquí se encuentra toda la lógica del cadete.
EnvioBean	Stateless SB	Aquí se encuentra toda la lógica del envío.
ComentarioBean	Stateless SB	Aquí se encuentra toda la lógica del comentario.
Mail	Stateless SB	Es quien se encarga de crear la conexión con el cliente, los parámetros y la cierra.
NotificarCadete	MDB	Es la clase que escucha, donde está el método "*onMessage*" y posee el *switch* con los diferentes tipos de notificación.
NotificarCliente	MDB	Es la clase que escucha, donde está el método "*onMessage*" y

		posee el *switch* con los diferentes tipos de notificación.
AutenticaciónBean	Stateless SB	Aquí se encuentra toda la lógica de la autenticación.
Cliente	Entity	Describe la estructura de un cliente.
Cadete	Entity	Describe la estructura de un cadete.
Envío	Entity	Describe la estructura de un envío.
Comentario	Entity	Describe la estructura de un envío.
Derby	Componente	Es quien maneja la BD.
Base de datos	Artefacto	Es quien guarda todos los datos.

Tabla 4: Catálogo de elementos

4.3 Interfaces

Interfaz: Cliente	
Paquete que la implementa: ClienteResource	
Servicio	***Descripción***
@GET *http://localhost:8080/Clientes-war/cliente*	Devuelve todos los clientes ingresados en el sistema. **Entrada: -** **Salida:** JSON con un listado de con los datos de los clientes ingresados.
@GET *http://localhost:8080/Clientes-war/cliente/obtener*	Devuelve los datos para un cliente específico. **Entrada:** QueryParams: id. **Salida:** JSON con los datos del cliente seleccionado.
@POST *http://localhost:8080/Clientes-war/cliente*	Permite crear un cliente con los datos ingresados. **Entrada:** Body: JSON con los datos del cliente. **Salida:** JSON con los datos del nuevo cliente creado.
@PUT *http://localhost:8080/Clientes-war/cliente*	Permite modificar un cliente existente en el sistema. **Entrada:** Body: JSON con los datos del cliente a modificar. **Salida:** JSON con los datos actualizados del cliente.

Tabla 5: Interfaz Cliente

Interfaz: Cadete	
Paquete que la implementa: CadeteResource	
Servicio	***Descripción***
@GET *http://localhost:8080/Cadetes-war/cadete*	Devuelve todos los cadetes ingresados en el sistema. **Entrada: -** **Salida:** JSON con un listado de con los datos de los cadetes ingresados.
@GET *http://localhost:8080/Cadetes-war/cadete/obtener*	Devuelve los datos para un cadete específico. **Entrada:** QueryParams: id. **Salida:** JSON con los datos del cadete seleccionado.
@GET *http://localhost:8080/Cadetes-war/cadete/buscar*	Devuelve los datos de los cuatro cadetes mas cercanos a cierta ubicación. **Entrada:** QueryParams: ubicación. **Salida:** JSON con los datos de los cuatro cadetes más cercanos a la ubicación indicada.
@POST *http://localhost:8080/Cadetes-war/cadete*	Permite crear un cadete con los datos ingresados. **Entrada:** Body: JSON con los datos del cadete. **Salida:** JSON con los datos del nuevo cadete creado.
@PUT *http://localhost:8080/Cadetes-war/cadete*	Permite modificar un cliente existente en el sistema. **Entrada:** Body: JSON con los datos del cliente a modificar. **Salida:** JSON con los datos actualizados del cliente.

Tabla 6: Interfaz Cadete

Interfaz: Envío	
Paquete que la implementa: EnvioResource	
Servicio	***Descripción***
@GET *http://localhost:8080/Envios-war/envio*	Devuelve todos los envíos realizados ingresados en el sistema. **Entrada: -** **Salida:** JSON con un listado de con los datos de los envíos ingresados.
@POST *http://localhost:8080/Envios-war/envio/crear*	Permite crear una solicitud de envío con los datos ingresados. **Entrada:** Body: JSON con los datos de la solicitud de envío. **Salida:** JSON con los datos de los cuatro cadetes más cercanos a la ubicación del cliente que solicitó el envío.
@POST *http://localhost:8080/Envios-war/envio/confirmar*	Permite confirmar la solicitud de envío asignándole un cadete. **Entrada:** Body: JSON con los datos de la solicitud de envío a confirmar y el cadete a asignársele. **Salida:** JSON con los datos del envío en estado iniciado.
@POST *http://localhost:8080/Envios-war/envio/recepcion*	Confirma la recepción del envío de un paquete. **Entrada:** QueryParams: id. **Salida:** JSON con los datos del envío en estado finalizado.
@GET *http://localhost:8080/Envios-war/envio/obtener*	Devuelve los datos para un envío específico. **Entrada:** QueryParams: id.

	Salida: JSON con los datos del envío seleccionado.

Tabla 7: Interfaz Envío

Interfaz: Comentario	
Paquete que la implementa: ComentarioResource	
Servicio	***Descripción***
@GET http://localhost:8080/Comentarios-war/comentario	Devuelve todos los comentarios a servicios ingresados en el sistema. **Entrada: -** **Salida:** JSON con un listado de con los datos de los comentarios realizados.
@POST http://localhost:8080/Comentarios-war/ /comentario/obtener	Devuelve los comentarios realizados sobre los servicios de un cadete específico. **Entrada:** QueryParams: id. **Salida:** JSON con los datos de los comentarios relativos al cadete seleccionado.
@POST http://localhost:8080/Comentarios-war/comentario/crear	Permite crear un comentario con los datos ingresados. **Entrada:** Body: JSON con los datos del comentario a realizar. **Salida:** JSON con los datos del comentario ingresado y su respectiva valoración.

Tabla 5: Interfaz Comentario

4.4 Comportamiento - Diagrama de secuencia

Para la realización del diagrama de secuencia es que elegimos una de las funcionalidades más permitentes de la aplicación como es el realización de un comentario.

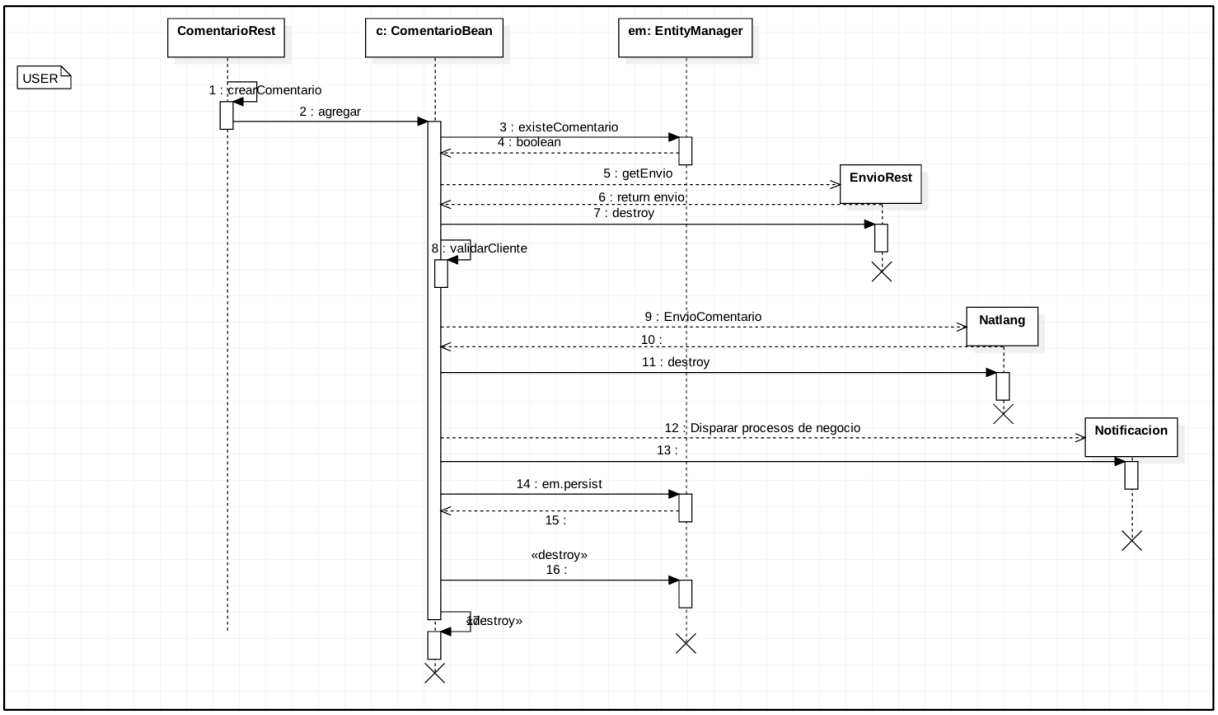

Figura 6: Diagrama de secuencia

4.5 Relación con elementos lógicos

A continuación se presenta la relación de cada componente con su paquete correspondiente. Es sencillo poder ver y encontrar los diferentes componentes en los distintos paquetes.

Componente	Paquetes
Cadete	*com.enviosya.cadete.dominio*
Vehiculo	*com.enviosya.cadete.dominio*
CadeteDto	*com.enviosya.cadete.dto*
VehiculoDto	*com.enviosya.cadete.dto*
CadeteBean	*com.enviosya.cadete.sessionbean*
DatoIncorrectoExcepcion	*com.enviosya.cadete.excepciones*
EntidadNoExisteException	*com.enviosya.cadete.excepciones*
EntidadYaExisteException	*com.enviosya.cadete.excepciones*
Cliente	*com.enviosya. cliente.dominio*
ClienteDto	*com.enviosya. cliente.dto*
ClienteBean	*com.enviosya.cliente.dto*
DatoIncorrectoExcepcion	*com.enviosya.cliente.excepciones*
EntidadNoExisteException	*com.enviosya.cliente.excepciones*
EntidadYaExisteException	*com.enviosya.cliente.excepciones*
Envio	*com.enviosya.envio.dominio*
EnvioDto	*com.enviosya.envio.dto*
EnvioMensaje	*com.enviosya.envio.dto*
EnvioBean	*com.enviosya.envio.sessionbean*
DatoIncorrectoExcepcion	*com.enviosya.envio.excepciones*
EntidadNoExisteException	*com.enviosya.envio.excepciones*
EntidadYaExisteException	*com.enviosya.envio.excepciones*

ErrorUrlException	*com.enviosya.envio.excepciones*
JmsExcepcion	*com.enviosya.envio.excepciones*
Comentario	*com.enviosya.comentario.dominio*
ComentarioDto	*com.enviosya.comentario.dto*
ComentarioBean	*com.enviosya.comentario.sessionbean*
DatoIncorrectoExcepcion	*com.enviosya.comentario.excepciones*
EntidadNoExisteException	*com.enviosya.comentario.excepciones*
EntidadYaExisteException	*com.enviosya.comentario.excepciones*
ErrorUrlException	*com.enviosya.comentario.excepciones*
EnvioMensaje	*com.enviosya.notificaciones*
Mail	*com.enviosya.notificaciones*
NotificarCadete	*com.enviosya.notificaciones*
NotificarCliente	*com.enviosya.notificaciones*
SeguridadBean	*com.enviosya.seguridad.sessionbean*
SesionBean	*com.enviosya.seguridad.sessionbean*
CifradoBean	*com.enviosya.seguridad.sessionbean*

4.6 Decisiones de diseño

Para garantizar un nivel de seguridad aceptable diseñamos un módulo de Seguridad que se encarga de validar que cada acción es realizada por un usuario registrado y con permisos para efectuar la misma. De esta forma se cumplen tácticas de resistencia de ataques, autenticando y autorizando cada usuario. Para esto se utiliza un mecanismo de asignación de un *token* (generado mediante encriptación MD5) único para cada usuario, el cual se valida cada vez que acceda al sistema.

Contar con un sistema modularizado que verifica las credenciales del usuario en cada funcionalidad, aumenta el grado de seguridad del sistema. Se aplica la táctica de separación de entidades.

Para registrar la información de accesos de usuario se implementó un sistema de log de cada registro, de forma de poder realizar auditorías y detectar el ingreso de usuarios no autorizados.

Por otro lado, se implementó el manejo de excepciones diseñando excepciones propias para cada módulo, de forma de poder detectar e informar las fallas pertinentes para cada uno. Descartamos la realización de un módulo independiente con todas las excepciones (donde los demás módulos lo implementarían) dado a que estarían ligados a excepciones que no son relevantes a su función.

Por otra parte, se contemplaron ciertas consideraciones de eficiencia a nivel de programación para lograr el menor tiempo de respuesta posible en cada solicitud y así favorecer la latencia:

- Uso de dtos para la comunicación en entre la interfaz y el dominio.

- Identificación de objetos por ID para reducir la latencia en comunicaciones.
- Utilización de StringBuilder para concatenar strings.

5. Vistas de Asignación

En esta sección se muestra únicamente la Vista de Despliegue ya que el la vista que nos parece más relevante presentar. No se desarrollaron la Vista de Instalación ni la de Asignación de recursos debido al pequeño tamaño de nuestra aplicación.

5.1 Vista de Despliegue

A partir de esta vista se aprecia cómo se comunican los diferentes elementos de software así como los servidores, los conectores y los componentes. Esta vista permite visualizar dónde se ejecutan los componentes del software.

5.2 Representación primaria

El usuario web se comunica desde el browser, que se encuentra en su PC a través de internet, con el servidor físico, el cual contiene al servidor *Glassfish*.

En el servidor *Glassfish* se encuentran deployados cada uno de módulos presentados anteriormente, junto con el componente *Derby* y a la propia base de datos.

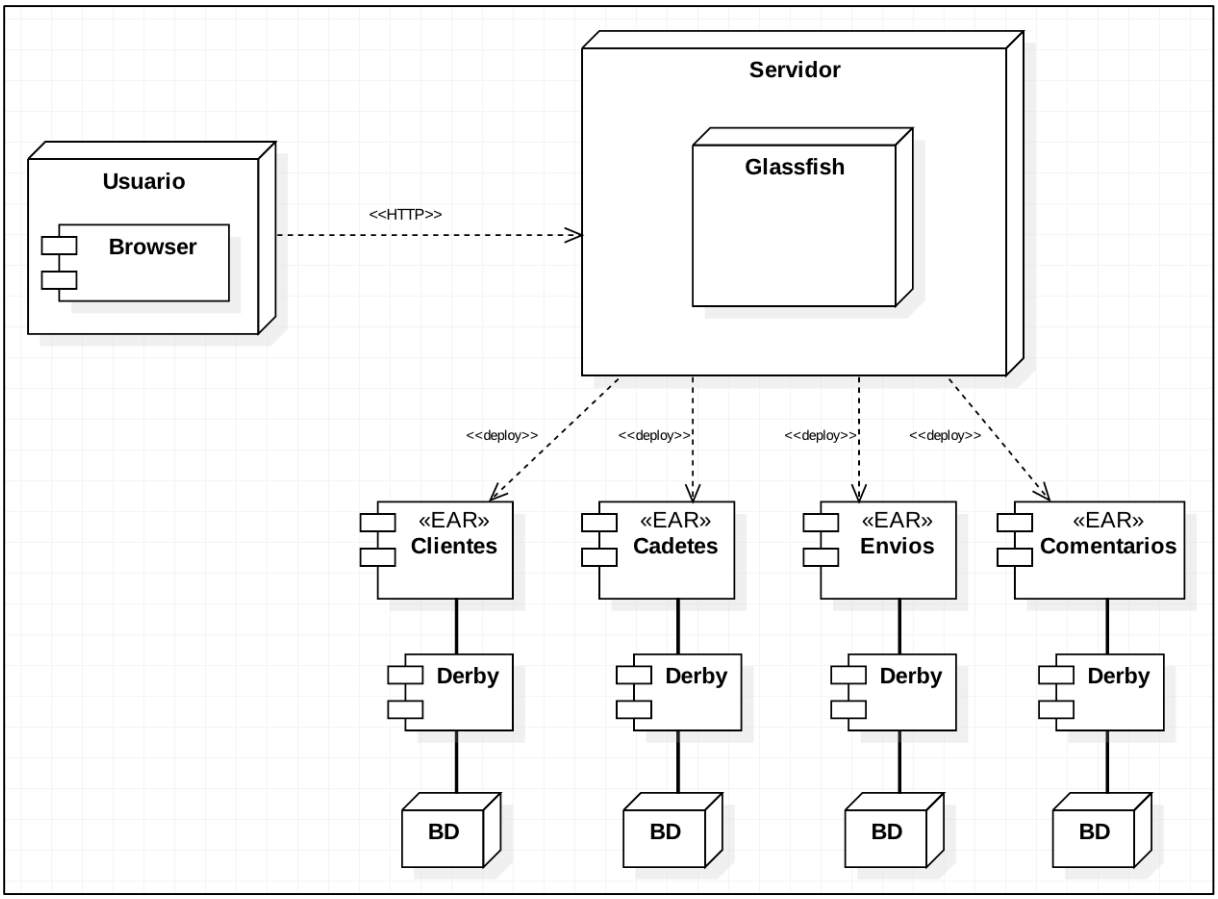

Figura 7: Vista de Despliegue

5.3 Catálogo de elementos

Componente	Descripción
Browser	Componente que utiliza el usuario para comunicarse con el servidor.
Derby	Componente que permite el manejo de la base de datos.
BD	Componente donde se almacenan físicamente los datos del sistema.
Servidor Glassfish	Ambiente en el cual corren los componentes del servidor.
Servidor físico	Máquina física en la cual se encuentra instalada la arquitectura mencionada en la representación.
PC	Máquina física en la cual el usuario hace uso del browser.

5.4 Decisiones de diseño

La utilización del servidor *Glassfish* fue elegido en base a las recomendaciones del curso acerca de este en la tecnología JEE.

Por otro lado, en cuando al manejador de base de datos, se seleccionó Derby por las mismas razones, sumado a la facilidad que brinda para crear los recursos de acceso a datos.

Para cumplir con el requerimiento de que cada módulo debe gestionar su propio acceso a datos se diseñó un base de datos para cada uno. Las mismas pueden ser cambiadas.

6. Librerías utilizadas

Se utilizaron las siguientes librerías para la implementación de la solución planteada.

- **Gson**: esta librería permitió la utilización del tipo de datos *json* en nuestro sistema. Se utilizó para decodificar el *json* obtenido de las requests, y también para codificar las respuestas.
- **Json**: se utilizó para devolver datos de un buffer que lee datos de un http.
- **Log4j**: librería para el log de errores.
- **EnviosYALogger**: librería propia creada para manejar el log de datos.

7. Requerimiento no implementados y errores conocidos

RNF 2 - Tiempo de respuesta de creación de una Review

- No se implementa.

RF 3 - Registrar una solicitud de envío

- No se calculan los cuatro cadetes más cercanos. Se retornan los primeros cuatro cadetes de la tabla Cadete.
- La foto del paquete es un hard code en base 64 de una imagen enviada al servicio REST provisto por los profesores. No se implementó seleccionar una imagen desde Postman.

RF 5 - Calificar el servicio y el Cadete

- La llamada REST a Natlang no está implementada, sino que se simula con es un método que crea un valor sentimental para el comentario basado en la cantidad de palabras del mismo.
- La llamada POST a Notificaciones no está implementada. Se pensó agregar los

comentarios una cola de mensajes que es escuchada en el módulo Notificaciones.

RNF 3 - Usuarios registrados - RNF 7 - Protección de los datos de los clientes.

- No se implementó el módulo Seguridad.

RNF 8 - Información de auditoría.

El diseño inicial cuenta con la idea de tener una libraría EnviosYaLogger.jar que los demás módulos implementarían, y de esta forma manejar los logs de errores y accesos. Como esto no funcionó como esperábamos, utilizamos un log para cada módulo. El archivo de log4j.properties lo colocamos en la carpeta de glassfish/domains/domain1/lib/config y de esta forma se genera un archivo logEnviosYa.out con todos los registros del log.

RNF 10 - Modificación del cálculo de costos de los envíos

- No se implementó el cálculo en un módulo separado, se hace en el módulo Envío.

8. Cómo y dónde se instala la aplicación

Para el funcionamiento de la aplicación, debemos escribir los comandos presentados a continuación, accediendo a la consola de "*asadmin*" en la ubicación "*C:\Program Files\glassfish-4.1\bin*".

- **Queues***:*

create-jms-resource --restype javax.jms.Queue --property Name=QueueCliente jms/QueueCliente

create-jms-resource --restype javax.jms.Queue --property Name=QueueCliente jms/QueueCadete

- **Pools***:*

create-jdbc-connection-pool --datasourceclassname org.apache.derby.jdbc.ClientDataSource --restype javax.sql.DataSource --property portNumber=1527:password=envio:user=envio:serverName=localhost:databaseName=envio:connectionAttributes=\;create\\=true envios_pool

create-jdbc-resource --connectionpoolid envios_pool jdbc/envios_pool

---

create-jdbc-connection-pool --datasourceclassname org.apache.derby.jdbc.ClientDataSource --restype javax.sql.DataSource --property portNumber=1527:password=cliente:user=cliente:serverName=localhost:databaseName=cliente:connectionAttributes=\;create\\=true clientes_pool

create-jdbc-resource --connectionpoolid clientes_pool jdbc/clientes_pool

---

create-jdbc-connection-pool --datasourceclassname org.apache.derby.jdbc.ClientDataSource --restype javax.sql.DataSource --property portNumber=1527:password=cadete:user=cadete:serverName=localhost:databaseName=cadete:connectionAttributes=\;create\\=true cadetes_pool

create-jdbc-resource --connectionpoolid comentarios_pool jdbc/cadetes_pool

---

create-jdbc-connection-pool --datasourceclassname org.apache.derby.jdbc.ClientDataSource --restype javax.sql.DataSource --property portNumber=1527:password=comentario:user=comentario:serverName=localhost:databaseName=comentario:connectionAttributes=\;create\\=true comentario_pool

create-jdbc-resource --connectionpoolid comentario_pool jdbc/comentario_pool

- ***Log***

El archivo de log4j.properties va en la ruta glassfish/domains/domain1/lib/config.

9. Datos de prueba

A continuación se incluyen datos de prueba para poder utilizar las demás funcionalidades del sistema.

@POST
http://localhost:8080/Clientes-war/cliente

```
{
        "cedula":"46104824",
        "nombre":"Emily",
        "apellido":"Symonds",
        "email":"emilysymondsc@hotmail.com"
}
{
        "cedula":"56676654",
        "nombre":"Diego",
        "apellido":"Carrion",
        "email":"diegcarrion@gmail.com"
}
```

@POST
http://localhost:8080/Cadetes-war/cadete

```
{
        "cedula":"478981",
        "nombre":"Martin",
        "apellido":"Perez",
        "email":"martinperez@gmail.com"
}
{
        "cedula":"88899988",
        "nombre":"Guillermo",
        "apellido":"Flores",
        "email":"gflore@gmail.com"
}
```

@POST
http://localhost:8080/Envios-war/envio/crear

```
{
  "descripcion": "Envio de paquete a Rio Negro",
  "clienteEmisor": "101",
```

```
  "clienteReceptor": "102",
  "formaPago": "1",
  "direccionRetiro": "Domingo Bazurro 7046, Montevideo",
  "direccionRecibo": "Asencio 1978, Young, Rio Negro",
  "fotoPaquete": "fotoPaquete12345678"
}
```

@POST
http://localhost:8080/Envios-war/envio/confirmar

```
{
  "envio":"251",
  "cadete":"301"
}
```

@POST
http://localhost:8080/Envios-war/envio/recepcion?id=251

@POST
http://localhost:8080/Comentarios-war/comentario/crear

```
{
    "rating": "4",
    "comentario": "Buenisimo servicio. Gracias, los voy a volver a contratar.",
    "envio": "251",
    "autor": "101"
}
```