**DOCUMENTO RESUMIDO - DESCRIPCION DE ARQUITECTURA**

***Obligatorio Arquitectura de Software***

***27-11-2014***

***143904 Richard Caetano***

***156773 Fernando Godoy***

**Índice**

[1. Introducción 3](#_Toc404802812)

[1.1 Propósito 3](#_Toc404802813)

[2. Antecedentes 3](#_Toc404802814)

[2.1 Propósito del sistema 3](#_Toc404802815)

[2.2 Requerimientos significativos de Arquitectura 3](#_Toc404802816)

[2.2.1 Resumen de Requerimientos Funcionales 3](#_Toc404802817)

[2.2.2 Resumen de Requerimientos No Funcionales 3](#_Toc404802818)

[3. Documentación de la arquitectura 5](#_Toc404802819)

[3.1 Vistas de Módulos 5](#_Toc404802820)

[3.1.1 Vista de Paquetes 5](#_Toc404802821)

[3.1.1.1 Representación primaria 5](#_Toc404802822)

[3.1.1.2 Catálogo de Elementos 6](#_Toc404802823)

[3.1.1.3 Decisiones de Diseño 6](#_Toc404802824)

[3.1.2 Vista de Layers 2](#_Toc404802825)

[3.1.2.1 Representación primaria 2](#_Toc404802826)

[3.1.2.2 Decisiones de diseño 2](#_Toc404802827)

[3.1.3 Comportamiento 1](#_Toc404802828)

[3.2 Vistas de Componentes y conectores 1](#_Toc404802829)

[3.2.1 Diagrama de componentes y conectores 1](#_Toc404802830)

[3.2.1.1 Representación primaria 1](#_Toc404802831)

[3.2.1.2 Catálogo de elementos 1](#_Toc404802832)

[3.2.1.3 Decisiones de diseño 2](#_Toc404802833)

[3.3 Vistas de Asignación 2](#_Toc404802834)

[3.3.1 Vista de Despliegue 3](#_Toc404802835)

[3.3.1.1 Representación primaria 3](#_Toc404802836)

# Introducción

Propósito

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

# Antecedentes

Propósito del sistema

TuRecetaAlToque se trata de una aplicación que permite a esa persona que llega a su hogar y no sabe que puede cocinar, ingresar los ingredientes principales que tiene a su alcance en la cocina y le devolverá una lista de recetas (ordenadas por valoración) que puede realizar con esos ingredientes.

Se quiere que las recetas se puedan consultar de múltiples plataformas, como ser teléfonos celulares y navegadores. El registro de recetas debe poder realizarse de manera asíncrona.

Los interesados en esta gran base de datos de recetas y servicios para consultarlas, son los responsables de elGourmetInc.com que se encargaran de desarrollar las distintas interfaces para cada usuario potencial.

Requerimientos significativos de Arquitectura

### Resumen de Requerimientos Funcionales

|  |  |
| --- | --- |
| **ID Requerimiento** | **Descripción** |
| 1. Registro de usuario | *Permite registrar un usuario brindando algunos datos básicos* |
| 2. Ingreso al sistema | *Permite a un usuario registrado, activar una sesión por 5 minutos* |
| 3. Buscar por ingredientes | *Permite a un usuario activo, realizar una búsqueda de recetas ingresando hasta 4 ingredientes básicos* |
| 4. Subir receta | *Permite a un usuario con una sesión activa, agregar una receta al sistema* |
| 5. Valorar receta | *Permite a un usuario con una sesión activa, agregar una valoración a una receta existente* |
| 6. Top 10 búsquedas | *Permite obtener la lista de ingredientes más buscados* |
| 7. Top 10 usuarios mejor valorados | *Permite obtener la lista de los usuarios con mejores valoraciones en sus recetas* |

### Resumen de Requerimientos No Funcionales

**Interoperabilidad:**

RNF 1 -Multiplataforma, la aplicación podrá ser consumida por distintos clientes para el registro/ingreso de usuarios y búsqueda de recetas

**Performance**:

RNF 2 -La funcionalidad de subir y valorar receta será asíncrona

RNF 3 - El tiempo de respuesta es importante, se espera que los usuarios reciban el resultado de una consulta en menos de 5 segundos. Las otras operaciones no tienen requerimientos de velocidad de respuesta, aunque deben ser aceptables teniendo como base cualquier aplicación web del mercado.

RNF 4 - Se espera que la aplicación sea usada desde dispositivos móviles, por lo tanto el formato de intercambio de datos deberá ser liviano

**Modificabilidad**:

RNF 5 - Es importante que la aplicación sea independiente del motor de base de datos

RNF 6 - Se debe desarrollar de manera evolutiva/incremental

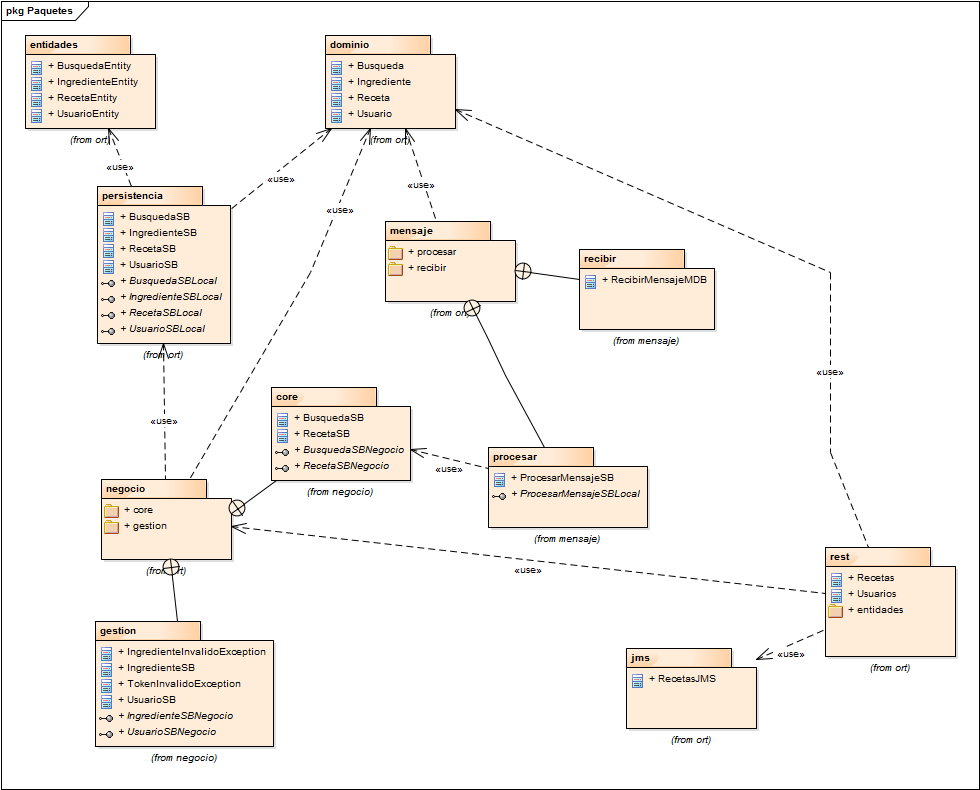
RNF 7 - La comunicación de la aplicación siempre se dará a través del protocolo HTTP pero es importante que la aplicación pueda comunicarse con distintos dispositivos.

# Documentación de la arquitectura

Vistas de Módulos

### Vista de Paquetes

#### Representación primaria



En este diagrama se representan todos los paquetes del sistema, su descomposición y la relación de uso que tienen entre ellos.

#### Catálogo de Elementos

|  |  |
| --- | --- |
| Paquete | Descripción |
| Dominio | Definición de los DTO del sistema. |
| Persistencia | Paquete encargado de la lógica de persistencia de datos, expone una interfaz para abstraer su implementación. |
| Entidades | Paquete con la definición de las entidades a persistir. |
| negocio.gestion | Se encarga de la lógica de negocio administrativa, usuarios e ingredientes. |
| negocio.core | Encargado de la lógica de negocio de búsquedas, alta y valoración de recetas. |
| mensaje.procesar | Encargado de la lógica detrás del procesamiento de un mensaje, brinda una interfaz al recibir. |
| mensaje.recibir | Encargado de la recepción del mensaje y lo pasa al procesar a través de la interfaz usando un callback. |
| Rest | Definición de los servicios REST/JSON expuestos. |
| rest.entidades | Paquete que contiene la entidades que se mapean con los JSONs que se usan en los servicios REST de la aplicación. |

|  |  |
| --- | --- |
| Interfaz | Descripción |
| BusquedaSBLocal | Interfaz para abstraer la implementación de la persistencia y recuperación de búsquedas. |
| IngredienteSBLocal | Interfaz para abstraer la implementación de la persistencia y recuperación de ingredientes. |
| RecetaSBLocal | Interfaz para abstraer la implementación de la persistencia y recuperación de recetas. |
| UsuarioSBLocal | Interfaz para abstraer la implementación de la persistencia y recuperación de usuarios. |
| IngredienteSBNegocio | Interfaz del paquete de negocio para abstraer la implementación de la lógica de ingredientes. |
| UsuarioSBNegocio | Interfaz del paquete de negocio para abstraer la implementación de la lógica de usuarios. |
| BusquedaSBNegocio | Interfaz del paquete de negocio para abstraer la implementación de la lógica de las búsquedas. |
| RecetaSBNegocio | Interfaz del paquete de negocio para abstraer la implementación de la lógica de las recetas. |
| ProcesarMensajeSBLocal | Interfaz para abstraer la implementación del procesamiento de mensajes |

#### Decisiones de Diseño

En esta vista podemos observar algunas de las decisiones que fueron tomadas para favorecer los requerimientos de Modificabilidad. Para esto se separan los módulos incrementando la coherencia semántica, restringiendo dependencias de modo de evitar ciclos y agregando intermediarios, como por ejemplo el uso del paquete de negocio intermediario entre los servicios expuestos y la persistencia. La comunicación entre módulos se realiza a través de interfaces, de modo de abstraer su implementación y minimizar el impacto de los cambios.

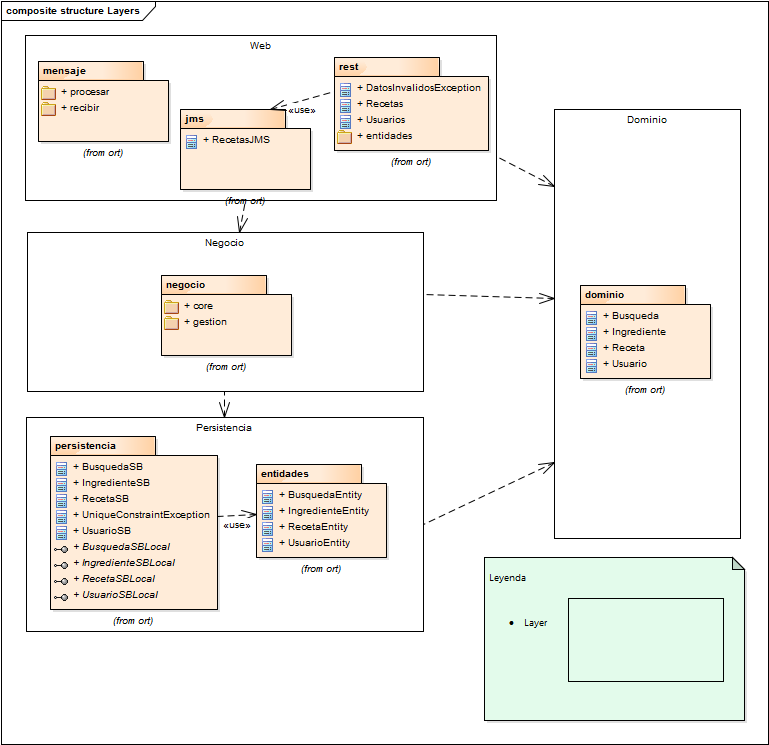
Se decidió implementar el punto de entrada de la aplicación (paquete rest) como una api RESTFUL por la simplicidad y extensibilidad de dicha técnica de arquitectura de software. La facilidad de REST se debe a que su comunicación se base en http con xml, formatos con un uso muy extendido en estos tiempos. De esta manera estamos cumpliendo con el RNF1 (la aplicación podrá ser consumida por distintos clientes) ya que no estamos atados a ningún lenguaje de programación o formato de comunicación propietario, cualquier aplicación que pueda comunicarse por http podrá ser nuestro cliente.

El hecho de utilizar JSON se debe a que es un formato con un overhead muy bajo y permite cumplir con el RNF4 (el formato de intercambio debe ser liviano para permitir una conexión fluida con dispositivos móviles).

### Vista de Layers

Podemos observar en la siguiente vista la disposición de los paquetes en las 4 capas lógicas definidas para esta solución.

#### Representación primaria



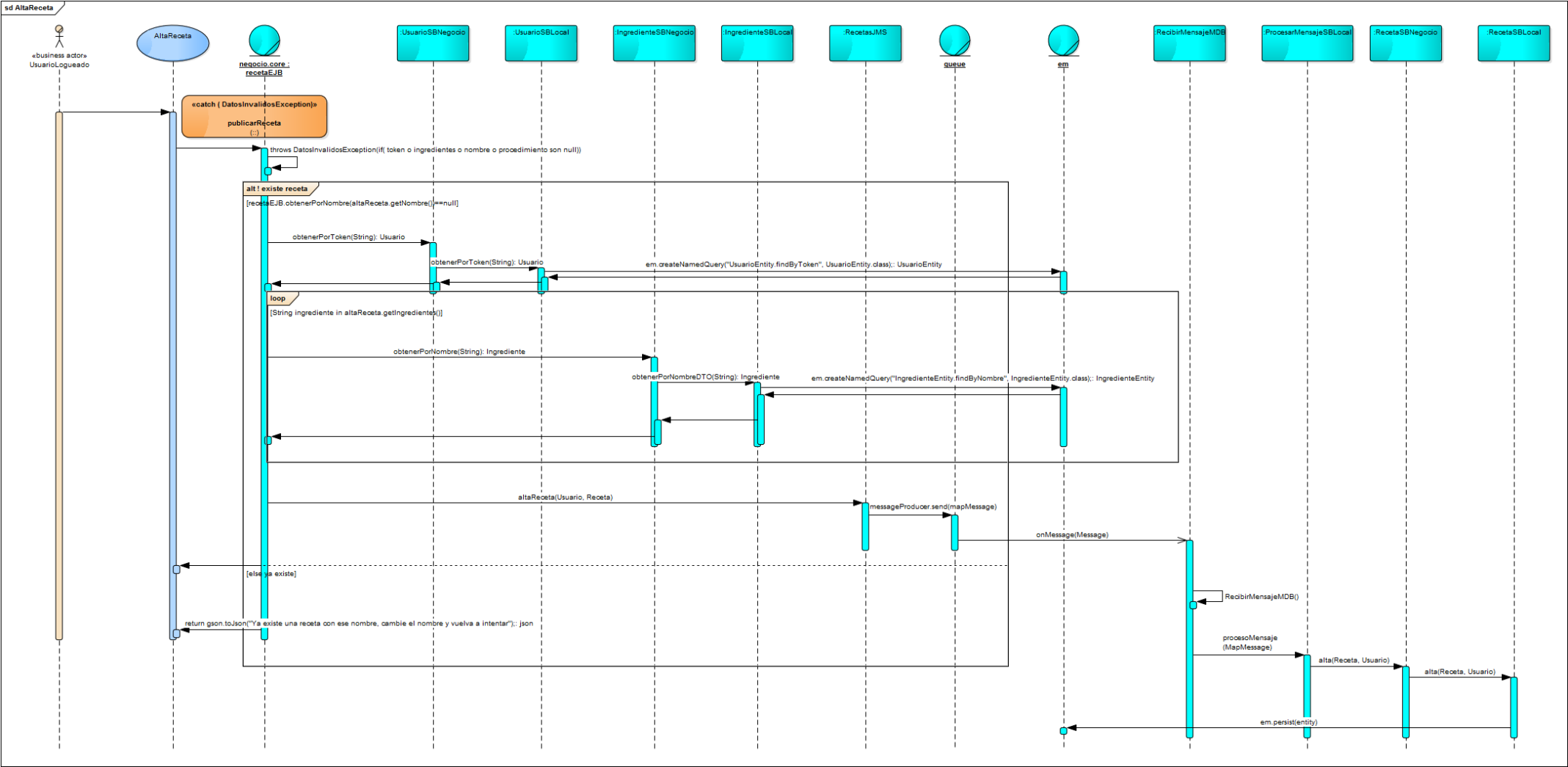
#### Decisiones de diseño

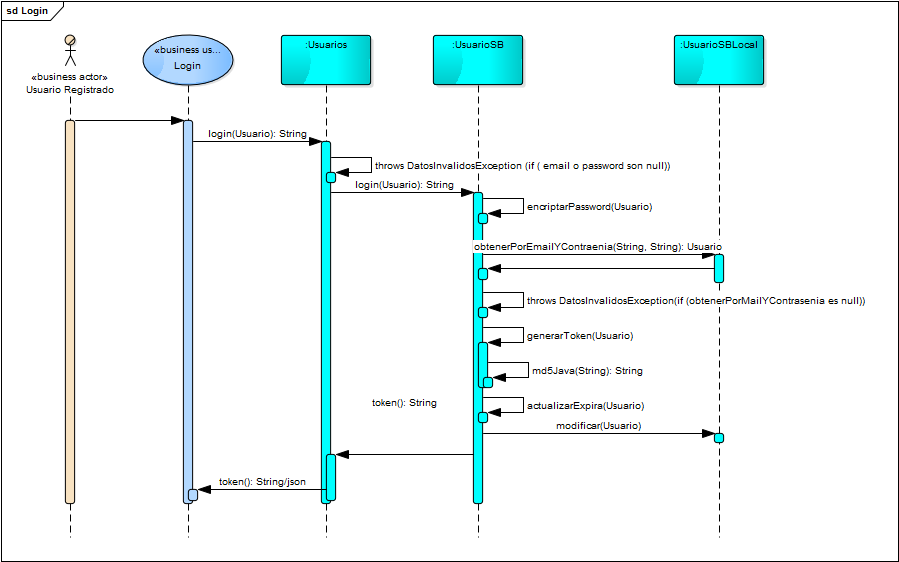
Utilizamos el patrón de capas (Layers) para beneficiar la eficiencia y la disponibilidad, soportando el reúso.

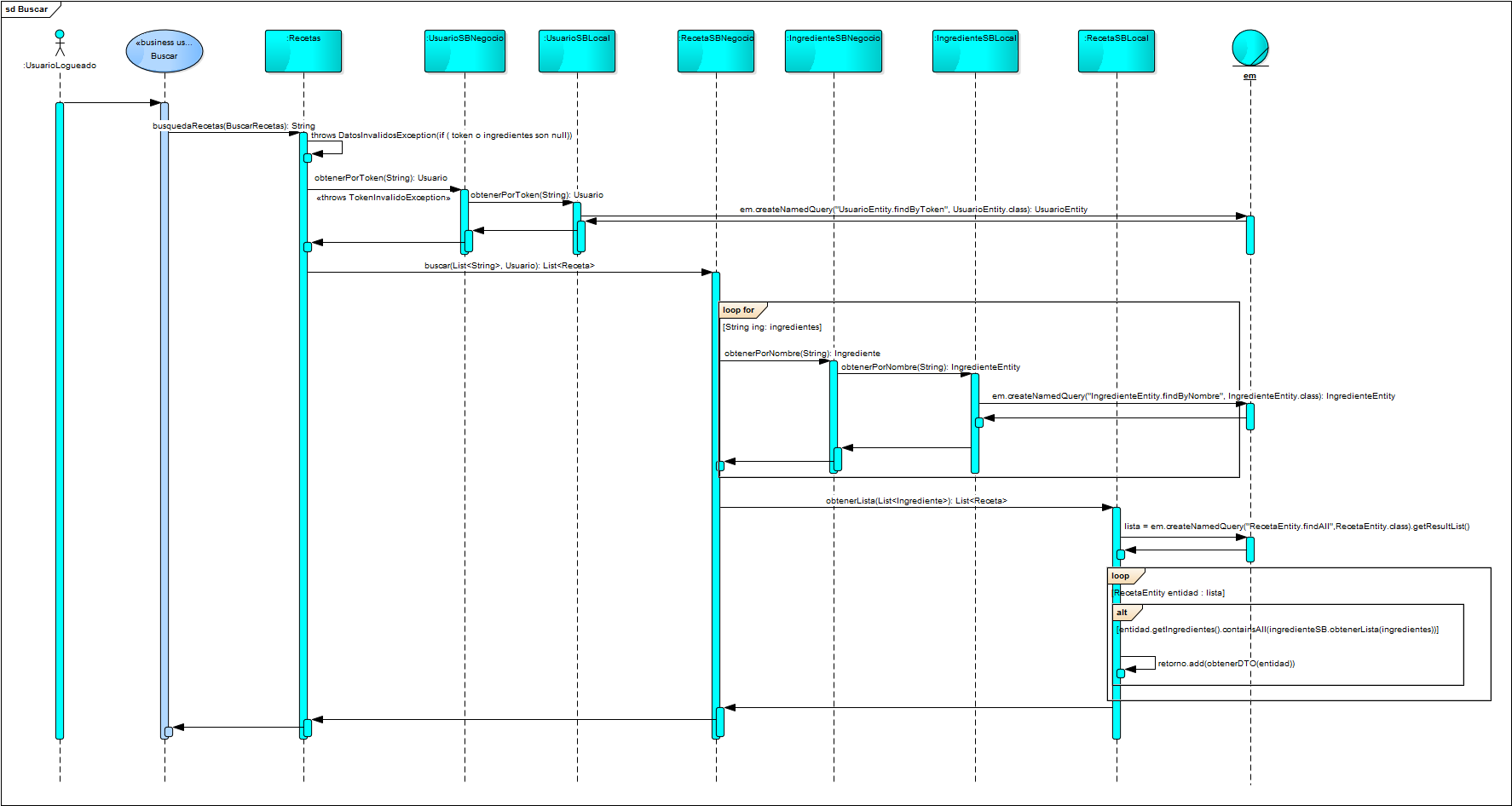
Se dividió la aplicación en 3 capas:

* Servicios y Mensajería: se encarga de manejar toda comunicación externa a la aplicación. Aquí se exponen los servicios REST y la comunicación con las colas de mensajes. Si en un futuro se desea cambiar la forma en que la aplicación se comunica con otras aplicaciones solo se debe modificar esta capa.
* Negocio: esta capa se encarga de la lógica de la aplicación, sabe que datos se deben validar y se comunica con la capa de persistencia para guardar los datos resultantes de sus funcionalidades.
* Persistencia: se encarga de manejar todo el acceso a datos

### Comportamiento



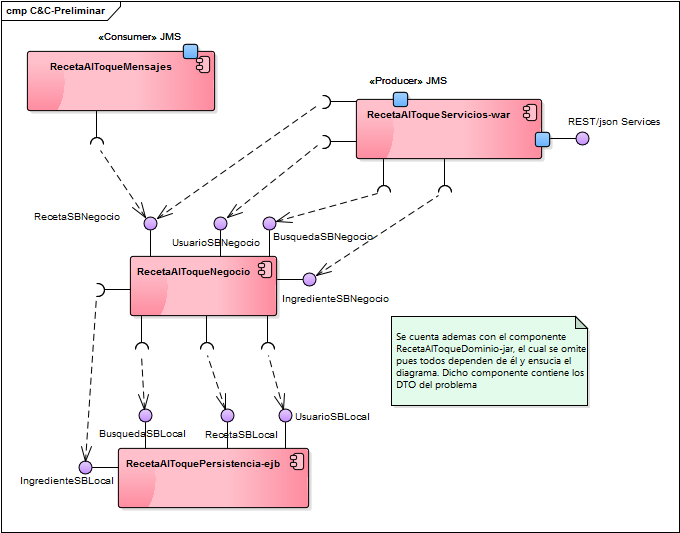




Vistas de Componentes y conectores

### Diagrama de componentes y conectores

#### Representación primaria



#### Catálogo de elementos

|  |  |
| --- | --- |
| Elemento | Descripción |
| RecetaAlToqueNegocio | EJB compuesto de varios SessionBeans con interface local que encapsulan la lógica del negocio. Contiene los paquetes: core y gestión. |
| RecetaAlToquePersistencia-ejb | EJB compuesto de varios SessionBeans con interface local que encapsulan el mecanismo de persistencia. Contiene los paquetes: entidades y persistencia. |
| RecetaAlToqueServicios-war | WAR que presenta a los clientes las operaciones disponibles, cuenta además con un productor de JMS para las operaciones asíncronas. Contiene los paquetes: rest y jms. |
| RecetaAlToqueMensajes | EJB encargado de la recepción y procesamiento de mensajería JMS, requiere una queue TRAQueue definida. Contiene los paquetes procesar y recibir. |

Las interfaces expuestas y consumidas fueron descritas en la vista de paquetes, corresponden a interfaces locales definidas por cada SessionBean.

#### Decisiones de diseño

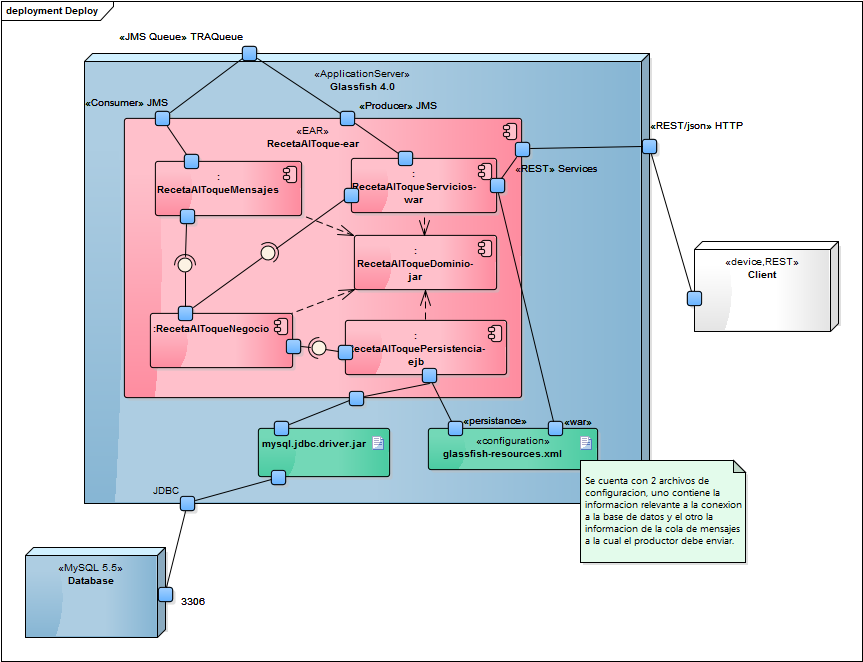
La separación de los distintos módulos en componentes se realizó según sus responsabilidades, de modo de favorecer la modificabilidad.

Vistas de Asignación

Se decide implementar esta solución con 3 capas físicas, cliente, servidor y base de datos. Donde lo que refiere a la implantación se trata de un ear conteniendo cada componente necesario para ejecutar la aplicación.

### Vista de Despliegue

#### Representación primaria



#### Catálogo de Elementos

|  |  |
| --- | --- |
| Elemento | Descripción |
| TRAQueue | Cola de mensajes utilizada para las operaciones asíncronas |
| Glassfish 4.0 | ApplicationServer utilizado |
| RecetaAlToque-ear | EAR que empaqueta la solución |
| RecetaAlToqueNegocio | Componente EJB que agrupa los módulos de la lógica de negocio |
| RecetaAlToquePersistencia-ejb | Componente EJB con los módulos encargados de la persistencia |
| RecetaAlToqueServicios-war | Componente WAR con los módulos de servicios REST y productor de JMS, expone las operaciones a los clientes |
| RecetaAlToqueDominio | Componente JAR con los DTO del problema |
| RecetaAlToqueMensajes | Componente EJB con los módulos encargados del consumidor de mensajería |
| mysql.jdbc.driver | Artefacto, es el driver jdbc utilizado para la comunicación con la base de datos |
| glassfish-resources.xml | Artefacto, archivo de configuración |
| Database | Host de base de datos mysql |
| ClienteRest | Host cliente, no es entregado en esta solución, pero forma parte del problema |

#### Decisiones de Diseño

Dados los varios requerimientos de modificabilidad expuestos y las tácticas aplicadas para cumplirlos, se ve notoriamente afectada la eficiencia. Utilizando la táctica de reducir el overhead se plantea una solución simple con 3 capas físicas, donde además del cliente y la base de datos nuestra aplicación se empaqueta en un ear para atacar el requerimiento de performance reduciendo el costo creado por la solución modificable. En caso de crecimiento de la aplicación, se podrá inicialmente utilizar la táctica de incrementar recursos.

#### Guías de Variabilidad

La eficiencia también podría mejorarse utilizando múltiples copias de cómputo, para lograrlo en esta solución se deberán modificar los componentes de servicios-war y mensajes para que utilicen sessionbeans con interfaces remotas del negocio en vez de locales y el componente de negocio para que sus sessionbeans expongan interfaces remotas en vez de locales. De esta manera seria posible pasar a una solución en 4 capas físicas separando los componentes de servicios y mensajes de los componentes de negocio y persistencia.

La posibilidad de cambiar de motor de persistencia de base de datos por uno nosql, en caso de crecimiento de la aplicación, también forma parte de los requerimientos y para esto se debe modificar el componente de persistencia, re-implementando tanto las entidades como la implementación de las interfaces locales. Un cambio relativamente sencillo se puede lograr pasando a MongoDB ya que JPA agrega soporte nativo tanto para MongoDB como para OracleNoSQL. Las características de ambas implementaciones según la wiki.eclipselink y el claro uso masivo y madurez de MongoDB hacen que en caso de pasar a una solución noSQL la opción sea MongoDB.