Enero | 09

Trabajo fin de carrera:

Implementación de un Sistema de Reservas para una Agencia de Viajes usando J2EE y prácticas de Desarrollo Ágil

Nombre del Estudiante
Daniel Blanch Bataller

Nombre del Consultor
Oscar Escudero Sanchez

Fecha de entrega
14/01/2009

Univesitat Oberta de Catalunya [Ingeniería Técnica en Infomática de Gestion]

Tabla de contenido

1. **Introducción** 4
 1.1. **Justificación del proyecto** 4
 1.2. **Objetivos del trabajo** 5
  1.2.1. Objetivos técnicos 5
  1.2.2. Objetivos metodologicos 6
 1.3. **Enfoque metodológico** 6
  1.3.1. Manifiesto ágil 7
  1.3.2. Aplicación de la metodología Ágil en nuestro proyecto. 8
 1.4. **Planificación del proyecto** 11
  1.4.1. Investigación tecnología j2ee y posgtreSql 11
  1.4.2. Instalacion de entorno 11
  1.4.3. Desarrollo de la aplicación por casos de uso (sprints) 11
  1.4.4. Ciclo de cada Sprint: 12
  1.4.5. Diagrama de Gannt 12
 1.5. **Productos obtenidos** 13
  1.5.1. Metodología y herramientas de testing 13
  1.5.2. Entrega del producto 17
2. **Especificación y requerimientos** 24
 2.1. **Información inicial** 24
 2.2. **Glosario** 24
 2.3. **Modelo del dominio** 25
 2.4. **Diagrama de casos de uso** 26
 2.5. **Documentación textual de los casos de uso** 26
  Explicación del modelo seguido para documentar los casos de uso 26
  2.5.1. Búsqueda de ofertas de paquetes turísticos 27
  2.5.2. Reserva de plazas de paquetes turísticos 28
  2.5.3. Login 30
  2.5.4. Mantenimiento cuenta de usuario 31
  2.5.5. Mantenimiento de ofertas 34
 2.6. **Requisitos de la interfaz de usuario** 36
3. **Análisis** 36
 3.1. **Identificación de las clases de entidades y sus atributos** 36
 3.2. **Diagramas de estados** 37
 3.3. **Diagramas de actividades** 37
  3.3.1. Transacción de reserva de plaza 37
  3.3.2. Reserva de plaza (búsqueda + reserva) 38
  3.3.3. Login 39
  3.3.4. Alta de usuario 40
  3.3.5. Integración (búsqueda, Login, alta, reserva) 41
 3.4. **Modelo de datos** 41
4. **Diseño técnico** 42
 4.1. **Vista-Controlador: Validator y Parser helpers** 42
 4.2. **El modelo** 44
  4.2.1. La Fachada 44
5. **Implementación** 46
 5.1. **Herramientas de desarrollo** 46
 5.2. **Proceso de instalación** 47

5.3. Juego de pruebas **47**

6. Conclusiones **51**

Uso de metodología ágil **51**

Ciclos de desarrollo cortos 51

Pruebas de unidad automatizadas 51

Metodología de tests y herramientas. **52**

7. Bibliografia **52**

1. Introducción

La aplicación de metodologías y prácticas ágiles dentro de empresas dedicadas al desarrollo de software, es en la actualidad una realidad que se viene acrecentando en los últimos años alrededor del mundo.

Son muchas las personas y empresas dedicadas al desarrollo de software que se enfrentan hoy por hoy con el dilema de ser o no ser ágiles. Bien sea porque sus actuales metodologías de desarrollo no dan los resultados esperados o bien porque desean incursionar en prácticas que están de moda a lo largo y ancho de las comunidades de desarrollo en todo el mundo, lo cierto es que el auge de las metodologías ágiles es un tema que apasiona y hace reflexionar a todo aquel que se encuentre inmerso dentro del proceso de desarrollo de software.

Este trabajo presenta un enfoque práctico para la elección y adecuación de metodologías ágiles de desarrollo de software a un proyecto real: El desarrollo de un sistema de reservas para una agencia de viajes usando tecnología J2EE.

1.1. Justificación del proyecto

El sector de las Agencias de Viaje ha vivido una crisis desde la aparición de Internet.

Según el Foro Internacional de Turismo, las compras de viajes por Internet han crecido de forma significativa hasta el punto de llevar a la crisis a las agencias de viaje tradicionales. La tendencia se presenta como consolidada aunque ello no tiene por qué suponer la desaparición de estas agencias sino un cambio en su concepción original de manera que se puedan adaptar a los nuevos tiempos.

Las compañías aéreas y las centrales de reservas de hoteles fueron algunas de las primeras entidades en usar grandes sistemas en red para gestionar la reserva y venta de sus productos. Estos sistemas conectaban los ordenadores centrales con los terminales de las Agencias de Viajes. Éstas por su parte vendían luego el producto al público final llevándose una comisión por la gestión.

La incorporación de las agencias a la red ha supuesto la liberalización masiva del sector, y cualquier competencia en la oferta es beneficiosa para el cliente ya que la competitividad conlleva una bajada de precios y una mejora del producto.

Con la aparición de Internet las compañías aéreas y las centrales de reservas de hoteles han extendido sus redes de computadores que antes únicamente servían a agencias de viajes. En muchos casos hoy en día sale mas barato comprar un billete por Internet, que en una Agencia. Y las compañías aéreas recortan cada vez más sus comisiones a las Agencias.

Las compañías de viajes (ferroviarias, aéreas, de alquiler de coches...), los hoteles, guías y otros muchos servicios que antes dependían de una agencia u organizador de viajes, ahora pueden ofertarse directamente en la red, sin intermediarios y sin monopolios que limiten su actividad comercial, de manera que se ponen en manos de una buena gestión para ofrecer la mejor opción al cliente y del marketing.

Internet es donde se compra, vende y se contratan la inmensa mayoría de los viajes, cualquiera con conexión puede hacerlo con cuatro clicks.

Las Agencias de viajes están pues en crisis, necesitan reinventarse, ya no pueden ser intermediarios en la venta de billetes de proveedores. Necesitan poder competir en este nuevo mercado de venta por Internet.

El producto diferenciado que ofrecen las agencias de viajes es el paquete turístico, que para poder ser un producto alternativo a las ofertas de productos individuales de transporte y alojamiento, debe contar con un sistema de reservas por Internet que supere al de las compañías aéreas y centrales hoteleras, un sistema de reservas que ofrezca al público los productos que las agencias elaboran.

El sistema de reservas que elaboraremos en este proyecto contará con una ventaja competitiva frente a los sistemas de las grandes mayoristas tradicionales. Nuestra tecnología estará basada en la potencia de Java y J2EE y utilizará metodologías ágiles apoyadas en tests automatizados con las que podremos ser más dinámicos frente a los cambios que puedan surgir en el futuro.

1.2. Objetivos del trabajo

Con este proyecto pretendemos explorar tanto objetivos técnicos como metodológicos.

1.2.1. Objetivos técnicos

Para poder competir en este nuevo mercado de venta de viajes por Internet, necesitamos por una parte que la aplicación sea extremadamente **rápida** en la gestión de reservas de paquetes túrisiticos. Para ello debemos limitar al máximo la contención en las transacciones y asegurar al mismo tiempo su corrección, o lo que es lo mismo, impedir interferencias con otras transacciones en curso por ausencia de bloqueos.

Por otro lado, también será deseable que la aplicación sea **testable** con facilidad, para ello construiremos una arquitectura en la que los objetos de negocio se puedan ejecutar tanto fuera como dentro del contenedor J2EE, para así poderlo someter a todas las pruebas necesarias, sin la complicación, y lentitud añadida de arrancar un servidor J2EE.

Otra característica necesaria y derivada de la anterior, es construir una arquitectura basada en el paradigma de la orientacion a objetos, conocidido como **Open / Closed principle**. Este principio postula que el código este abierto para la extension pero cerrado para la modificación ya que cada vez que modifiquemos

código corremos el riesgo de estropearlo. Por ello debemos utilizar las tecnicas de herencia y composición para añadir funcionalidad a nuestros programas.

1.2.2. Objetivos metodologicos

Nosotros en este proyecto, usaremos prácticas de desarrollo ágil como las siguientes:

Frecuente retroalimentacion: para ello se han planificado casos de uso cortos y en la medida de lo posible autocontenidos, es decir que no necesitan de otras partes para ser probados.

Pruebas automatizadas: se comenzará el desarrrollo empezando por las pruebas. Haremos pruebas de unidad y de integración en un ciclo constante como veremos en más detalle más adelante.

1.3. Enfoque metodológico

Para este proyecto exploraremos cómo aplicar las principales prácticas de lo que se conoce como "desarrollo ágil".

El enfoque tradicional de desarrollo de aplicaciones heredado de otras ramas de la ingeniería, ha sido el de hacer un estudio exahustivo del problema, establecer un plan y llevar a cabo la construccion. Es la metodología conocida como desarrollo en cascada, con las fases consecutivas de Analisis, Diseño, Codificacion, Pruebas e Implementación.

Pero este enfoque no siempre es óptimo para el desarrollo de software. La programación no es exactamente comparable a otras ramas de la ingeniería o a la construcción.

El desarrollo de software, es un proceso puramente creativo, análogo a lo que en la construcción de una vivienda sería la elaboración de los planos.

A diferencia de la ingeniería civil, en el desarrollo de software no existe un proceso de construcción en el que se sigan unos planos de modo mecánico previamente establecidos por el arquitecto o ingeniero. En cada fase hay que ser creativo y tomar decisiones.

En realidad sí existe un proceso análogo al de la pura construcción, en el que no hay que tomar decisiones y solo aplicar lo dictado por el arquitecto. Es el paso del código fuente al objeto, la compilación. Pero su coste es despreciable frente al coste del diseño. Justo lo contrario a lo que sucede con la ingeniería civil, donde el coste del diseño es infinitamente menor que el de la construcción.

El objetivo de las metodologías de desarrollo ágil de software es la organización de un trabajo creativo, que suele ser bastante caótico. Se intenta dar prioridad a la

ejecución sobre la planificación. A medida que se profundiza en el conocimiento de un problema se cambian los planes. Cuando el cliente vea nuestras propuestas, se le ocurriran nuevas ideas que cambiarán los planes. Cuando profundicemos en el conocimiento de nuevas teconologías haremos descubrimientos que cambiarán de nuevo los planes.

La planificacion excesiva es el problema habitual de los enfoques de desarrollo de software en cascada. Una planificacion es excesiva cuando desconocemos parte o gran parte del problema al que nos enfrentamos y aún así establecemos objetivos y plazos. En el mejor de los casos renunciaremos a las ventajas que se obtendrían de un mejor entendimiento del problema. En el peor, nuestro plan fracasará por encontrarnos riesgos con los que no contábamos.

Ágil quiere decir, adaptable. Desde esta premisa, los cambios son bienvenidos, derivan de un mejor entendimiento del problema y son una oportunidad para mejorar el software.

Este proyecto será ágil porque tendremos que tratar con cierta incertidumbre tecnológica al tener que evaluar la tecnología J2EE y el gestor de bases de datos PostgreSQL.

También será ágil porque hemos de estar dispuestos a adaptarnos a la mejor solución que encontremos al problema de conseguir un elevado rendimiento en las transacciones de reservas de viaje, algo que puede afectar al planteamiento general de la aplicación.

El desarrollo ágil no equivale al caos. Para poderse llevar a cabo necesita basarse en en dos pilares: **pruebas automáticas** y **retroalimentacion** (feedback) **temprana**, es decir objetivos palpables a corto plazo.

1.3.1. Manifiesto ágil

Dado que uno de los principales objetivos de este proyecto es la valoración del uso de metodologías de desarrollo ágil, presentaremos el manifiesto ágil, suscrito por personalidades tan reputadas como Alistair Cockburn y Martin Fowler entre otros. Se puede ver el manifiesto original siguiendo este enlace http://agilemanifesto.org/

Manifiesto ágil:

Valorar más a los individuos y su interacción que a los procesos y las herramientas

Este es posiblemente el principio más importante del manifiesto. Por supuesto que los procesos ayudan al trabajo. Son una guía de operación. Las herramientas mejoran la eficiencia, pero sin personas con conocimiento técnico y actitud adecuada, no producen resultados.
Las empresas suelen predicar muy alto que sus empleados son lo más importante, pero la realidad es que en los años 90 la teoría de producción basada en procesos, la re-ingeniería de procesos ha dado a éstos más relevancia de la que pueden tener en tareas que deben gran parte de su valor al conocimiento y al talento de las personas que las realizan.
Los procesos deben ser una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés. La defensa a ultranza de los procesos lleva a postular que con ellos se pueden conseguir resultados extraordinarios con personas mediocres, y lo cierto es que este principio es peligroso cuando los trabajos necesitan creatividad e innovación.

Valorar más el software que funciona que la documentación exhaustiva

Poder ver anticipadamente como se comportan las funcionalidades esperadas sobre prototipos o sobre las partes ya elaboradas del sistema final ofrece una retroalimentación (feedback) muy estimulante y enriquecedor que genera ideas imposibles de concebir en un primer momento; difícilmente se podrá conseguir un documento que contenga requisitos detallados antes de comenzar el proyecto.
El manifiesto no afirma que no hagan falta. Los documentos son soporte de la documentación, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan. Menos trascendentales para aportar valor al producto.
Los documentos no pueden sustituir, ni pueden ofrecer la riqueza y generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos. Por eso, siempre que sea posible debe preferirse, y reducir al mínimo indispensable el uso de documentación, que genera trabajo que no aporta un valor directo al producto.
Si la organización y los equipos se comunican a través de documentos, además de perder la riqueza que da la interacción con el producto, se acaba derivando a emplear a los documentos como barricadas entre departamentos o entre personas.

Valorar más la colaboración con el cliente que la negociación contractual

Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio.
Para el desarrollo ágil el valor del resultado no es consecuencia de haber controlado una ejecución conforme a procesos, sino de haber sido implementado directamente sobre el producto. Un contrato no aporta valor al producto. Es una formalidad que establece líneas divisorias entre responsabilidades, que fija los referentes para posibles disputas contractuales entre cliente y proveedor.
En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.

Valorar más la respuesta al cambio que el seguimiento de un plan

Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente al cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

1.3.2. Aplicación de la metodología Ágil en nuestro proyecto.

Hemos intentado seguir una metodología basada en SCRUM, pero el intento de asumir los roles de cliente, jefe de proyecto y programadores en una sola persona hacían muy artificial este enfoque. Por ello nos hemos centrado en dos aspectos de la metodología ágil que a nuestro juicio tienen la mayor importancia: las pruebas automatizadas, y otorgar mas importancia al software que a la documentación, es decir la documentación será una herramienta para el software y no otro producto a añadir.

1.3.2.1. Pruebas Automatizadas

Las pruebas automatizadas son la piedra angular de cualquier metodología de desarrollo ágil, ¿Cómo si no vamos a permitirnos cambiar las características de una aplicación en

mitad del desarrollo y tener garantías de no haber estropeado nada?

A pesar de los esfuerzos de la ingeniería de software por mantener una independencia de los diferentes componentes que conforman un sistema., inevitablemente los módulos de una aplicación tienen que interactuar entre si generando interdependencias que a veces no son evidentes.

Al cambiar la funcionalidad de alguno de estos módulos se pueden tener consecuencias imprevistas en otras partes del sistema. Estas consecuencias las podemos detectar ejecutando pruebas automatizadas que comprueben todas las partes del sistema.

Las pruebas, por tanto nos permiten afrontar con mayores garantías el cambio. Además el hecho de que sean **automatizadas** nos permite repetirlos infinidad de veces sin encarecer el coste de desarrollo.

Justificación económica de las pruebas automatizadas

Esta aplicación tiene unos unos 140 métodos. Cada vez que hemos incluido o modificado un método hemos ejecutado una suite de pruebas que ejecuta prácticamente todos los métodos.

Supongamos que hemos ejecutado la suite unas 500 veces (ejecuto la suite varias veces mientras desarrollo) Si un programador tuviese que depurar a mano esas pruebas tardaría 140 * 500 * 2 minutos por prueba = 140.000 minutos, o lo que es lo mismo 291 jornadas de trabajo.

El escribir las pruebas me ha costado entre 30 y 60 minutos por método, hay 56 métodos en el proyecto de test, por lo que he invertido 56 * 45minutos = unas 40 horas o 5 jornadas.

El escribir las pruebas automatizadas nos ha ahorrado 286 jornadas de trabajo.

Eso suponiendo que el programador que ejecutase las pruebas no se equivocase nunca y fuese tan perfecto como las pruebas automáticas.

Justificación de pruebas de unidad y de integración

En una aproximación ingenua podríamos pensar que una prueba se podría limitar a arrancar la aplicación, seguir el manual de usuario y ver que hace lo que tiene que hacer, es decir, lo que es una prueba de integración.

Sin embargo, por poner un ejemplo, si un programa tiene 6 controles con 4 valores de entrada posibles para dar un resultado determinado, probarlo solo con una prueba de integración nos obligaría a escribir 4 * 4 * 4 * 4 * 4 * 4 , pruebas que son las distintas posibilidades de entrada que tendría el sistema, o lo que es lo mismo 4 ^ 6 = 46.656

pruebas diferentes.

Con las pruebas de unidad podemos probar estos componentes individualmente, es decir probaríamos el primer control, para lo que escribiríamos cuatro pruebas para sus cuatro posibles valores de entrada. Haríamos lo mismo con los 5 controles restantes con lo que escribiríamos 4 pruebas para cada uno, en total

4 + 4 + 4 + 4 + 4 + 4 = 4 * 6 = 24 pruebas para probar todos los controles frente a las 46.656 pruebas que tendríamos que hacer si solo hiciésemos pruebas de integración.

1.3.2.1.1. Pruebas de unidad

El objetivo de las pruebas no es demostrar que el sistema funciona, sino encontrar errores. Empíricamente hemos visto que la mayor parte de los errores en las funciones se encuentran en la entrada de valores nulos, las cadenas vacías y los valores límite. Por valores límite entendemos que si una función admite un rango de valores, los limites son las fronteras de esos valores.

Normalmente en todas las pruebas el primer paso ha sido probar que hace lo que tiene que hacer con los valores esperados.

Luego normalmente se han probado con valores nulos, cadenas vacías y valores límite.

No hemos probado todas las clases, solo las que tenían cierta complejidad. Habitualmente se ha hecho una prueba por método aunque en algunos casos se han tenido que hacer más.

El framework elegido para las pruebas de unidad ha sido JUnit.

1.3.2.1.2. Pruebas de integración

Para las pruebas de unidad hemos usado HttpUnit y pruebas manuales. HttpUnit es un cliente http hecho en java que podemos combinar con JUnit para hacer pruebas de integración contra el servidor http.

1.3.2.1.3. Pruebas de rendimiento

Para ver la velocidad de las transacciones hemos usado clases sencillas con un cronómetro software, para medir los rendimientos, ya que la velocidad era una parte crítica del sistema.

1.3.2.2. Documentación.

En cuanto a la documentación, no hemos prescindido de ella, hemos constatado que son de gran utilidad los casos de uso bien documentados junto con diagramas de actividad UML.

Nos han sido especialmente útiles para describir el comportamiento de los controladores de cada uno de los casos de uso.

Los diagramas nos han ayudado a mejorar la productividad ya que permitían centrarse en la funcionalidad que se estaba desarrollando y dejar de lado las preocupaciones de las subsiguientes tareas.

No nos han parecido útiles sin embargo los diagramas de clases, de colaboración y de secuencia., y por tanto no los hemos usado. Esto es porque siempre hemos usado los mismos patrones: MVC , Façade, Decorators, Factorys y DAO..

1.4. Planificación del proyecto

A pesar de ser un proyecto ágil, no hemos renunciado a una mínima planificación, inicialmente identificamos las siguientes etapas:

1.4.1. Investigación tecnología j2ee y posgtreSql

Se dedicará un tiempo finito a la formación e investigación. Queremos evaluar la nueva tecnología JEE, Hibernate, Spring y JSF. Acabado ese tiempo elegiremos la tecnología que mejor se adapte a nuestro problema o la desarrollaremos nosotros mismos.

1.4.2. Instalacion de entorno

Antes de empezar tan siquiera a hacer los casos de uso Instalaremos las siguientes herramientas:

- Eclipse con soporte para jee (Ganymede)
- Java 1.6
- Herramienta de uml (Magic Draw)
- Tomcat
- PostgreSql, jdbc y PgAdmin.
- Junit (pruebas de unidad)
- HttpUnit (pruebas de integración)
- Metrics (para medir aspectos de calidad del software)

Establecemos también un sistema de backup automatizado a una unidad externa. Se evaluarán la instalación de herramientas automatizadas de construcción de proyectos como *Ant* y *Maven*.

1.4.3. Desarrollo de la aplicación por casos de uso (sprints)

En la terminología SCRUM se denominan sprints. SCRUM es la metodología de desarrollo a la que intentamos atenernos en un principio.

1.4.3.1. Sprint1. Transaccion de reserva de plazas.

Al ser la parte mas crítica, la de la gestión de las reservas con transacciones, se hará en primer lugar para despejar todas las incertidumbres al respecto. Se harán pruebas de estrés para asegurar el bue n rendimiento de la solución adoptada.

1.4.3.2. Sprint2. Búsqueda de ofertas (paquetes turísticos)

Se creará una pantalla de búsqueda para localizar las ofertas por diversos criterios y se enlazará con la reserva de plaza.

1.4.3.3. Sprint 3. Login, Alta de usuario e integracion con el sistema

Se crearán pantallas para el usuario se pueda dar de alta en el sistema en cualquier punto de la aplicación y que exija estar registrado para hacer la reserva

1.4.3.4. Sprint 4. Gestión completa de cuenta de usuario (CRUD) e integración con el sistema.

Se crearán las pantallas para el mantenimiento de la cuenta de usuario. Alta, lectura, modificación y borrado.

1.4.3.5. Sprint 5. Gestión de oferta de paquetes turísticos (CRUD)

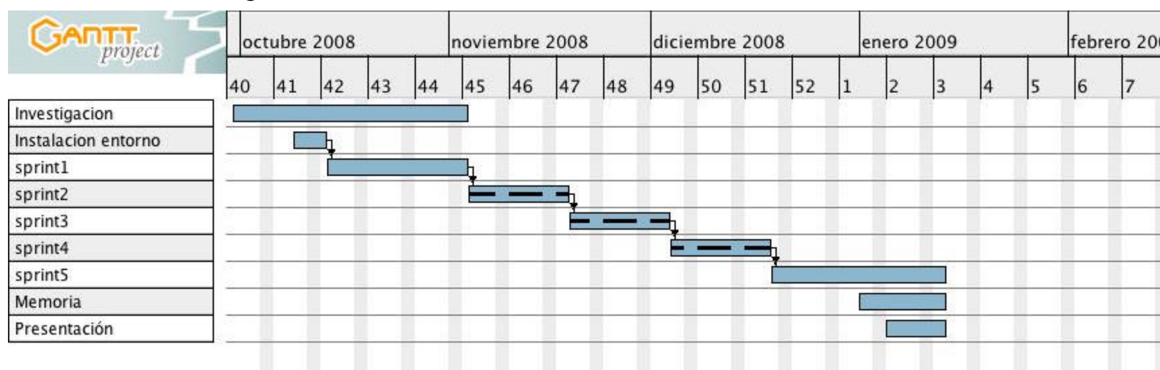
Se crearan las pantallas para la gestión de las ofertas. , lectura, modificación y borrado.

1.4.4. Ciclo de cada Sprint:

6H seguirá un modelo en espiral con las etapas de análisis, diseño, pruebas y codificación e integración.

Insertar gráfico mostrando la espriral

1.4.5. Diagrama de Gannt



1.5. Productos obtenidos

1.5.1. Metodología y herramientas de testing

A través de la experiencia obtenida en el desarrollo de este proyecto, hemos llegado a desarrollar una metodología de testing que resuelve algunos de los problemas habituales en la implantación de pruebas automatizadas.

Aislamiento de el código de pruebas del de producción.

Hemos creado un *workspace* específico para el proyecto de reservas, dentro de él se ha creado un *proyecto java* para las pruebas: *reservasTest*. Este proyecto, *reservasTest* accede a los otros proyectos del *workspace*, que contienen el código de producción, así puede acceder a sus clases, instanciarlas y probar sus métodos.

El proyecto *reservasTest* tiene en su *classpath* referencias a las librerías JUnit, HttpUnit, y las librerías JDBC de Postgresql.

Los otros proyectos no tienen ninguna referencia a estas bibliotecas. En la fase de despliegue no se exporta el proyecto *reservasTest* y los componentes del código de producción no contienen ningún test ni ninguna referencia a estos.

Pruebas de métodos privados.

Un problema frecuente con las pruebas es la prueba de los métodos privados de las clases.

Nosotros hemos resuelto este problema sustituyendo la visibilidad *private* de campos y métodos por la visibilidad de paquete, la visibilidad por defecto.

Las clases de prueba las ponemos en el mismo paquete que las clases a probar, pero en el proyecto de *reservasTest,* es decir replicamos el paquete (la estructura de directorios)

Para java, cuando usa el classpath es como si estuviesen en el mismo paquete, y por ello las clases de prueba, que están en otro proyecto pueden ver los miembros con visibilidad de paquete.

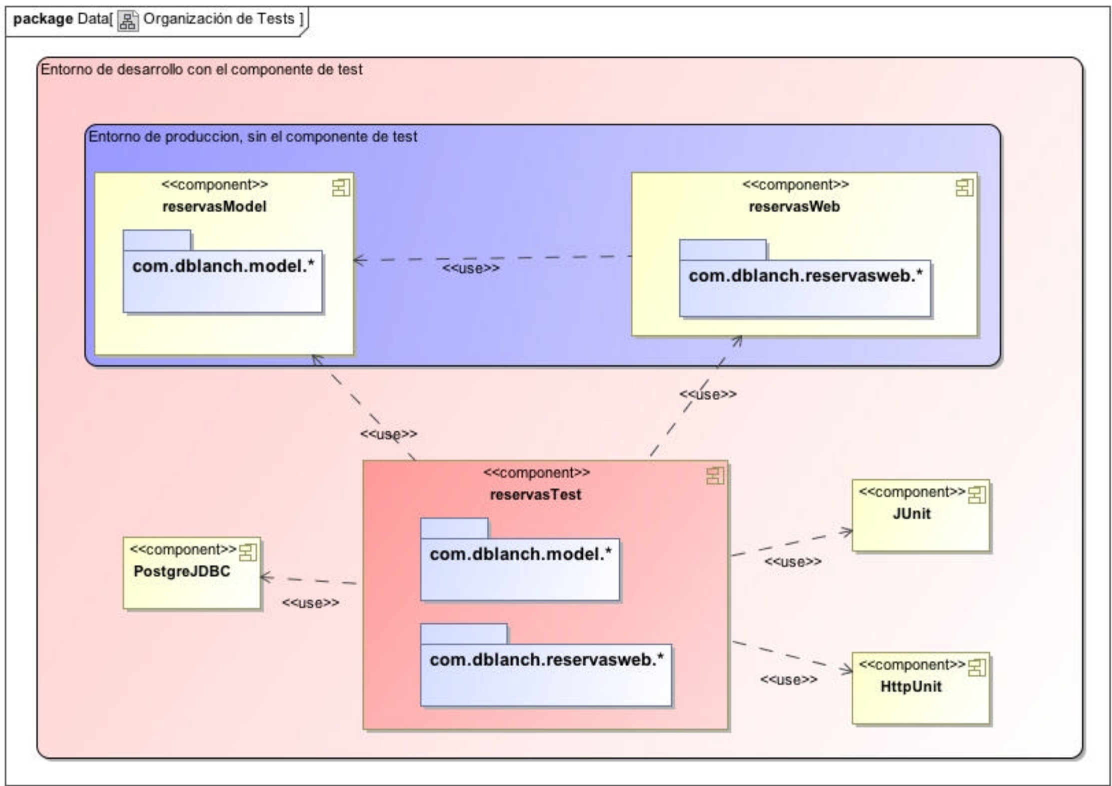


Ilustración 1 Aislamiento del código de pruebas del de producción

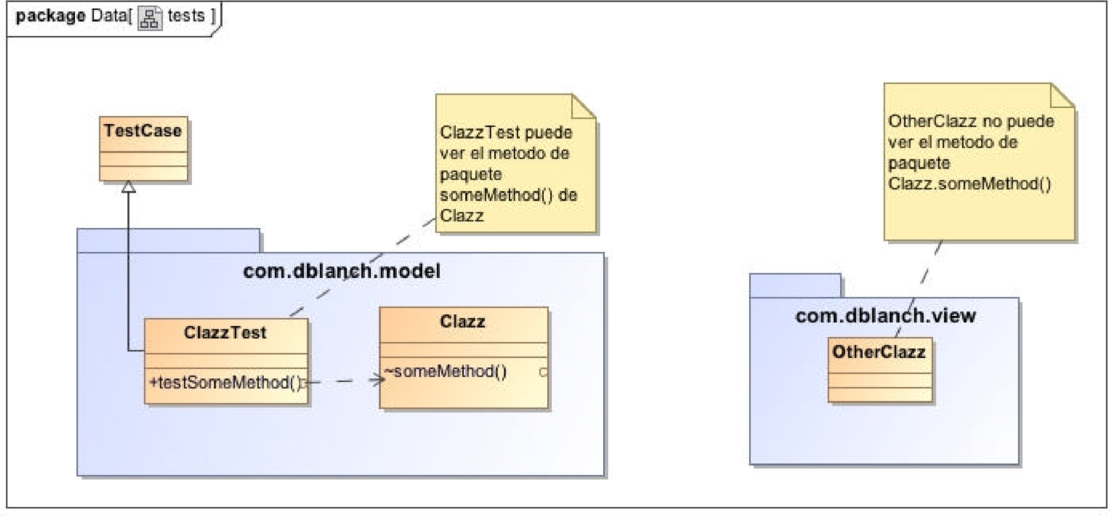


Ilustración 2 pruebas de métodos privados

Utilidades para pruebas

Se han creado unas clases de utilidad que pueden ser reutilizadas en cualquier otro proyecto estas clases están en el paquete *com.dblanch.testutils*

Tabla 1 Test utils

ConnectionPostgres	Proporciona una nueva conexión a PostgreSql, Servidor: localhost. User: postgres, Password: postgres	
SQLUtil	Utilidades para ser usadas con pruebas que incluyan una base de datos.	`Void` **executeBatch**`(file)` Ejecuta un fichero batch sql.
		Integer **queryInteger**(sql) Ejecuta una sentencia que ha de devolver un entero.
TestUtils	Utilidades para su uso en Tests. Parser de fechas...	
Timer	Cronómetro simple para medir rendimientos en Tests.	

Pruebas de clases con acceso a bases de datos

Creemos que hemos hallado una solución extremadamente efectiva a la vez que sencilla para probar clases que acceden a bases de datos.

Para probar estas clases tradicionalmente a menudo se acababa escribiendo complicado código jdbc o clases dao, para pruebas sencillas. El resultado es que acababa desistiendo de hacer todas las pruebas necesarias a estas clases.

La solución que hemos hallado está en la clase **SQLUtil** con sus dos métodos.

El método **SQLUtil.executeBatch(String file)** permite la ejecución de un fichero batch sql, que usamos usado para preparar el estado de la base de datos para las pruebas.

El método **SQLUtil.queryInteger(String sql)** ejecuta una sentencia sql que ha de devolver un entero. Como por ejemplo un SELECT COUNT(*), Luego ese valor se usará para verificar las pruebas.

A continuación mostramos un ejemplo del uso de esta clase para las pruebas:

```
	public void testReservaNormal() throws Exception {

		Connection connection = createConnection();

		SQLUtil sqlUtil = new SQLUtil(connection);
		sqlUtil.executeBatch("sql/PruebaUnitariaReserva_Postgresql.sql");

		ReservaCommandPostgresql commandPostgresql = new
ReservaCommandPostgresql(connection);

		ReservaVO reserva = new ReservaVO();
		reserva.setIdOferta(1);
		reserva.setIdCliente(1);
		reserva.setPlazasReservadas(1);

		commandPostgresql.doReserva(reserva);
```

```
                connection.commit();

                assertEquals(new Integer(1), sqlUtil.queryInteger("SELECT COUNT(*) FROM
RESERVA"));

                connection.close();
        }
```

Fichero batch "sql/PruebaUnitariaReserva_Postgresql.sql"

```
TRUNCATE OFERTA, CLIENTE, RESERVA CASCADE;

INSERT INTO OFERTA
    VALUES(1, 'PORTUGAL HERMOSO', 'VIAJE POR LAS ZONAS MAS BELLAS DE PORTUGAL', '2009-1-
1', '2009-1-9', 1, 1, '2008-12-31');


INSERT INTO CLIENTE
    VALUES(1, 'DANIEL', 'BLANCH BATALLER', 'DBLANCH', '1234', '2008-11-22', NULL);

INSERT INTO OFERTA
    VALUES(2, 'España', 'España', '2009-1-1', '2009-1-9', 10, 10, '2008-12-31');

INSERT INTO OFERTA
    VALUES(3, 'Francia', 'Francia', '2009-1-1', '2009-1-9', 20, 20, '2008-12-31');
```

El método **SQLUtil.queryInteger(String sql),** es todo lo que nos hace falta para los test, ya que se aprovecha de la flexibilidad y potencia de SQL.

Al menos hasta ahora hemos podido probar todo que nos ha hecho falta.

Por ejemplo, si en la prueba anterior hubiésemos querido comprobar que se habían insertado todos los atributos en sus correspondientes campos, podríamos haber escrito la sentencia sql del método de este otro modo :

SELECT COUNT(*) FROM RESERVA WHERE ID_OFERTA=1 AND ID_CLIENTE=1 AND PLAZAS_RESERVADAS=1

Y la función habría devuelto 1, validando el test...¡Así de fácil!

1.5.2. Entrega del producto

1.5.2.1. Alcance del producto final

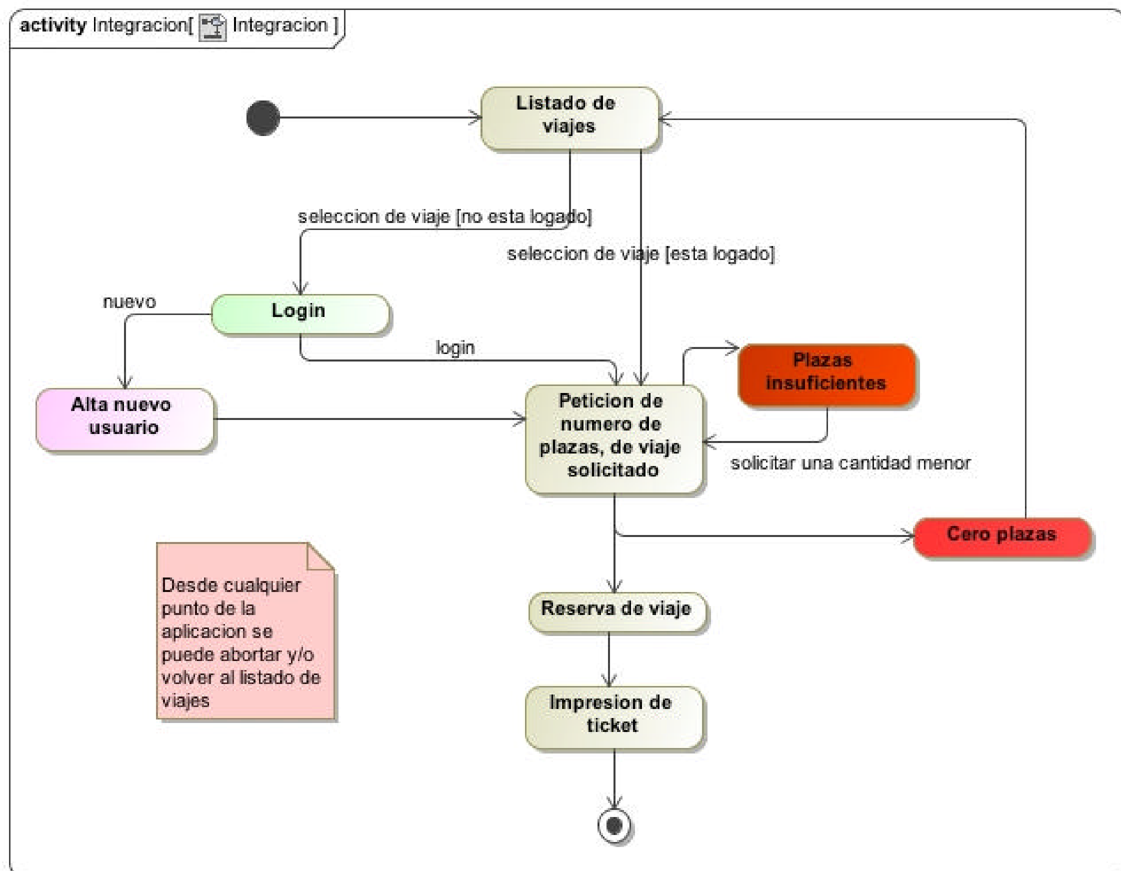
El rigor con el que hemos llevado los tests ha hecho que nos hayamos tenido que replantear el alcance del proyecto, algo que es perfectamente aceptable siguiendo la filosofía del Desarrollo Ágil.

Hemos dejado de lado los dos últimos sprints, que eran los menos importantes, los del mantenimiento de las cuentas del Cliente y el mantenimiento de las Ofertas.

Esta sería la etapa en la que presentaríamos el producto a nuestro cliente obtendríamos el feedback y reconduciríamos la aplicación para hacerla aún mejor.

En esta entrega, hemos implementado los 3 primeros sprints programados conformando un solo caso de uso completamente funcional, y con calidad contrastada.

Para explicar el alcance de la entrega lo ilustraremos con el siguiente diagrama de actividad :



1.1.1.1. Navegación por las Pantallas

La primera pantalla de la aplicación es la del listado de viajes. Nos permite filtrar por todos los campos de la forma que queramos.

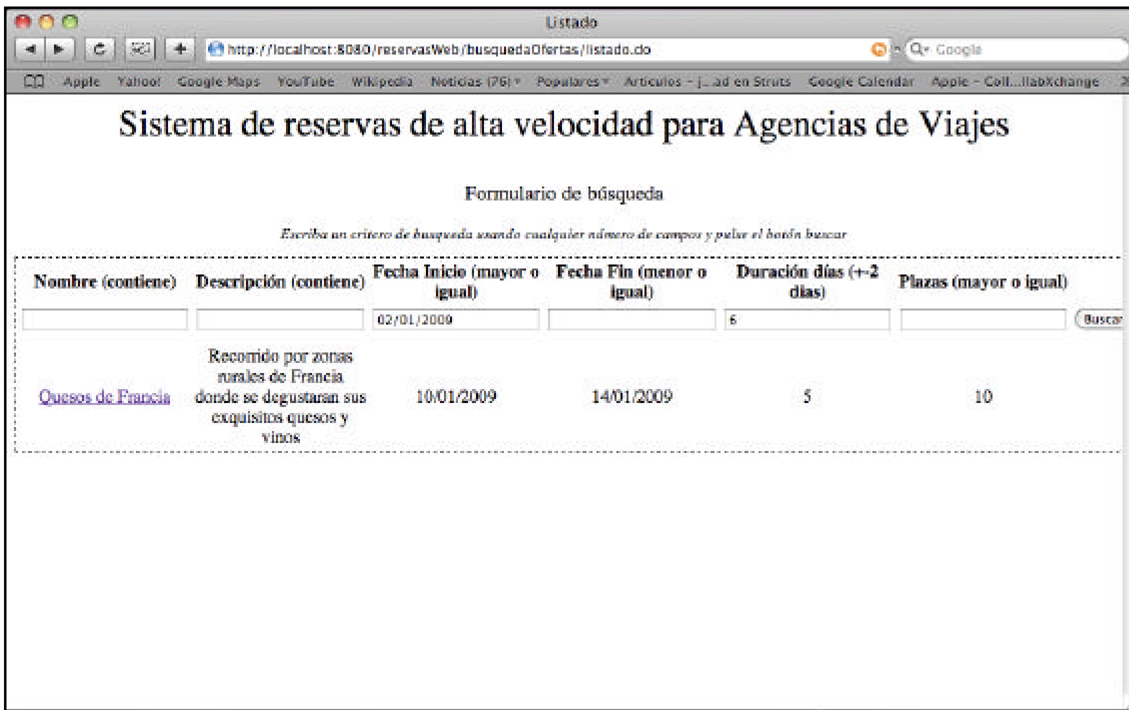


Ilustración 3 Listado de viajes

Si introducimos parámetros no válidos el sistema nos indicara el error por cada campo.

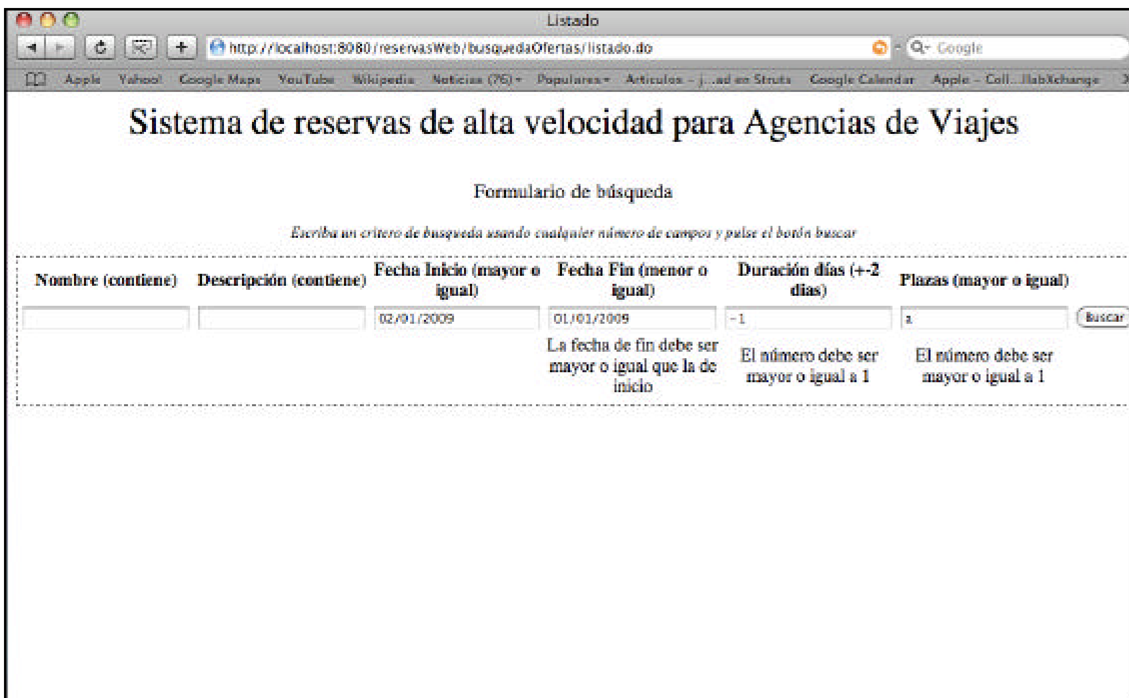


Ilustración 4 Listado de viajes. Error

Si no estamos logados, al hacer click sobre un elemento de la lista se nos mostrará la pantalla de login.

Ilustración 5 Login

El sistema valida los datos de entrada , y nos muestra los errores.

Ilustración 6 Login. Error

Comprueba si existe el usuario y si la contraseña es correcta.

Ilustración 7 Login. no usuario

Una vez logado, el sistema nos lleva al elemento de la lista que habíamos seleccionado y que disparó este evento.

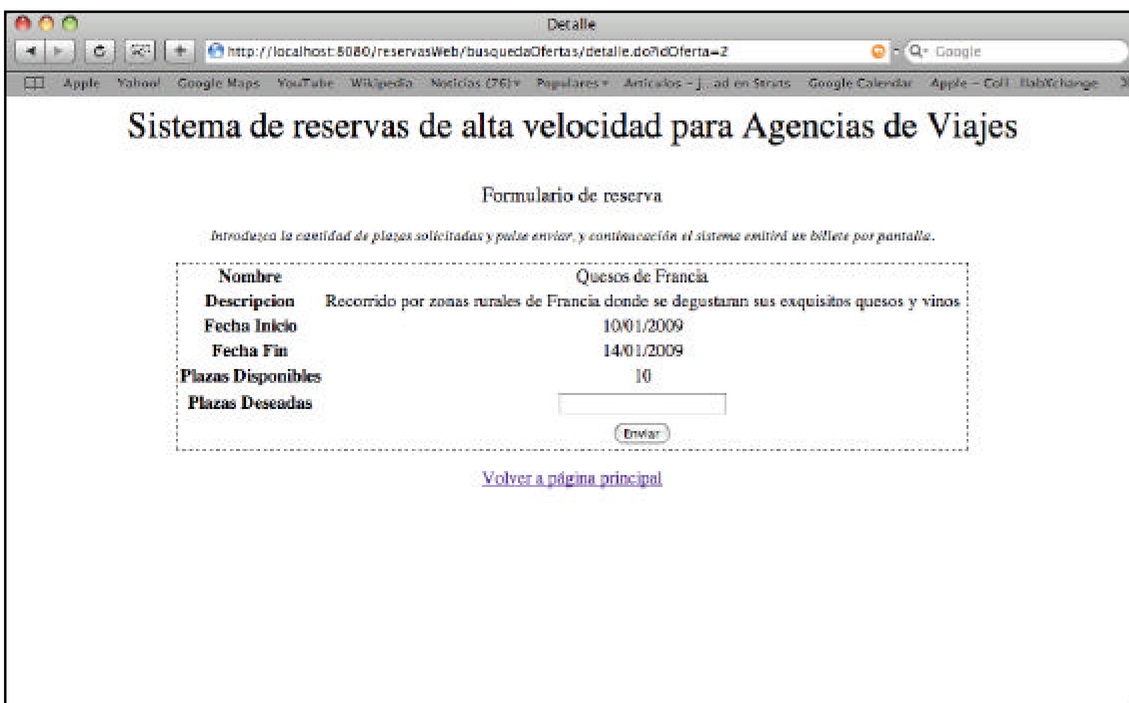


Ilustración 8 Petición de número de plazas

El sistema comprueba que el valor introducido es correcto, rechazándolo si no es así.

Ilustración 9 Petición de número de plazas. Error

Si se pide un número de plazas inferior o igual al disponible, el sistema hace la reserva e imprime un ticket

Ilustración 10 Impresión de ticket

Si se piden mas plazas de las que se ofrecen pero queda todavía alguna plaza, el sistema nos deja pedir una cantidad menor.

Formulario de reserva

Introduzca la cantidad de plazas solicitadas y pulse enviar, y continucación el sistema emitirá un billete por pantalla.

Plazas insuficientes, pruebe con una cantidad menor

Nombre	Quesos de Francia
Descripcion	Recorrido por zonas rurales de Francia donde se degustaran sus exquisitos quesos y vinos
Fecha Inicio	10/01/2009
Fecha Fin	14/01/2009
Plazas Disponibles	8
Plazas Deseadas	9

Enviar

Volver a página principal

Ilustración 11 Plazas insuficientes

Si no queda ninguna plaza el sistema nos informa del hecho y no nos deja mas opción que volver a la pantalla inicial.

Informe de plazas agotadas

Pulse salir para volver a la pantalla de búsqueda de viajes

Plazas agotadas

No existe ninguna plaza para la oferta solicitada. Otra transaccion adquirió las plazas antes que usted.

Volver a página principal

Ilustración 12 Ninguna plaza

En el proceso de Login, también podíamos haber elegido darnos de alta como usuario nuevo. El sistema hace las habituales comprobaciones de los datos de entrada.

Formulario de alta de nuevo usuario

Rellene todos los campos y a continuación pulse 'Enviar'.

El sistema registrará sus datos y le enviará a la página que hubiera solicitado antes de iniciar el proceso de registro.

Nombre	dddddddddddddddddddd	El campo no puede contener mas de 40 caracteres
Apellidos		El campo no puede estar vacío
Login (por ejemplo su e-mail)		El campo no puede estar vacío
Password		El campo no puede estar vacío
	Enviar	

Volver a página principal

Ilustración 13 Alta nuevo usuario. Error

El sistema no permite la entrada de dos Clientes con el mismo Login.

Rellene todos los campos y a continuación pulse 'Enviar'.

El sistema registrará sus datos y le enviará a la página que hubiera solicitado antes de iniciar el proceso de registro.

Ya existe un login con ese nombre, elija otro.

Nombre	Ana
Apellidos	Blanch Bataller
Login (por ejemplo su e-mail)	ana.blanch@fastweb.it
Password	••••
	Enviar

Volver a página principal

Ilustración 14 Alta nuevo usuario. Duplicado

2. Especificación y requerimientos

2.1. Información inicial

Queremos crear un sistema reservas on-line para una agencia de viajes. El negocio de una agencia de viajes es la venta de paquetes turísticos que consisten en una combinación de diferentes servicios (transporte, alojamiento, excursiones, guías, etc.…) en un solo producto.

El paquete turístico consta de un código interno, un nombre, un precio por persona, una descripción, una presentación más o menos elaborada y un número de plazas ofertadas.

El sistema de reservas debe ofrecer un catálogo de paquetes turísticos por el que se pueda buscar por diferentes criterios, como fechas de salida, duración del viaje, precio, etc.

Cuando un cliente hace una reserva, el sistema debe actualizar el número de plazas disponibles, restando las que el cliente ha reservado, evitando que se puedan solicitar más plazas de las existentes.

El sistema debe guardar con la reserva, el nombre del cliente, el paquete solicitado y el número de plazas que quiere.

El sistema debe permitir la inclusión futura con de una interfaz con medios de pago electrónico.

El sistema tiene que permitir la introducción de paquetes turísticos en el catálogo, su modificación, consulta y borrado. La modificación y el borrado solo se podrán hacer si no se ha vendido ninguna reserva.

El sistema tiene que permitir el registro de los clientes donde introducirán información sobre sus datos personales, tarjetas de crédito, etc. También la modificación, consulta y borrado.

El sistema tiene que manejar de manera óptima las transacciones, minimizando bloqueos y asegurando la consistencia.

2.2. Glosario

Oferta: Producto en venta por la agencia de viajes

Cliente: Usuario de la aplicación registrado

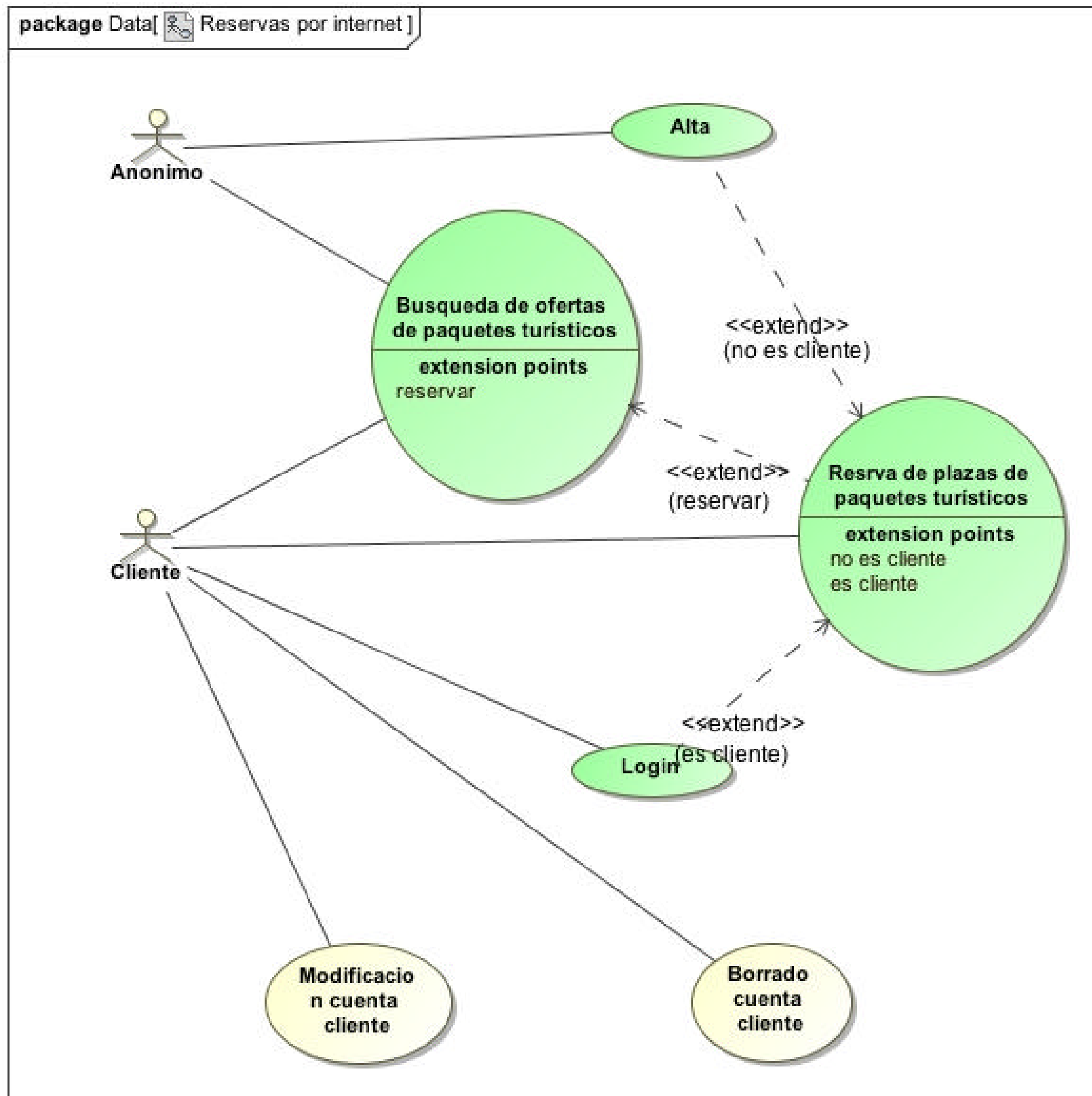
Reserva: Solicitud de venta de una oferta

Paquete Turístico: Agregado de servicios turísticos que vende una agencia de viajes

2.3. *Modelo del dominio*

- Oferta:
  - Nombre
  - Descripción
  - Fecha de Inicio
  - Fecha de Fin
  - Duración
  - Plazas Disponibles
  - Plazas Totales
  - Fecha límite de compra
- Reserva
  - Numero de plazas reservadas
  - Oferta
  - Cliente
- Cliente
  - Nombre
  - Apellidos
  - Login
  - Password
  - Fecha de Alta

2.4. *Diagrama de casos de uso*



2.5. *Documentación textual de los casos de uso*

Explicación del modelo seguido para documentar los casos de uso

Se ha seguido un modelo de Alistair Cockburn, gran especialista en metodologías de desarrollo.

La única pequeña dificultad de la estructura de estos casos de uso es el seguimiento del flujo. Como sabemos, una interacción con un sistema puede tener ramificaciones, y estas suelen ser difíciles de explicar. Trataremos de mostrar como con un ejemplo como las expresa este modelo

Caso de uso:
Sistema de alta seguridad de acceso a control de lanzamiento de misiles
Flujo principal:
1 El usuario se sienta a los mandos y pulsa el botón de acceso

2 El sistema pregunta su nombre de usuario y contraseña

3 El usuario introduce su nombre y contraseña

4 El sistema comprueba sus credenciales y le permite el acceso al control de lanzamiento de misiles

Flujo alternativo:

*a El usuario pulsa el botón 'ABORT'

    1 El sistema vuelve a la posición standby

4a El sistema comprueba que sus credenciales son erróneas

    1 El sistema devuelve al usuario al punto 2 informándole del error

4b El sistema detecta que el usuario ha introducido sus credenciales erróneamente tres veces

    1 El sistema bloquea la cuenta de usuario

    2 El sistema cierra las compuertas blindadas del edificio

    3 El sistema abre las válvulas de gas paralizante

    4 El sistema dispara la alarma de INTRUSO EN INSTALACIONES

Explicación:

El flujo principal se numeran los pasos

En el flujo alternativo se explican las ramificaciones. El primer número es el paso del flujo principal, la letra es la referencia de la ramificación de ese paso. Si un paso 1 se puede hacer de 3 formas diferentes, en el flujo alternativo tendrá tres ramas 1a , 1b, 1 c.

Dentro de la rama del flujo alternativo los pasos se numeran empezando por el uno.

2.5.1. Búsqueda de ofertas de paquetes turísticos

Actores:

Usuario

Flujo principal:

1) El usuario elije la opción de búsqueda de oferta.

2) El sistema muestra una pantalla con los criterios de selección de ofertas:

- nombre (criterio: contiene palabra)

- descripción (criterio: contiene palabra)
- fechaInicio (criterio: mayor o igual que)
- fechaFin (criterio: menor o igual que)
- duración (criterio: igual a con dos días por arriba y dos días por abajo)
- plazasDisponibles (criterio: mayor que)

3) El usuario elije uno o varios de los criterios de búsqueda y selecciona el filtro.

4) El sistema muestra una pantalla con los resultados de la búsqueda. En ningún caso mostrará aquellas ofertas para las que no existan plazas disponibles, ni cuya fecha límite de compra sea mayor o igual a la actual.

5) El usuario elige una de las ofertas.

6) El sistema muestra el detalle de la oferta.

Flujo alternativo:

*a) El usuario cancela en cualquier momento.

1) el sistema devuelve a la pantalla principal

4a) El sistema detecta que no se ha seleccionado ningún criterio de búsqueda

1) El sistema informa del error y devuelve al punto 2

4b) El sistema detecta que alguno de los campos no contienen datos válidos para la búsqueda. Ej. las fechas no son fechas válidas, el campo duración o el campo plazasDisponibles no son números

1) El sistema informa del error y devuelve al punto 2

Puntos de extensión:

7) El usuario elije reservar la oferta mostrada en detalle.

1) El sistema enlaza con el caso de uso de reserva de plaza.

Requisitos no funcionales:

El sistema debe ser rápido en devolver los resultados. Esto implica que la transacción debe permitir hacer lecturas sucias, ya que es mas importante la velocidad de respuesta que la exactitud del contenido.

Temas pendientes:

Comprobar riesgo de *sql injection* al construir una cláusula *where* dinámica.

2.5.2. Reserva de plazas de paquetes turísticos

Actores:

Usuario

Disparador:

El usuario ha seleccionado una de las ofertas existentes que tienen disponibilidad de plaza en ese momento.

Precondición:

El usuario está registrado.

Flujo principal:

1) El sistema presenta una pantalla con los datos de la oferta:

- código interno (no se mostrará) (no editable)
- nombre (no editable)
- descripción (no editable)
- fechaInicio (no editable)
- fechaFin (no editable)
- duración (no editable)
- número de plazas disponibles (no editable)
- número de plazas deseadas (editable)

2) El usuario introduce el número de plazas deseado

3) El sistema registra la reserva y notifica al usuario el éxito de la operación.

Flujo alternativo:

1-2a) El usuario cancela la operación

    1) El sistema devuelve a la pantalla principal

2a) El número de plazas que solicita el usuario es 0 o nulo

    1) El sistema informa del error y vuelve al punto 1 (con datos actualizados).

2b)El número de plazas que solicita el usuario es superior al número de plazas disponibles

    1) El sistema informa del error y vuelve al punto 1 (con datos actualizados).

Poscondición:

El usuario tiene N reservas de la oferta O

Requisitos no funcionales:

Las transacciones deben ser rápidas y correctas., por lo que deben bloquear la lectura de plazas para luego poder actualizarla, restando las plazas, pero el mínimo tiempo posible para permitir que otras transacciones operen sobre el mismo registro.

La transacción tiene que ser segura, no puede ser falsificable alterando parámetros de la petición.

2.5.3. Login

Disparadores:

El usuario elige hacer una reserva y no esta registrado en el sistema.

El usuario elige la opción de registrarse en el sistema.

Flujo principal:

1) El sistema muestra una pantalla con los campos de:

- nombre de usuario
- password

2) El usuario escribe su nombre y password.

3) El sistema comprueba que el usuario existe y su contraseña es correcta, por lo que devuelve al usuario al punto donde se originó el evento, es decir pantalla de inicio o a la reserva que estaba apunto de reservar.

Flujo alternativo:

1-2a) El usuario cancela la operación.

1) El sistema devuelve al usuario al menú principal

3a) El sistema detecta que el usuario no existe o que la contraseña es incorrecta.

1) El sistema notifica el error y devuelve al usuario al punto 1 con los datos introducidos por el usuario.

Requisitos no funcionales:

En ningún caso debe intercambiarse entre el navegador y el servidor el nombre de usuario y/o la contraseña una vez pasada la validación, para evitar suplantación de identidad o robo de contraseña.

Temas pendientes:

Hacer que el proceso de autenticación vaya encriptado con SSL (estimar el coste)

Hacer que el password de usuario vaya encriptado dentro de la base de datos (estimar coste)

2.5.4. Mantenimiento cuenta de usuario

2.5.4.1. Alta de usuario

Disparadores:

Desde pantalla de login el usuario elige la opción de darse de alta en el sistema.

Desde la pantalla de mantenimiento de cuenta de usuario, el usuario elige la opción de darse alta.

Flujo principal:

1) El sistema muestra una pantalla con los campos de (todos obligatorios):

- nombre (campo obligatorio)
- apellidos (campo obligatorio)
- login (campo obligatorio)
- password (campo obligatorio)

2) El usuario rellena los datos.

3) El sistema registra los datos en la base de datos, rellenando una fecha de alta y devuelve al usuario a la pantalla de login comunicando el éxito de la operación.

Flujo alternativo:

1-2a) El usuario cancela la operación.

1) El sistema devuelve al usuario a la pantalla de inicio.

3a) El sistema detecta que no se han rellenado todos los campos

1) El sistema indica que campos son los que no se han rellenado y devuelve al punto 1 con los datos que ha llenado el usuario.

3c) El sistema detecta que ya existe un usuario con el mismo login

1) El sistema comunica el error e insta al usuario a cambiar el login

2) El sistema devuelve al usuario al punto 1

Requisitos no funcionales:

En ningún caso debe ser visible el nombre de usuario y/o la contraseña una vez pasada la validación.

Temas pendientes:

Hacer que el proceso de alta vaya encriptado con SSL (estimar el coste)

Hacer que la contraseña de usuario vaya encriptada dentro de la base de datos (estimar coste)

Requerir una contraseña con un nivel de seguridad aceptable es decir con un número de caracteres suficientes, que contenga números, etc. (estimar coste)

2.5.4.2. Consulta de usuario

Precondición:

El usuario se ha logado en el sistema previamente con lo que tiene visible la opción de mantenimiento de cuenta de usuario.

Flujo principal:

1) El usuario elige la opción de mantenimiento de usuario

2) El sistema recupera los datos del usuario y muestra una pantalla con los campos de:

- nombre
- apellidos
- login
- password

Flujo alternativo:

*a) El usuario cancela la operación

1) El sistema devuelve a la pantalla de inicio

Puntos de extensión:

Borrar (desactivar) usuario.

Modificar usuario.

2.5.4.3. Modificación de usuario

Disparadores:

Desde la pantalla de mantenimiento de cuenta de usuario, el usuario elige la opción de modificar su cuenta.

Precondición:

El usuario se ha logado en el sistema previamente con lo que tiene accesible la opción de mantenimiento de cuenta de usuario.

Flujo principal:

1) El sistema recupera los datos del usuario y muestra una pantalla con los campos de:

- nombre
- apellidos
- login
- password

2) El usuario rellena los datos.

3) El sistema registra los datos en la base de datos y devuelve al usuario a la pantalla inicial.

Flujo alternativo:

1-2a) El usuario cancela la operación

1 El sistema devuelve a la pantalla anterior

3a) El sistema detecta que no se han rellenado todos los campos

1) El sistema indica que campos son los que no se han rellenado y devuelve al punto 1

3c) El sistema detecta que ya existe un usuario con el mismo login

1) El sistema comunica el error e insta al usuario a cambiar el login

2) El sistema devuelve al usuario al punto 1

Temas pendientes:

Hacer que el proceso de modificación vaya encriptado con SSL (estimar el coste).

Hacer que la contraseña de usuario vaya encriptada dentro de la base de datos (estimar coste).

Requerir una contraseña con un nivel de seguridad aceptable es decir con un número de caracteres suficientes, que contenga números, etc. (estimar coste).

2.5.4.4. Borrado de usuario

Precondición:

El usuario se ha logado en el sistema previamente con lo que tiene accesible la opción de mantenimiento de cuenta de usuario.

Flujo principal:

1) El usuario elige borrar o darse de baja

2)El sistema pide confirmación

3) El usuario acepta

4) El sistema escribe en *fecha de baja* el día de hoy, con lo que queda registrada la baja. (baja lógica) y devuelve al usuario a la pantalla de inicial de la aplicación.

Flujo alternativo:

1-3) El usuario cancela la operación

1 El sistema vuelve a la pantalla de detalle de usuario.

2.5.5. Mantenimiento de ofertas

Sistema opcional que estará situado dentro de la intranet del cliente, por lo que no será en principio necesario ninguna consideración de seguridad especial.

2.5.5.1. Alta de oferta

Flujo principal:

1) El usuario elige la opción de crear una nueva oferta.

2) El sistema muestra una pantalla con los campos:

- Nombre (campo obligatorio)
- Descripción (campo obligatorio)
- fechaInicio (campo obligatorio)
- fechaFin (campo obligatorio)
- plazasTotales (campo obligatorio)
- fechaLimiteDeCompra (campo obligatorio)

3) El usuario rellena los datos.

4) El sistema registra los datos en la base de datos, inicializa plazasDisponibles = plazasTotales, y devuelve al usuario a la pantalla de login comunicando el éxito de la operación.

Flujo alternativo:

1-3a) El usuario cancela la operación

1) El sistema devuelve al usuario a la pantalla de inicio.

4a) El sistema detecta que no se han rellenado todos los campos

1) El sistema indica que campos son los que no se han rellenado y devuelve al punto 2 con los datos que rellenó el usuario.

4b) El sistema detecta inconsistencias en los datos. Ej. fechaInicio mayor o igual a fechaFin. fechaLimite de compra menor que hoy().

1) El sistema informa de los errores y devuelve al punto 2 con los datos que rellenó el usuario.

2.5.5.2. Consulta de oferta

Flujo principal:

1) El usuario elije la opción de búsqueda de ofertas

2) El sistema muestra una pantalla con los criterios de selección de ofertas:

- nombre (criterio: contiene palabra)
- descripción (criterio: contiene palabra)
- fechaInicio (criterio: mayor o igual que)
- fechaFin (criterio: menor o igual que)

3) El usuario elije uno o varios de los criterios de búsqueda y selecciona el filtro.

4) El sistema muestra una pantalla con los resultados de la búsqueda.

5) El usuario elige una de las ofertas.

6) El sistema muestra el detalle de la oferta.

Flujo alternativo:

*a) El usuario cancela en cualquier momento.

1) El sistema devuelve a la pantalla principal.

4a) El sistema detecta que no se ha seleccionado ningún criterio de búsqueda

1) El sistema informa del error y devuelve al punto 2

4b) El sistema detecta que alguno de los campos no contienen datos validos para la búsqueda. Es decir las fechas no son fechas válidas o el rango no es correcto.

1) El sistema informa del error y devuelve al punto 2

Requisitos no funcionales:

En el listado deben salir todas las ofertas pero deberíamos evitar las lecturas sucias, ver el nivel de aislamiento optimo de transacciones.

2.5.5.3. Modificación y baja de oferta

Queda fuera del ámbito del proyecto por ser demasiado complejo. El sistema necesitaría detectar si se ha hecho alguna reserva, si es así dependiendo del dato que se quiera modificar, deberá permitir o no la modificación. También debería avisar a los usuarios y permitir que cancelasen la reserva, etc.

En cuanto a las transacciones, cuando menos, tendría que bloquear durante la modificación, la inserción de nuevos registros.

2.6. Requisitos de la interfaz de usuario

La interfaz de usuario será HTML, ya que es una aplicación Web. Prescindiremos de javascript, porque el usuario lo puede desactivar. Evitaremos los diseños complicados para que el tratamiento de pruebas con httpUnit sea posible.

Haremos hincapié en que la navegación esté muy controlada. La navegación ha de permitir buscar los viajes y solo cuando se seleccione uno, nos pedirá logarnos, o darnos de alta en el sistema.

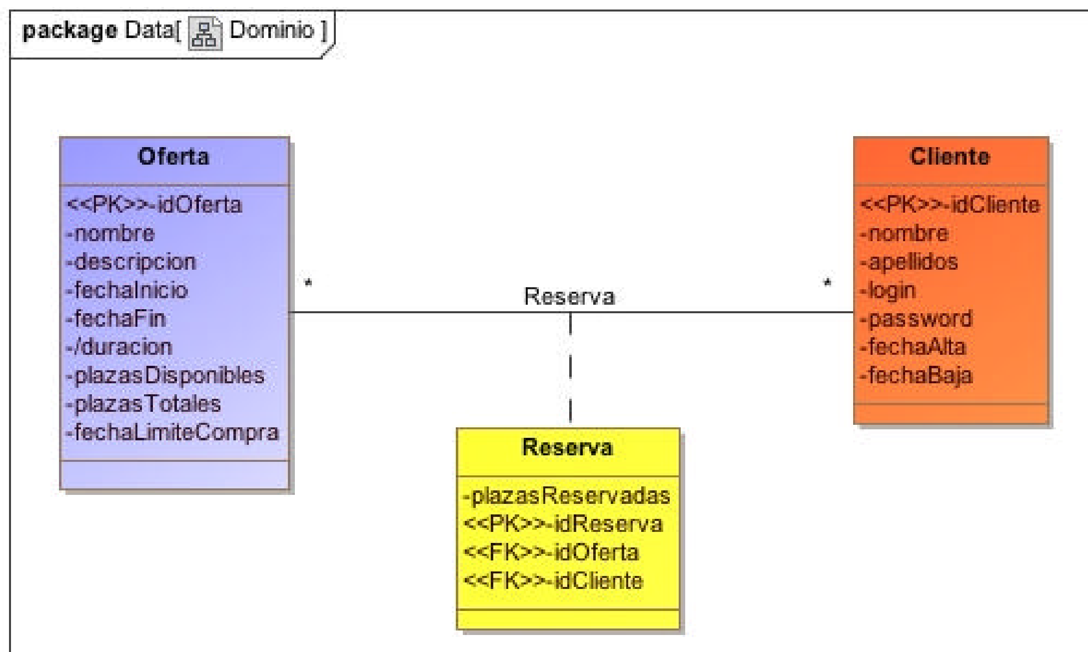
El sistema ha de ser amigable, en particular a la hora de mostrar los errores de validación de los formularios. Estos tienen que presentarse al lado del mismo campo que originó el error.

Usaremos css para separar el diseño de los datos.

Cuidaremos al máximo la seguridad evitando el paso de parámetros sensibles al navegador, como por ejemplo datos de la cuenta de usuario.

3. Análisis

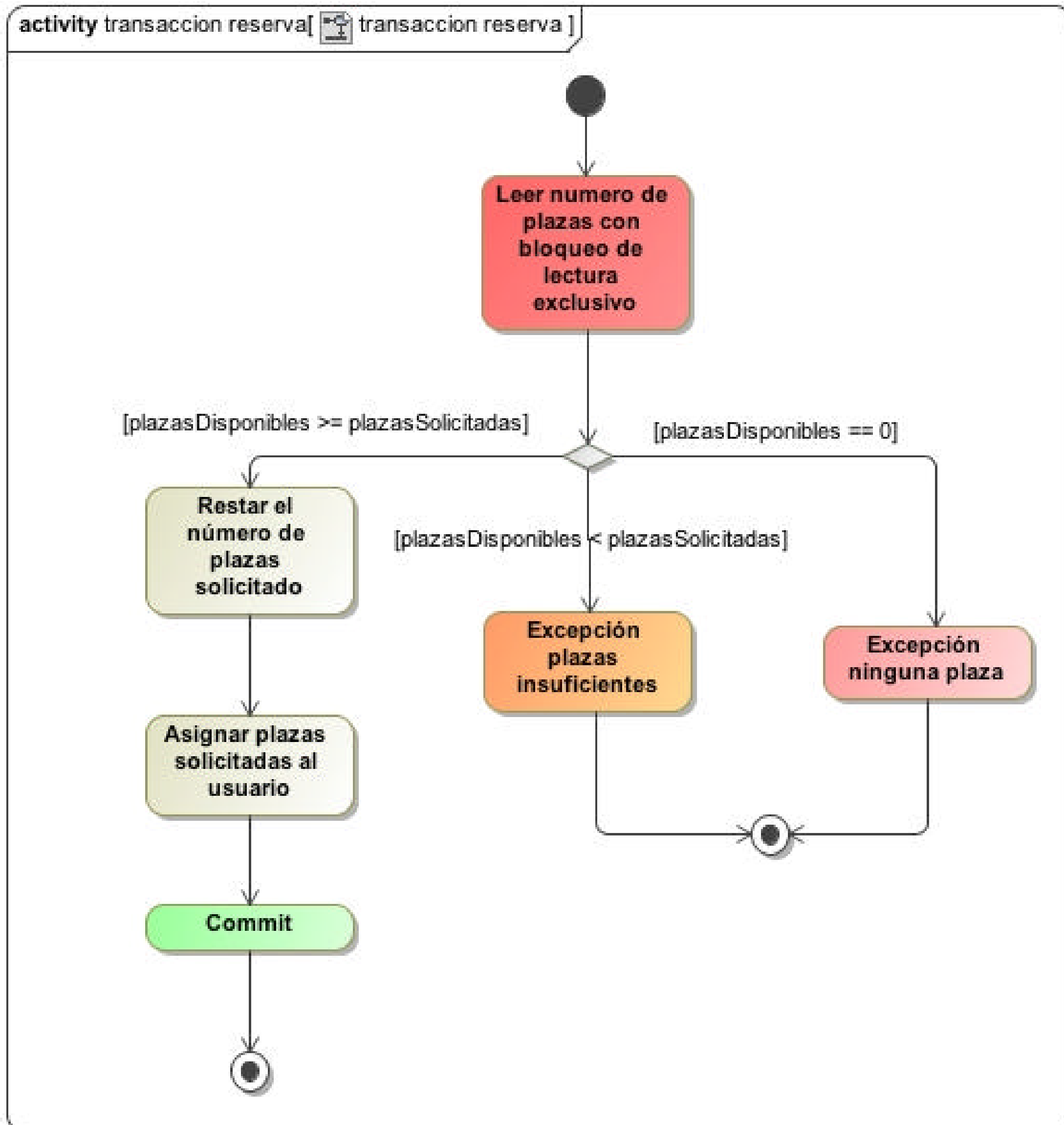
3.1. Identificación de las clases de entidades y sus atributos



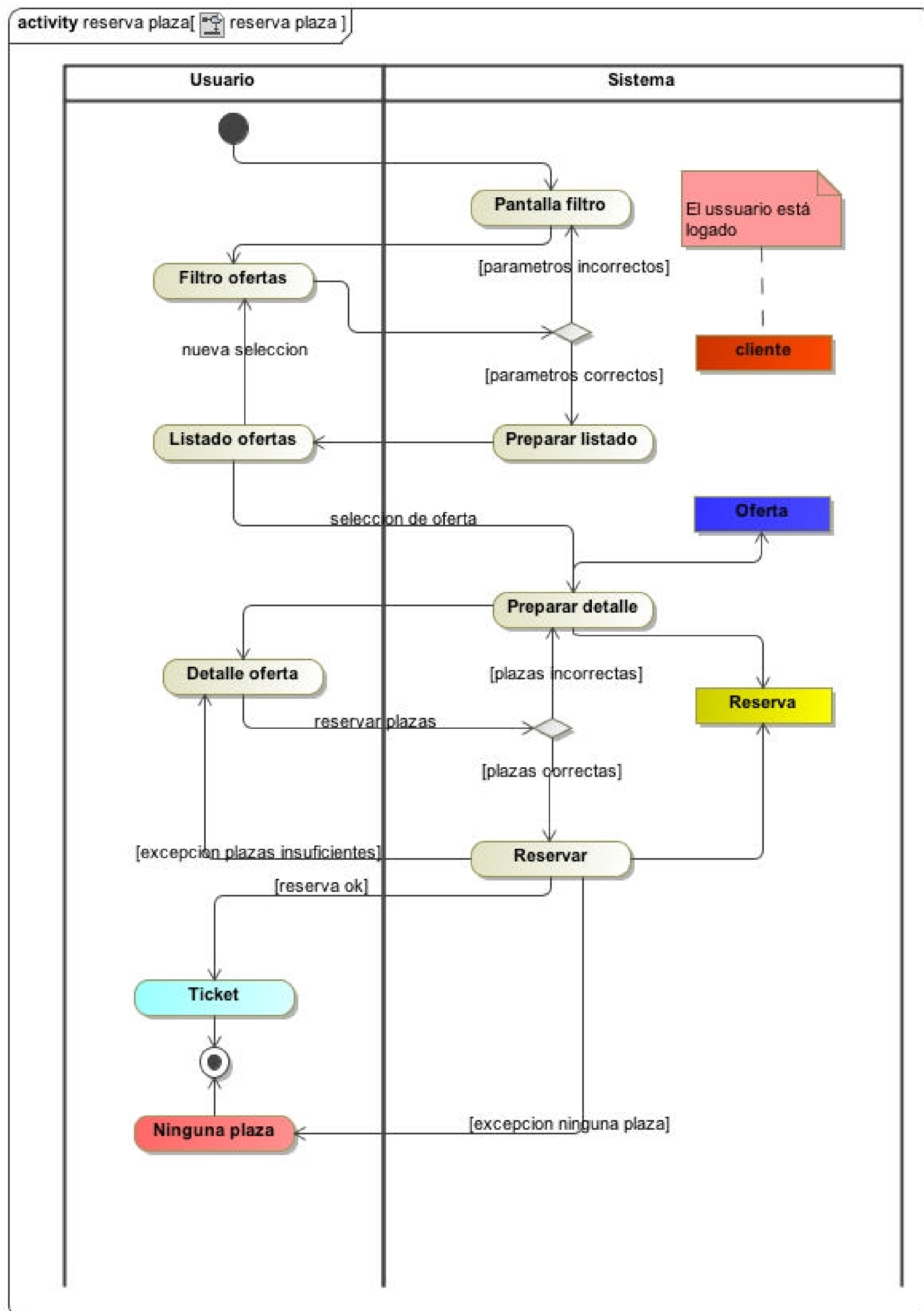
3.2. *Diagramas de estados*

3.3. *Diagramas de actividades*

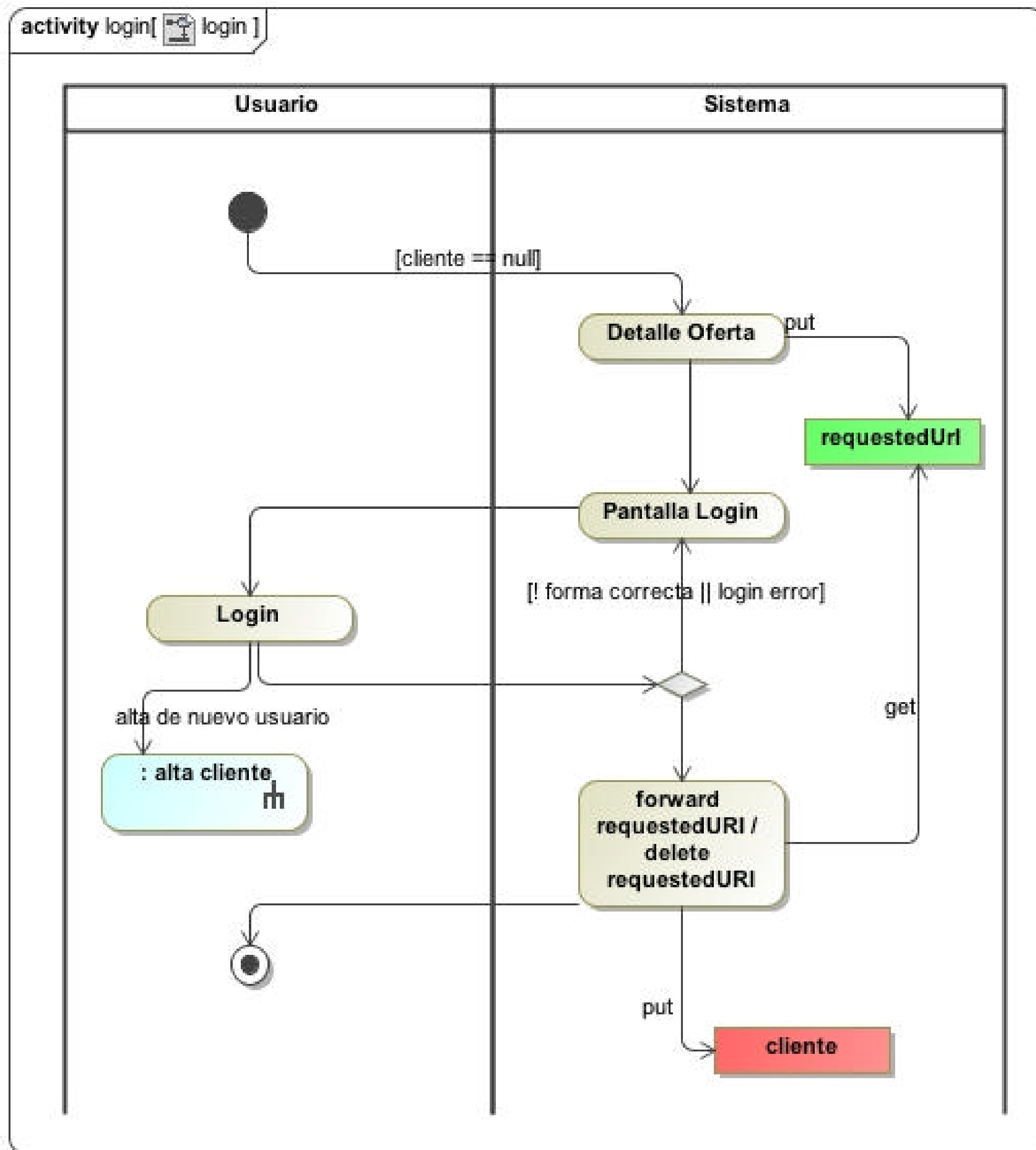
3.3.1. Transacción de reserva de plaza



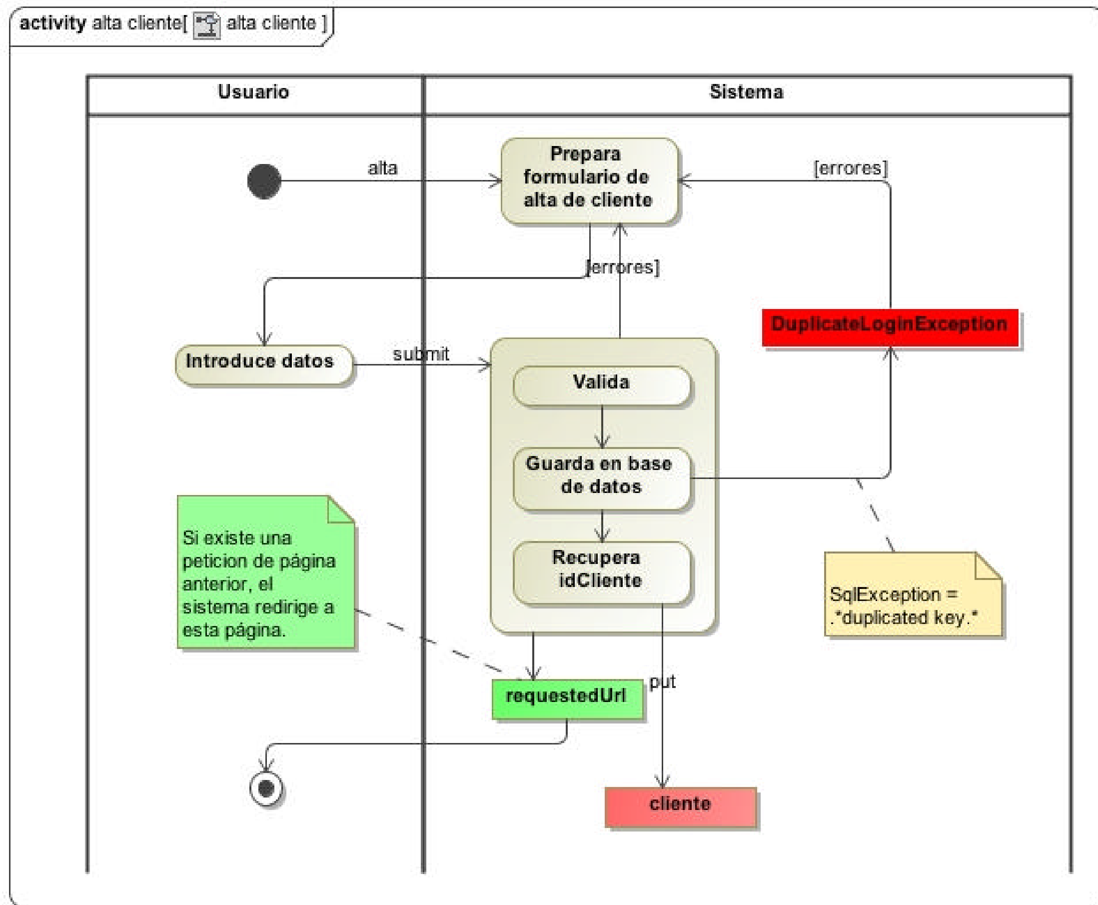
3.3.2. Reserva de plaza (búsqueda + reserva)



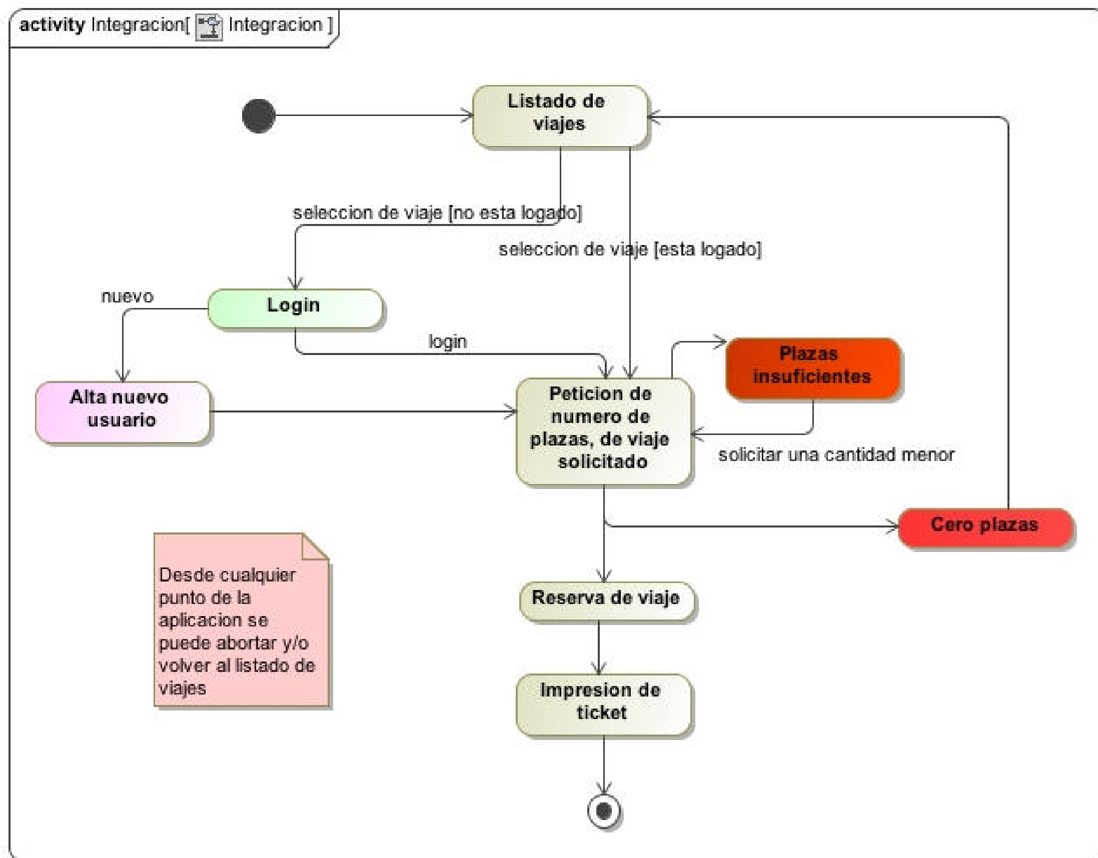
3.3.3. Login



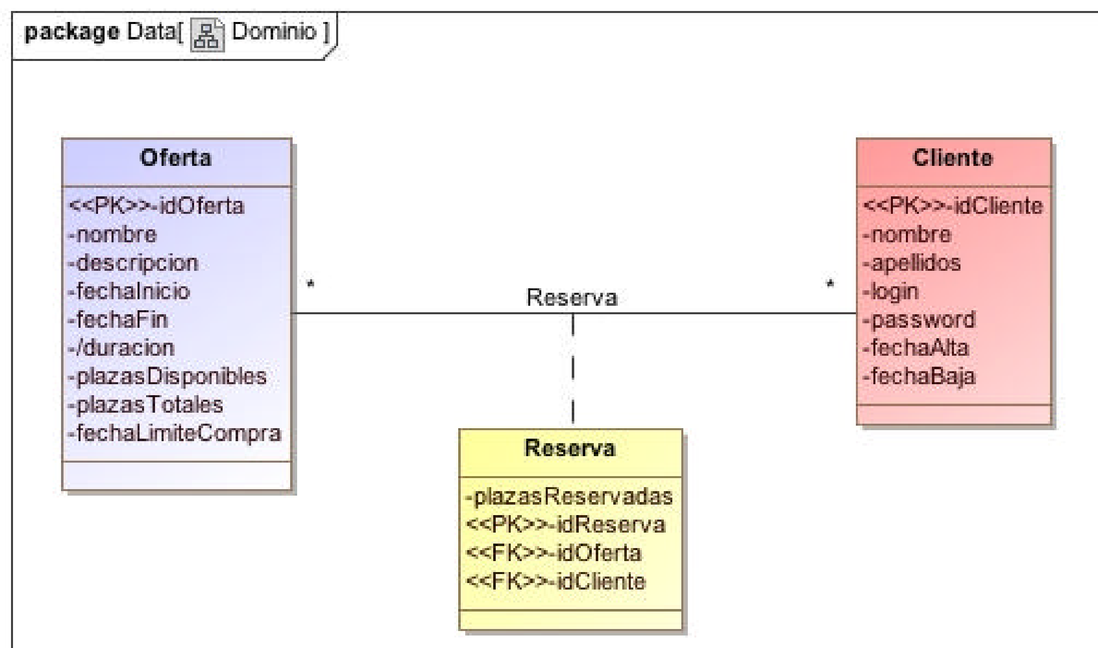
3.3.4. Alta de usuario



3.3.5. Integración (búsqueda, Login, alta, reserva)



3.4. Modelo de datos



4. Diseño técnico

La aplicación está basada en Struts que es una implementación del patrón arquitectónico MVC. Modelo Vista Controlador. Es un Framework antiguo ya, pero sigue siendo válido.

4.1. Vista-Controlador: Validator y Parser helpers

A la arquitectura habitual de Struts, le hemos añadido unas clases que han simplificado el desarrollo en gran medida. En estas clases hemos invertido mucho tiempo en pruebas ya que según nuestra experiencia es en la validación de los datos del usuario, donde mas suelen fallar las aplicaciones.

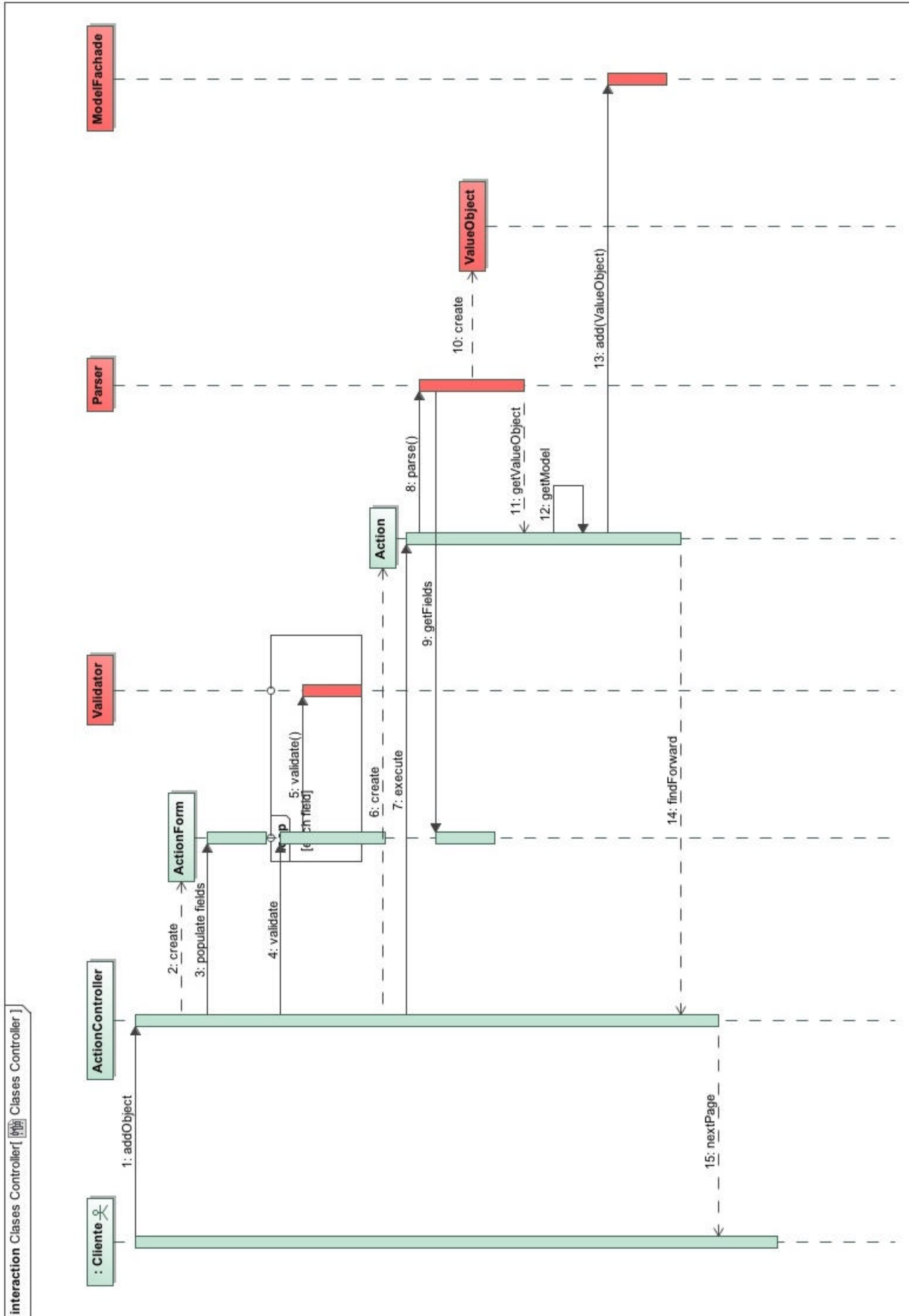
En nuestra aplicación el método validate del ActionForm se hace sólo con métodos de la clase Validator.

En el Action, la conversión de los campos del ActionForm a objetos se hace exclusivamente con la clase Parser.

A continuación exponemos sus tarjetas CRC. En el diagrama de secuencia posterior se puede ver su papel el la parte arquitectura Struts.

Clase: com.dblanch.utils.Validator
Responsabilidad: validar las entradas de los campos de los ActionForm, que son siempre strings
Colaboraciones: Parser, ActionForm

Clase: com.dblanch.utils.Parser
Responsabilidad: convertir las cadenas de los ActionForm en objeto,s como por ejemplo de un String a un Date
Colaboraciones: Validator. ActionForm, Action



interaction Clases Controller[Clases Controller]
: Cliente
ActionController
ActionForm
Validator
Action
Parser
ValueObject
ModelFachade
1: addObject
2: create
3: populate fields
4: validate
5: validate()
6: create
7: execute
8: parse()
9: getFields
10: create
11: getValueObject
12: getModel
13: add(ValueObject)
14: findForward
15: nextPage

4.2. El modelo

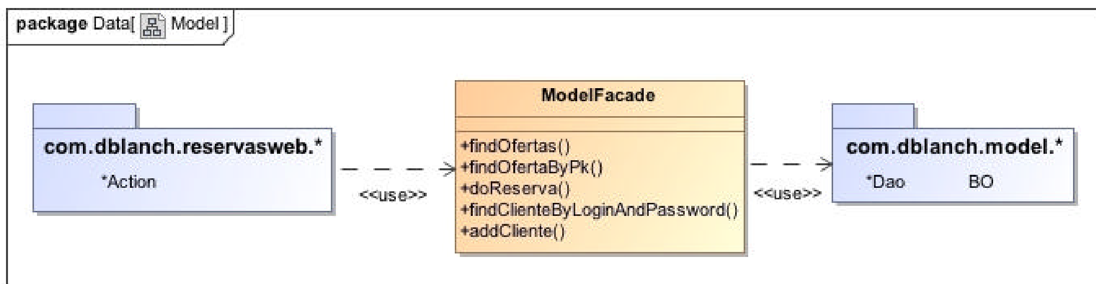
El modelo esta en su propio proyecto *reservasModel*, así permitimos este mismo componente se pueda usar en otro entorno que no sea el web.

4.2.1. La Fachada

Para acceder a sus funcionalidades, lo hacemos exclusivamente a través de un objeto **Fachada**, que es un objeto que permite ofrecer una interfaz unificada sencilla y que delega todos sus métodos en otros objetos.

En nuestra aplicación, la fachada delega en los DAO y a los objetos de negocio.

Los Action de Struts, para acceder al modelo solo interactúan con la fachada, nunca invocan Objetos de negocio aisladamente.



Hemos mentido un poco, nuestra fachada no es puramente una fachada, hace algo mas que delegar los métodos, obtiene un objeto java.sql.Connnection y se los pasa a los objetos en los que delega. También se encarga de gestionar la transacción y cerrar la conexión una vez terminado con el método.

Esto lo hace implementando otro Patrón que es el Template. El método java.sql.Connection getConnection() es abstracto y las clases hijas lo implementan de diferente forma.

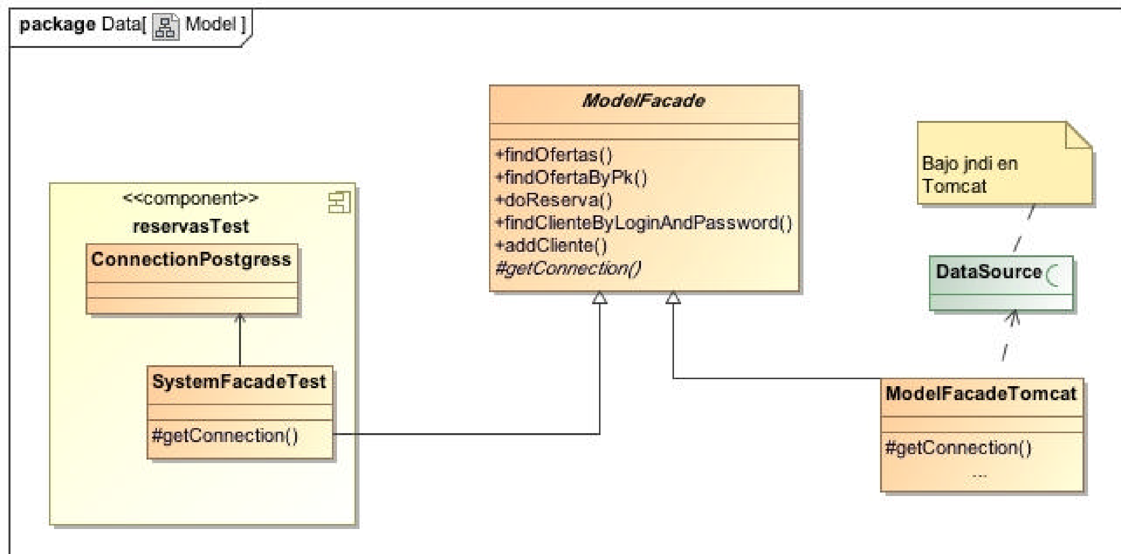
La forma de obtener la conexión a la base de datos en un contenedor J2EE es distinta a la forma de hacerlo fuera. En un contenedor se localiza un DataSource que provee un pool[1] de conexiones registrado bajo un nombre[2].

Fuera del contenedor se obtiene de la forma habitual.

---

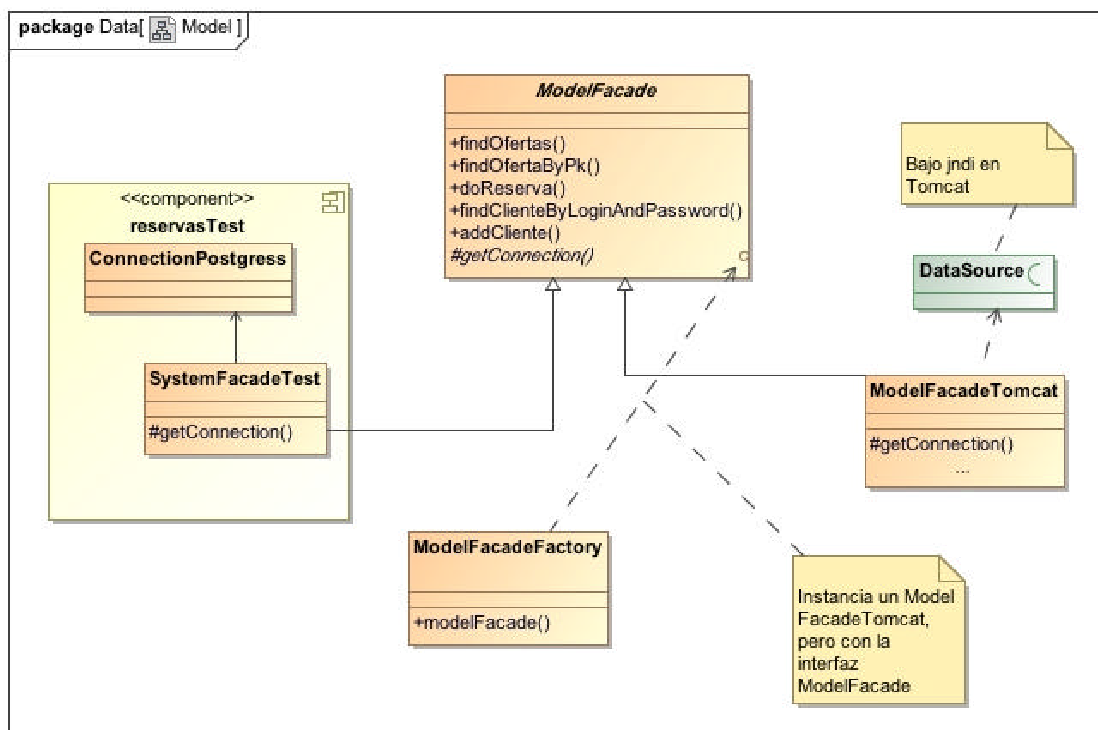
[1] Es un conjunto de conexiones, como el coste de obtener nuevas conexiones es elevado, los servidores J2EE suelen tener un grupo de estas abiertas y las reciclan entre sus peticiones.

[2] Consultar tecnología JNDI



Como vemos en el diagrama hemos implementado dos ModelFacade. Uno para el servidor Tomcat y otro para Test, con este último es con el que usamos para realizar las pruebas.

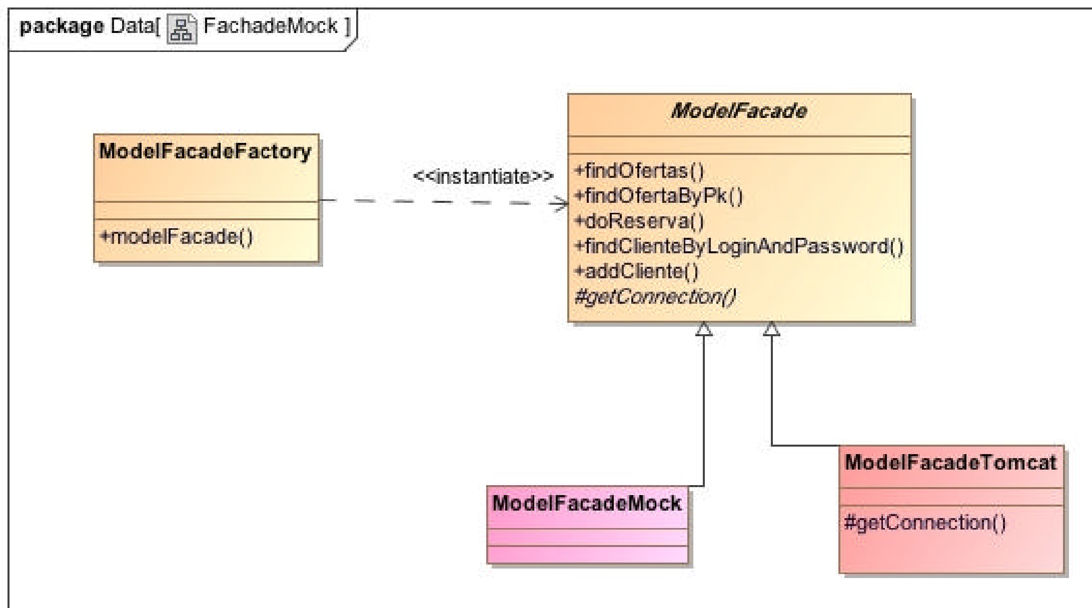
Aún queda un punto de complejidad por explicar, el objeto ModelFacadeTomcat no se instancia directamente con un operador new. Si no que hemos usado una clase Factory para instanciarlo.



La razón detrás de esta aparente complejidad es que esta arquitectura nos permite poder cambiar el objeto de implementación ModelFacade por otro que podría ser un ModelFacadeMock, por ejemplo.

Los Mock objects son objetos que no implementan funcionalidad real y sirven solo para pruebas.Los métodos que devuelven datos del objeto Mock, devuelven también objetos Mock. Esta estrategia se puede usar para desarrollar la parte web, sin depender de unos datos de la base de datos o para poder trabajar en la parte web, sin tener terminada la parte del modelo. Así se puede trabajar en paralelo en ambas partes.

Para cambiar de implementación de ModelFacade, sólo habría que cambiar el Factory.



5. Implementación

5.1. *Herramientas de desarrollo*

Eclipse con soporte para jee (Ganymede)

Java 1.6

Herramienta de uml (Magic Draw)

Tomcat

PostgreSql, jdbc y PgAdmin.

Junit (pruebas de unidad)

HttpUnit (pruebas de integración)

Metrics (para medir aspectos de calidad del software)

5.2. Proceso de instalación

Creación de base de datos

Se debe crear una base de datos llamada tfc y un usuario postgres, con pasword postgress

Luego ejecutar en orden los siguientes scripts

ReservasDateabase/DataBaseDefinition_Postgresql.sql (crea la base de datos)

ReservasDatabase/ProcedimientoReserva_Postgresql.sql (crea el script de la función para hacer la reserva)

ReservasDatabase/Integracion_Postgresql.sql (carga un pequeño juego de datos de prueba)

Puesta a punto del servidor j2ee

Necesitamos un DataSource para postgresql bajo el nombre jndi : jdbc/postgres que se conecte a la base de datos tfc

En tomcat se modifica el fichero context

```
<?xml version="1.0" encoding="UTF-8"?>


<Context>

        <WatchedResource>WEB-INF/web.xml</WatchedResource>

        <Resource name="jdbc/postgres" auth="Container" type="javax.sql.DataSource"
                driverClassName="org.postgresql.Driver"
                url="jdbc:postgresql:tfc"
                username="postgres" password="postgres" maxActive="20" maxIdle="10"
                maxWait="-1" />

</Context>
```

Obtencion de las librerías de struts, junit 1.3, httpUnit, y jdbc de Postrgres

Las librerías struts 1.1 y las librerías dependientes irán en

/reservasWeb/WebContent/WEB-INF/lib

Las librerías de junit.1.3 y httpUnit y el jdbc de postrares irán en el proyecto

/reservasLibs

El proyecto reservasTest ha de importar todas estas librerías.

Arrancar eclipse en el workspace y ejecutar la aplicación web

5.3. Juego de pruebas

Dentro del proyecto *reservasWeb* podemos ejecutar las siguientes pruebas:

Suite de pruebas de unidad (junit):

/reservasTest/src/com/dblanch/suite/AllTests.java

Pruebas de integración (httpUnit):

Primero arrancar la aplicación web.

/reservasTest/src/com/dblanch/integración/IntegracionTest.java

Pruebas de rendimiento:

/reservasTest/src/com/dblanch/integración/RendimientoEscrituraADisco.java

/reservasTest/src/com/dblanch/integración/RendimientoTransacciones.java

/reservasTest/src/com/dblanch/integración/TransaccionesMultithread.java

Con eclipse podemos ejecutar todas las pruebas que se encuentren en un direcctorio, que es como hemos hecho estas.

Resultado de la ejecución del juego de pruebas, de la aplicación, (reservasTest). como vemos tenemos 71 métodos de prueba, todos se han ejecutado con éxito.


```
<?xml version="1.0" encoding="UTF-8"?>
<testrun name="src" project="reservasTest" tests="71" started="71" failures="0"
errors="0" ignored="0">
        <testsuite name="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0">
                <testcase name="testSetNombre"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                <testcase name="testSetDescripcion"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                <testcase name="testSetFechaInicio"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                <testcase name="testSetFechaFin"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                <testcase name="testSetPlazasDisponibles"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                <testcase name="testSetDuracion"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                <testcase name="testComposeSQL"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
        </testsuite>
        <testsuite name="com.dblanch.integration.IntegracionTest" time="7.228">
                <testcase name="testAltaNombreDuplicado"
classname="com.dblanch.integration.IntegracionTest" time="3.652" />
                <testcase name="testReservaNuevoUsuario"
classname="com.dblanch.integration.IntegracionTest" time="1.851" />
                <testcase name="testReservaLogin"
classname="com.dblanch.integration.IntegracionTest" time="1.725" />
        </testsuite>
        <testsuite name="com.dblanch.frontend.SystemFacadeAbstractTest" time="0.064">
                <testcase name="testFindOfertas"
classname="com.dblanch.frontend.SystemFacadeAbstractTest" time="0.038" />
                <testcase name="testAddClienteTest"
classname="com.dblanch.frontend.SystemFacadeAbstractTest" time="0.026" />
        </testsuite>
        <testsuite name="com.dblanch.utils.ParserTest" time="0.002">
                <testcase name="testParseDate" classname="com.dblanch.utils.ParserTest"
time="0.001" />
```

```
                <testcase name="testParseInteger"
classname="com.dblanch.utils.ParserTest" time="0.001" />
        </testsuite>
        <testsuite name="com.dblanch.cliente.ClienteVOTest" time="0.004">
                <testcase name="testToString"
classname="com.dblanch.cliente.ClienteVOTest" time="0.004" />
        </testsuite>
        <testsuite name="com.dblanch.cliente.ClienteDAOTest" time="0.083">
                <testcase name="testAddDuplicate"
classname="com.dblanch.cliente.ClienteDAOTest" time="0.028" />
                <testcase name="testAdd" classname="com.dblanch.cliente.ClienteDAOTest"
time="0.017" />
                <testcase name="testFindClienteByLoginAndPassword"
classname="com.dblanch.cliente.ClienteDAOTest" time="0.038" />
        </testsuite>
        <testsuite name="All tests" time="0.495">
                <testsuite name="com.dblanch.oferta.OfertaDAOTest" time="0.058">
                        <testcase name="testFindOfertas"
classname="com.dblanch.oferta.OfertaDAOTest" time="0.042" />
                        <testcase name="testFindOfertaByPk"
classname="com.dblanch.oferta.OfertaDAOTest" time="0.016" />
                </testsuite>
                <testsuite name="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.003">
                        <testcase name="testSetNombre"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.001" />
                        <testcase name="testSetDescripcion"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                        <testcase name="testSetFechaInicio"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                        <testcase name="testSetFechaFin"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.001" />
                        <testcase name="testSetPlazasDisponibles"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                        <testcase name="testSetDuracion"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.0" />
                        <testcase name="testComposeSQL"
classname="com.dblanch.oferta.OfertaQueryBuilderTest" time="0.001" />
                </testsuite>
                <testsuite name="com.dblanch.reserva.ReservaCommandTest" time="0.283">
                        <testcase name="testReservaNormal"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.028" />
                        <testcase name="testReservaPlazasInsuficientes"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.027" />
                        <testcase name="testReservaNingunaPlaza"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.119" />
                        <testcase name="testUpdateOferta"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.109" />
                </testsuite>
                <testsuite
name="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.007">
                        <testcase name="testNingunCritierioDeBusqueda"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.005"
/>
                        <testcase name="testFechaInicio"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.0"
/>
                        <testcase name="testFechaFin"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.001"
/>
                        <testcase name="testPlazasDisponiblesNoValido"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.001"
/>
                        <testcase name="testDuracionNoValido"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.0"
/>
                </testsuite>
                <testsuite name="com.dblanch.utils.ValidatorTest" time="0.003">
                        <testcase name="testFechaValida"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
                        <testcase name="testRangoFechasCorrecto"
classname="com.dblanch.utils.ValidatorTest" time="0.001" />
                        <testcase name="testNumeroNatural"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
                        <testcase name="testValorNulo"
classname="com.dblanch.utils.ValidatorTest" time="0.002" />
                        <testcase name="testExceedsMaxSize"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
```

```
                        <testcase name="testNotReachMinSize"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
                </testsuite>
                <testsuite name="com.dblanch.utils.ParserTest" time="0.001">
                        <testcase name="testParseDate"
classname="com.dblanch.utils.ParserTest" time="0.001" />
                        <testcase name="testParseInteger"
classname="com.dblanch.utils.ParserTest" time="0.0" />
                </testsuite>
                <testsuite name="com.dblanch.frontend.SystemFacadeAbstractTest"
time="0.051">
                        <testcase name="testFindOfertas"
classname="com.dblanch.frontend.SystemFacadeAbstractTest" time="0.028" />
                        <testcase name="testAddClienteTest"
classname="com.dblanch.frontend.SystemFacadeAbstractTest" time="0.023" />
                </testsuite>
                <testsuite name="com.dblanch.reservasweb.reserva.ReservaFormTest"
time="0.001">
                        <testcase name="testPlazasDeseadasValido"
classname="com.dblanch.reservasweb.reserva.ReservaFormTest" time="0.001" />
                </testsuite>
                <testsuite name="com.dblanch.cliente.ClienteVOTest" time="0.001">
                        <testcase name="testToString"
classname="com.dblanch.cliente.ClienteVOTest" time="0.001" />
                </testsuite>
                <testsuite name="com.dblanch.cliente.ClienteDAOTest" time="0.087">
                        <testcase name="testAddDuplicate"
classname="com.dblanch.cliente.ClienteDAOTest" time="0.025" />
                        <testcase name="testAdd"
classname="com.dblanch.cliente.ClienteDAOTest" time="0.026" />
                        <testcase name="testFindClienteByLoginAndPassword"
classname="com.dblanch.cliente.ClienteDAOTest" time="0.036" />
                </testsuite>
        </testsuite>
        <testsuite name="com.dblanch.reservasweb.cliente.ClienteParserTest" time="0.001">
                <testcase name="testParse"
classname="com.dblanch.reservasweb.cliente.ClienteParserTest" time="0.001" />
        </testsuite>
        <testsuite name="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest"
time="0.003">
                <testcase name="testNingunCritierioDeBusqueda"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.001"
/>
                <testcase name="testFechaInicio"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.0"
/>
                <testcase name="testFechaFin"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.001"
/>
                <testcase name="testPlazasDisponiblesNoValido"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.001"
/>
                <testcase name="testDuracionNoValido"
classname="com.dblanch.reservasweb.busquedaofertas.BusquedaOfertasFormTest" time="0.0"
/>
        </testsuite>
        <testsuite name="com.dblanch.reservasweb.reserva.ReservaFormTest" time="0.0">
                <testcase name="testPlazasDeseadasValido"
classname="com.dblanch.reservasweb.reserva.ReservaFormTest" time="0.0" />
        </testsuite>
        <testsuite name="com.dblanch.reserva.ReservaCommandTest" time="0.104">
                <testcase name="testReservaNormal"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.022" />
                <testcase name="testReservaPlazasInsuficientes"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.022" />
                <testcase name="testReservaNingunaPlaza"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.027" />
                <testcase name="testUpdateOferta"
classname="com.dblanch.reserva.ReservaCommandTest" time="0.032" />
        </testsuite>
        <testsuite name="com.dblanch.utils.ValidatorTest" time="0.004">
                <testcase name="testFechaValida"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
                <testcase name="testRangoFechasCorrecto"
classname="com.dblanch.utils.ValidatorTest" time="0.001" />
                <testcase name="testNumeroNatural"
classname="com.dblanch.utils.ValidatorTest" time="0.003" />
```

```
                <testcase name="testValorNulo"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
                <testcase name="testExceedsMaxSize"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
                <testcase name="testNotReachMinSize"
classname="com.dblanch.utils.ValidatorTest" time="0.0" />
        </testsuite>
        <testsuite name="com.dblanch.oferta.OfertaVOTest" time="0.001">
                <testcase name="testGetDuracion"
classname="com.dblanch.oferta.OfertaVOTest" time="0.001" />
        </testsuite>
        <testsuite name="com.dblanch.oferta.OfertaDAOTest" time="0.086">
                <testcase name="testFindOfertas"
classname="com.dblanch.oferta.OfertaDAOTest" time="0.06" />
                <testcase name="testFindOfertaByPk"
classname="com.dblanch.oferta.OfertaDAOTest" time="0.026" />
        </testsuite>
</testrun>
```

6. Conclusiones

Uso de metodología ágil

No hemos podido aplicar una metodología de desarrollo ágil completamente, porque nos faltaba la parte fundamental que es tener al cliente dentro del equipo de desarrollo, para verificar el producto y proponer cambios. Pero hemos usado otras prácticas que si podemos evaluar, como los ciclos de desarrollo cortos y las pruebas automatizadas.

Ciclos de desarrollo cortos

Hemos afrontado el proyecto como una sucesión de pequeños subprogramas completos.

Como primera consecuencia, el centrarnos en una pequeña parte nos ha permitido definir los casos de uso perfectamente, sin flecos, en especial la interacción con el sistema de forma excelente.

La notación de casos de uso de Alistair Cockburn y los diagramas de actividad han demostrado ser unas excelentes herramientas para esta labor.

Pruebas de unidad automatizadas

Han sido mucho mas laboriosas de hacer de lo que pensaba, creo que tardábamos hasta cuatro o cinco veces mas en hacer las pruebas que en escribir el código. Y escribir código que se pueda probar ya es de por si mas laborioso.

Sin embargo creemos que han sido muy útiles, apenas hemos pasado tiempo depurando código. Y no hemos perdido nada de tiempo en la integración.

Creemos que en el punto 1.3.2.1. queda demostrado la necesidad de hacer pruebas de unidad, si se quiere probar el código. Solo con las pruebas de integración es imposible.

Hoy en día todavía se siguen dejando las pruebas para el final en los desarrollos, entonces solo podrán ser pruebas de integración. Y como hemos visto la

complejidad de una prueba completa de integración crece exponencialmente en función del número de componentes sobre los que opere, mientras que el coste de las pruebas de unidad es lineal

En definitiva, creemos que las pruebas de unidad deben ser un paso necesario en todo desarrollo de software.

Metodología de tests y herramientas.

Creemos que hemos logrado un buen sistema para hacer pruebas con Eclipse, la forma de separar las pruebas en un proyecto aparte es muy útil.

7. Bibliografía

JUnit Recipes , JBRainsberr Editorial Manning

Head First Design Patterns, Eric Freeman &Elisabeth Freeman Editorial O'Reilly

Piensa en Java, Bruce Eckel Editorial Prentice Hall

Core Java Volume II Advanced Features, Cay S. Horstmann Editorial PH PTR

Struts KICK START, James Turner and Kevin Bedell