Vision CondorTrip

Condor Trips es una propuesta software orientada a ofrecer un servicio de viajes turísticos. En seguida se mostrará un caso de uso frecuente, que se proyecta el producto va a presentar:

Un cliente se acerca a cualquier punto CondorTrip ubicado a lo largo y ancho del país; Seguidamente pide a un funcionario que le bosqueje un plan turístico, el Funcionario le muestra los planes turísticos todo incluido que se encuentran en la página web, dando detalles de destinos turísticos como de precios. si el cliente busca algo más personalizado puede libremente escoger entre cualquier continente del mundo y país, pudiendo escoger el hotel que más le guste, asistir a city tours para conocer más la ciudad y la cultura que está visitando. El cliente con plena libertad puede hacer un plan compuesto, por ejemplo de ir a Barcelona por 5 días y después pasar por París por 4 días y terminar en las bellas playas de Eagle en Aruba. Los precios de los paquetes personalizados a menudo son un poco más elevados que los paquetes todo incluido, pero siempre hay sus excepciones, por ejemplo si es cliente regular CondorTrip es una excepción.

Una vez el cliente haya decidido puede pagar en efectivo, cheques, con tarjeta de crédito, débito, a cuotas o de contado. El cliente en el tiempo establecido para modificaciones, el cual dependerá de los tiempos fijados para el viaje, puede ir a las instalaciones y modificar el hotel, el plan de alimentación o los tours que decidió reservar, ya sea añadiendo o eliminando características de ellos a su gusto**, sin ningún cargo adicional**, la única excepción a esta regla es si el cliente decide cambiar los países que decidió viajar, sólo en este caso se añadirán cargos al cliente.

El sueño de CondorTrip es posicionarse como una de las mejores agencias de viajes del país y por qué no del mundo, y ese sueño lo cumpliremos, cumpliendo los suyos.

Atte: Toda la comunidad que hace posible CondorTrip

Priorización de requisitos:

La priorización de los requisitos se basó en los requisitos que más tuvieran valor para el cliente, los cuales escogimos los siguientes 3:

- **Requisito 1**
-
- **Requisito 2**
-
- **Requisito 6**

En el proceso de **incepción** del desarrollo se hizo una especificación simple de los casos de uso de estos 3 requisitos, haciendo énfasis en la etapa de análisis, visión del producto, alcance y requisitos. En la **primera iteración** se hicieron los diagramas de casos de uso detallados, y se empezó la parte de diseño diagramas de clase diagramas de interacción, y se hizo la correspondiente elaboración; como el ciclo de desarrollo de software que se está siguiendo es iterativo**, RUP**, No hay tanto como una secuencialidad en **análisis-diseño e implementación** sino que en cada punto de la iteración se trabaja el análisis, diseño y la implementación para que sea en realidad un proceso iterativo, y no uno en cascada que sabemos que para los proyectos software independiente de su magnitud no es muy adecuado.

Segunda y última iteración

Se hizo la siguiente priorización de requisitos:

- Requisito 4
- Requisito 2
- Requisito 3
- Requisito 5

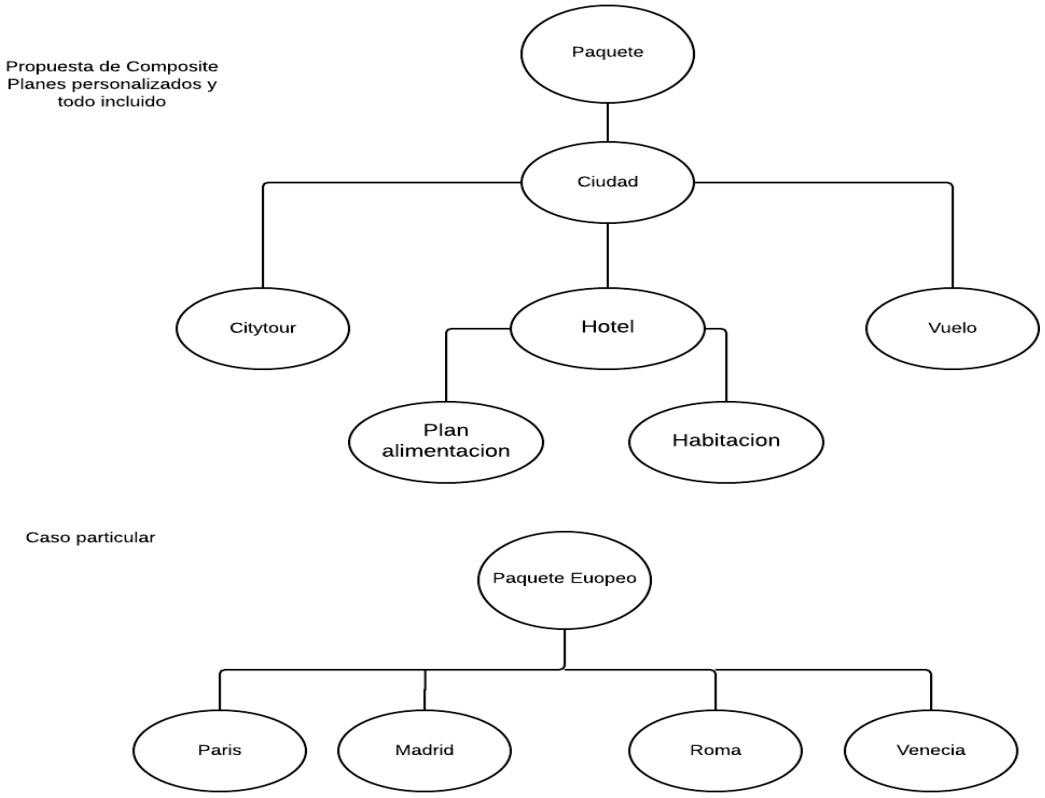
Volvimos a añadir el requisito 2 a esta nueva iteración ya que en la primera iteración no se alcanzó a terminar este requisito ya que tenía muchos conceptos, que gracias al profesor
Se alcanzaron a conseguir en esta iteración

Rationale Arquitectonico:
Base de datos HSQLB: Se eligió la HSQLDB por su tamaño y rapidez de operación con los datos relacionales, nos provee las mismas capacidades que en la mayoría de bases de datos y no es tan pesada. Por esa misma razón ha tenido una muy buena acogida en los proyectos.

Servidor Sockets: Entre los dos servidores que hemos aprendido a programar con el ingeniero Libardo Pantoja, es el que más entendimiento de sus bases tenemos, haciendo que si hay un problema con el servidor sepamos mucho más rápido de que manera actuar y por donde abordar el problema.

Requisito 1: Se decidió que una interface fuera la que ofreciera los servicios de modificar, actualizar, eliminar y consultar, e instanciar al servidor socket, y no el servidor socket directamente para hacer extensible la comunicación cliente-servidor de manera que si se quisiera sin ningún problema se añadiría una conexión por RMI

Requisito 2: En el requisito 2 se tomó la decisión de utilizar el patrón de diseño composite para mejorar el manejo de los paquetes personalizados.
Decidimos proponer un diseño composite diferente del propuesto por el Ingeniero, nuestra propuesta es la siguiente:



Al elegir el patrón composite nos premitió manejar los elementos compuestos de una manera más fácil a la hora de agregar y eliminar tanto hojas como compuestos, pero a la hora de enviar un paquete por json fue necesario hacer una distinción entre los objetos; como sabemos el composite permite tratar a componentes hojas y compuestos de la misma forma, y ese fue una de las consecuencias al elegir el patrón composite, por eso en la deserialización y serializacion de objetos compuestos tales como hotel, ciudad o paquete, fue necesario agregar tag cases indicando de qué tipo de componente se está tratando para hacer el mapeo del json a una instancia paquete que evidencie los datos del json.
.
Una solución aparente sería manejar la conexión al servidor por RMI y no por sockets.

Características adicionales:

Se añadieron funcionalidades tales como “ver carrito” que dejan al funcionario moniterar el paquete que está creando, puede ser que se haya equivocado a la hora de seleccionar un item del paquete o que el cliente haya cambiado de opinión es necesario hacer un delete no dentro de la base de datos sino dentro del composite que después se carga a la base de datos.

Consecuencias:

Como se puede evidenciar en el hay un uso extensivo del método “instanceof” lo cual muestra que hay una necesidad de diferenciar los componentes de un paquete para realizar una busqueda de un componente por su código único o “id” ya que un id puede estar repetido de un componente a otro por ejemplo un hotel con id igual 1 (id=1) y una ciudad con id igual a 1(id =1) es necesario hacer esa distinción.

Por otro lado en el código no hay un agregarHotel, agregarPlanAlimentacion, agregarCityTour, agregarCiudad, agregarVuelo, agregarHotel, tampoco un eliminarHotel, eliminarCityTour, eliminarPlanAlimentacion, eliminarVuelo, etc. Esto nos permitió una mayor flexibilidad a la hora de añadir y eliminar componentes a un paquete.

Requisito 3: Los paquete todo incluído se manejan de la misma forma que los paquetes personalizados, pues no hay razón para hacer clases diferentes para una y para otra, de manera pues que los paquetes todo incluído también se manejaron con el patrón composite y se evidenció también los pros y contras de esta decisión.

Requisito 4: En la venta de los paquetes, no se hizo uso de un patrón en específico, lo que se utilizo fue el paso de argumentos a los constructores de los jframe’s para capturar los datos del anterior jframe que lo creó, de esta manera no se pierde la información capturada en un previo jframe que es relevante para la nueva ventana creada

Requisito 5: En la autenticación se hizo uso del patrón fábrica abstracta con una modificación:

- Se definió a la fábrica como una fábrica concreta

Se manejó como interfaz común a GUPrincipal de las que heredan GUIPrincipalAdministrador y GUIPrincipalFuncionario, la fábrica devuelve un tipo GUIPrincipal y depende del tipo que se mande a la fábrica abstracta esa es la que devuelve.

Los usuarios se consultan a la base de datos en busca del nombre de usuario y contraseña si alguna de estas no coinciden se reportará un error de inicio de sesión

Requisito 6: Al ser un CRUD muy parecido al del requisito 1, las decisiones tomadas para el requisito 1, fueron “transferidas” al requisito 6, pues el requisito 6 no tiene ninguna otra decisión arquitectónica diferente de requisito 1.

Pruebas:
Se hicieron pruebas de gestión cliente y gestión usuarios, gestionPlanesPersonalizados, gestionPlanesTodoIncluido, haciendo assertiones sobre métodos de estas clases y verificar si se está haciendo lo que le corresponde al método, a continuación descripción detallada del alcance de las pruebas: de software realizadas para el sistema:

Ya que el requisito 1 y 6 comparten un grado alto de similitud, estos dos requisitos comparten las pruebas de software

Pruebas de software (Requisitos 1 y 6):

El Alcance de las pruebas de software aplicadas al sistema con respecto al requisito 1 y 6 es garantizar el correcto funcionamiento de los items a ser probados, para que la aplicación de escritorio de cóndor Trips sea confiable y se garantice la integridad de los datos de los clientes y usuarios del sistema.

Los ítems a ser probados fueron:

-Creación de un nuevo usuario/cliente al sistema:
Se espera que el sistema sea capaz de agregar un nuevo usuario/cliente al sistema, para esto el sistema deberá consultar si el usuario/cliente a ingresar ya esta registrado en la base de datos del sistema, sera capaz de informar al funcionario/administrador que esta haciendo uso de la aplicación si este fue agregado exitosamente o si ocurrió un error en el proceso de la petición.

-La actualización de usuario/cliente existente en el sistema:
Se espera que el sistema sea capaz de editar la información existe de un usuario/cliente registrado en la base de datos del sistema, el sistema debe ser capaz de consultar si el cliente/usuario a editar esta registrado en la base de datos. Al finalizar la aplicación debe informar si el cliente fue editado con éxito o si ocurrió un error en mientras se ejecutaba la petición.

-La consulta de la información de un usuario/cliente del sistema:
Se espera que el sistema sea capaz de consultar la información relacionada con un usuario/cliente especifico registrado en la base de datos y en caso de que este registro no existe el sistema deberá informar al funcionario/administrador que inicio la funcionalidad

-La eliminación de un usuario/cliente del sistema
El sistema debe ser capaz de eliminar la información de un cliente/usuario registrado en la base de datos del sistema. El sistema debe informar al administrador/funcionario que inicio la operación.

En el código fuente de la aplicación de escritorio (condorTrips) se encuentran las pruebas codificadas.

Pruebas de software (Requisitos 3):

El Alcance de las pruebas de software aplicadas al sistema con respecto al requisito 3 es garantizar el correcto funcionamiento de los items a ser probados, para que la aplicación de escritorio de cóndor Trips sea confiable, segura y se garantice la integridad de los datos de los clientes y usuarios del sistema.

Los ítems a ser probados fueron:

-Verificación correcta de un usuario que desea ingresar al sistema:

El sistema sera capaz de hacer una consulta a la base de datos para corroborar la información suministrada por un usuario que desea iniciar sesión en la aplicación de cóndor Trips, El sistema sera capaz de unicamente permitir el acceso a la pagina principal de la aplicación unicamente si el usuario existe en la base de datos y ademas si esta información es correcta.
El sistema sera capaz ante de dirigir dependiendo de su tipo de usuario (administrador o funcionario) al su pagina principal correspondiente.

-Verificación del comportamiento del sistema ante la eliminación de un usuario del sistema, el cual previamente inicio sesión:

El sistema sera capaz de verificar cada determinado momento si el usuario el cual ha iniciado sesión se encuentra activo o existe en la base de datos del sistema, es decir si en algún momento el usuario previamente logeado fue eliminado de la base de datos, el sistema deberá cerrar la ventana principal del usuario.

Pruebas PlanesPersonalizados:

- Se verifica que el plan realizado por un cliente se guarde correctamente en la base de datos, haya una correcta serialización y deserialización que es un componente crítico para este requisito.
-

Pruebas Planes Todo Incluido:

- Los planes personalizados y todo incluido comporten todo, la estructura de composite como el método de agregar plan todo incluido, si acierta las pruebas los planes personalizados acierta planes todo incluido.

Interfaz:

Se decidió trabajar tabbed panes para facilitar la navegación de los funcionarios a través del sistema; se pudiera haber hecho presionando botones y que se pasarán los paneles uno por toque, pero nos pareció mejor navegabilidad y ubicación que el funcionario pudiera ver las pestañas de modificar, eliminar, consultar y crear, para darle un mejor contexto.

Singleton:

Cuando se utilizaba el patrón de diseño singleton a las vistas que queríamos que tuvieran esta característica, el programa mostraba un error cuando se intentaba añadir a la ventana padre de nuevo una ventana que ya tenía, y se bloqueaba. Lo que se planteó fue hacer una clase que tuviera por atributo ventanaUnica: boolean para que una instancia de esta clase se psara como argumento entra las vistas de manera que cuando se saliera de la vista en cuestion cambiaba el atributo pemitiendo que se instancia otra ventana, ya que la previo se cerró.