Reporte
Experimento controlado

Resumen

1. Objetivos

El objetivo de este experimento es validar el modelo de Rationale arquitectónico definido para capturar y documentar las razones arquitecturales desde el código fuente. Este modelo se mantuvo en constante evolución debido a los comentarios positivos y negativos que se brindaron en las diferentes socializaciones de este estudio. Este experimento busca comprobar a través de una muestra poblacional, que existe una significancia estadística en la eficiencia, efectividad y la comprensión cuando se realizan cambios arquitecturales y estos se documentan de forma tradicional o a través de anotaciones de código fuente con el modelo de Rationale arquitectónico planteado en este estudio.

2. Planificación

Hipótesis

La hipótesis de este experimento es que la existencia de anotaciones de código fuente con información del Rationale arquitectónico mejora la mantenibilidad de la arquitectura con respecto a la eficiencia, efectividad y comprensión al realizar cambios arquitecturales. Para confirmar esta hipótesis se formula la hipótesis nula y alternativa para cada aspecto de interés.

Eficiencia:

- NULA: La eficiencia al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es menor o igual a la eficiencia de realizar el cambio sin anotaciones de código.
- ALTERNATIVA: La eficiencia al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es mayor a la eficiencia de realizar el cambio sin anotaciones de código.

Efectividad:

- NULA: La efectividad al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es menor o igual a la efectividad de realizar el cambio sin anotaciones de código.
- ALTERNATIVA: La efectividad al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es mayor a la efectividad de realizar el cambio sin anotaciones de código.

Comprensión:

- NULA: La comprensión de la Arquitectura y su Rationale al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es menor o igual a la comprensión al realizar el cambio sin anotaciones de código.
- ALTERNATIVA: La comprensión de la Arquitectura y su Rationale al realizar un cambio arquitectural en un sistema con la documentación del Rationale arquitectónico en anotaciones de código es mayor a la comprensión al realizar el cambio sin anotaciones de código.

Variables

Independiente:

En este trabajo la única variable independiente es la existencia o no de anotaciones de código fuente con información del Rationale arquitectónico. Sus únicos valores posibles son “Si” o “No”, esta variable es categórica puesto que representa si un participante debe realizar los cambios a un código con anotaciones de código o sin ellas.

Dependiente:

Para el cálculo de las variables dependientes eficiencia y esfuerzo, es necesario medir otras variables como el tiempo y el nivel de correctitud en la realización de las tareas. Además de definir los valores estimados, con el fin de obtener los resultados de las variables eficiencia y efectividad en términos de porcentajes. Otra variable dependiente es el nivel de comprensión de la Arquitectura y su Rationale, esta se comprende en valores entre 0 y 100. Los participantes deben realizar 3 tareas a las cuales se les califica según el nivel de correctitud de cada tarea, este nivel se describe a continuación:

Nivel Correctitud Tarea 1 (NC1)	
Valor	**Descripción**
100	No realizó el cambio.
200	Realizó cambios, pero no logró terminar la tarea.
300	No completó la tarea, pero mantuvo el diseño establecido.
400	Completó la tarea pero sin mantener el diseño inicial
500	Completó la tarea manteniendo el diseño original

Nivel Correctitud tarea 2 (NC2)	
Valor	**Descripción**
100	No realizó el cambio.
200	Realizó modificaciones, pero no logró cambios funcionales.
300	Hizo la tarea cambiando el diseño y manteniéndose dentro de la arquitectura.
400	Hizo la tarea cambiando el diseño y la arquitectura.
500	Hizo la tarea manteniendo el diseño y cambiando la arquitectura.

Nivel Correctitud tarea 3 (NC3)

Valor	Descripción
100	No escribió la documentación.
200	Documentó los cambios a nivel funcional.
300	Documentó los cambios a nivel de diseño.
400	Documentó los cambios a nivel arquitectural sin su Rationale.
500	Documentó los cambios arquitectónicos junto con su Rationale.

La suma de los niveles de correctitud de todas las tareas se define como **'NCT'** y está expresada de la siguiente manera:

$$NCT = \sum_{i=1}^{n} NCi$$

(*n*: cantidad total de tareas)

El tiempo en que un participante realiza una tarea se denota por la letra **'t'**. El tiempo en que tarda en realizar todas las tareas **'T'** se define como la sumatoria de cada uno de los tiempos tomados en cada tarea, tal como se expresa en la siguiente ecuación:

$$T = \sum_{i=1}^{n} ti$$

Los valores estimados que se definen para el tiempo total y la correctitud total se muestran a continuación: el tiempo estimado para la primera, segunda y tercera tarea es de 1 hora, 1 hora y 20 minutos respectivamente, sumando un tiempo total estimado **'Te'** para realizar todas las tareas de 140 min.

$$\boldsymbol{Te = 140}$$

Nivel de correctitud total estimado **'NCTe'** en la realización de todas las tareas asignadas.

$$NCTe = \boldsymbol{NC1\,(500) + NC2\,(500) + NC3\,(500) = 1500}$$

Eficiencia

Valor medido respecto a la correctitud y el tiempo total obtenidos en la realización de las tareas, la eficiencia **'E'** se define a continuación como:

$$E = \frac{NCT}{T}$$

Para dar un valor de la eficiencia en términos de porcentaje es necesario establecer un punto de referencia en la realización de las tareas, el cual se define como:

$$ER = \frac{NCTr}{Te}$$

Donde el nivel de correctitud total referente **'NCTr'** es:

$$NCTr = NC1\,(400) + NC2\,(500) + NC3\,(500) = 1400$$

Con lo cual conseguimos el siguiente valor de eficiencia referente **'ER'**:

$$ER = \frac{1400}{140} = 10\ \frac{NCT}{min}$$

Finalmente obtenemos el porcentaje de eficiencia por cada participante:

$$\%\ eficiencia = \frac{E}{ER} * 100$$

Efectividad

Valor medido mediante la correctitud en la realización de todas las tareas dividido por la correctitud total esperada. En este trabajo se define la efectividad **'EF'** como:

$$EF = \frac{NCT}{NCTe}$$

Finalmente obtenemos el porcentaje de efectividad de cada participante mediante:

$$\%\, efectividad = EF * 100$$

Comprensión

el nivel de comprensión de los participantes al realizar cada una de las tareas. Para ello se establecen nuevos criterios que permiten definir el nivel de entendimiento del Rationale arquitectónico cuando los participantes realizan las tareas. Este nivel de comprensión se evalúa mediante un cuestionario adjunto en el ítem Materiales y recursos. El formulario en cuestión contiene las siguientes preguntas:

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.
B. ¿Cuál es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)? Justifique su respuesta.
C. Explique los componentes que creó.
D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?
E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios? Justifique su respuesta.

Cada pregunta está dirigida a especificar las razones de las decisiones que se tomaron en el proceso de realización de las actividades. Cada pregunta tiene una calificación de 100 puntos para un total de 500 puntos, este valor se divide entre el número total de preguntas, dando como resultado máximo un nivel de comprensión **C** = 100. La puntuación de las respuestas se verifica de acuerdo con la correspondencia en la definición del Rationale brindada por la ISO/IEC/IEEE 42010. Además de contrastar las razones iniciales del arquitecto, por las que se define la arquitectura antes de realizar el experimento.

Con lo cual el nivel de comprensión del Rationale arquitectónico en la realización de las tareas se define como:

$$C = \frac{\sum_{i=1}^{n} Pi}{n}$$

Participantes

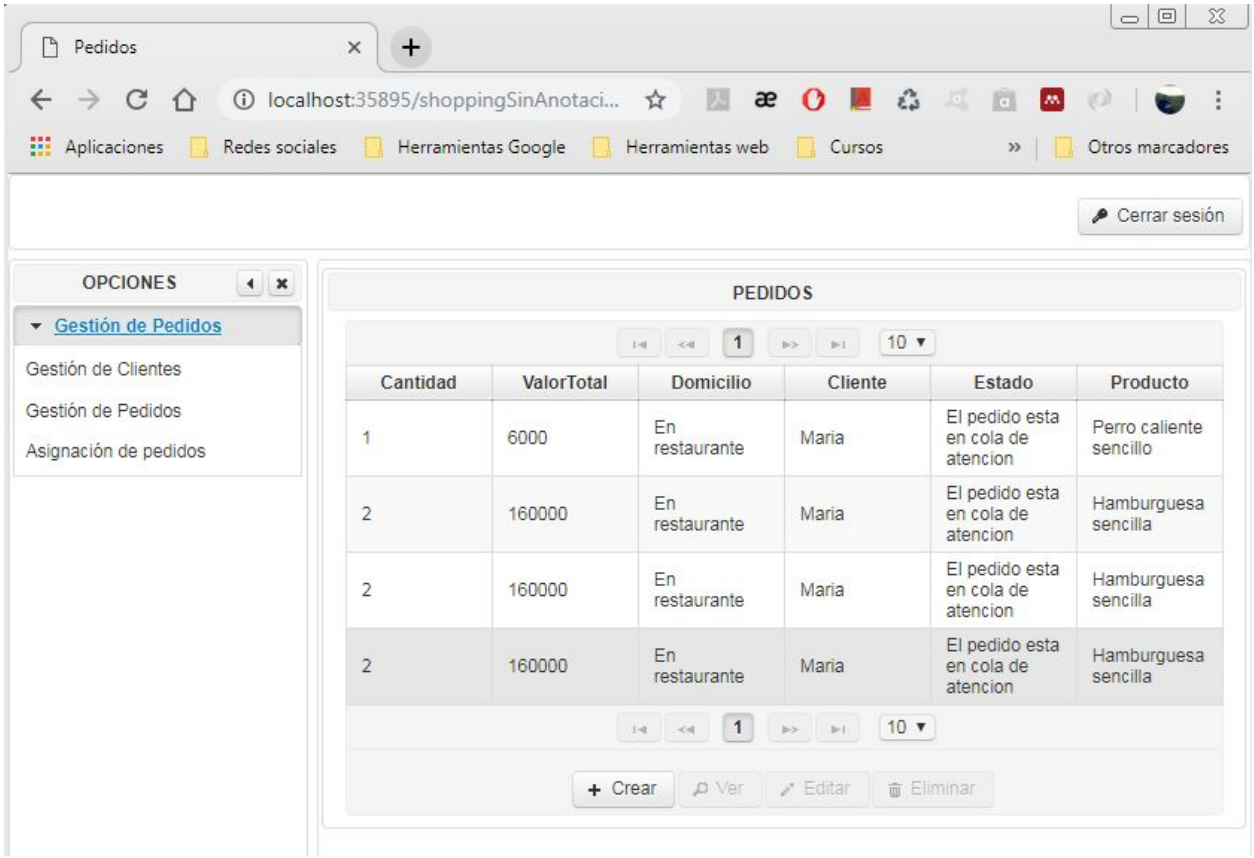
El grupo poblacional consta de 21 ingenieros de sistemas y electrónica, en este grupo se encuentran profesionales actualmente trabajando en la industria de desarrollo de software y tesistas de grado de la Universidad del Cauca. La mayoría de personas son hombres de aproximadamente 22 a 35 años de edad, con experiencia de por lo menos 2 años en desarrollo de software en el lenguaje de programación Java. Todas las personas habitan en la ciudad de Popayán en áreas cercanas a sitio del experimento. Se realiza un muestreo aleatorio asignando a cada participante un número consecutivo, la selección de cada participante se realiza generando un número del 1 al 21 y se contacta directamente a la persona por algún medio disponible (Redes sociales, Correo electrónico, Llamada telefónico o mensaje de Whatsapp), si esta persona por cualquier motivo no puede asistir al evento se contacta directamente a la persona que corresponda con el siguiente número consecutivo, hasta completar la muestra de 8 participantes.

Finalmente, al día del experimento llegan 7 personas de las cuales 1 de ellas llega tarde y se le presentan inconvenientes con la instalación y el despliegue de las herramientas necesarias para la realización de este experimento, por lo cual se decide extraer de la muestra poblacional y trabajar con una muestra de 6 participantes.

Material experimental

Para este experimento se realiza un sistema que permite la gestión de productos, ingredientes, tiendas, pedidos, usuarios, clientes y pagos. Las tecnologías utilizadas para este sistema son Java EE, Hibernate, Oracle 11g, EJB y Primefaces. Este sistema simula la gestión de los productos para pequeñas y medianas empresas dedicadas a la venta de productos alimenticios. Además de gestionar la información relacionada con los productos y sus ingredientes, este sistema permite configurar tiendas, categorías de productos, los usuarios que hacen uso del sistema junto con sus respectivos roles, la información de los clientes y los pedidos que estos realizan.

En las siguientes imágenes se puede observar algunas capturas de pantalla del programa de prueba.



Este código fuente se entrega con la documentación del sistema en un documento .docx en la carpeta reports y con un pdf con las actividades que deben realizar; para los participantes que tienen las anotaciones de código incorporadas con información del Rationale arquitectónico cuentan con los reportes generados por cada anotación marcada directamente en el código. El modelo de anotaciones ARAT utilizado para este experimento se muestra a continuación:

```java
@Documented
@Retention(RUNTIME)
@Target({METHOD, PACKAGE, TYPE})
public @interface Rationale3 {
    enum QualityAtribute {
        FUNCTIONAL_ADECUATION, PERFORMANCE, COMPATIBILITY, USABILITY,
        RELIABILITY, SECURITY, MAINTENANCE, PORTABILITY
    }
    String id() default "";
    boolean hiden() default false;
    String[] links() default {};
    QualityAtribute[] quality_attributes();
    String[] causes();
    String[] tactics() default {};
    String[] patterns() default {};
    String[] alternatives() default {};
    String[] decisions_record();
    String[] reasons();
}
```

Esta declaración además de tener los elementos del Rationale arquitectónico también tiene otros atributos de configuración que permiten ocultar los elementos marcados en el código en la generación de los reportes, además también tiene un elemento que permite agregar links a otras fuentes de información de Rationale arquitectónico. A continuación, se especifica cada uno de los atributos de la anotación. Nótese que los elementos de dividen en atributos de configuración y de información. Los atributos de información son un conjunto de listas, a diferencia de los atributos de configuración.

1. Atributos de configuración (opcionales):
    - id: String que permite identificar una anotación.
    - hiden: boolean que permite ocultar la información de una anotación en la generación del reporte. Por defecto viene con valor false
    - links: Lista de enlaces a otras fuentes de información
2. Atributos de información:
    - quality_atributes: Lista de atributos de calidad que quieren considerar.
    - causes: Lista de causas por las cuales es necesario cumplir con los atributos de calidad.
    - tactics (opcional/Recomendado): Lista de tácticas arquitecturales planteadas para lograr la consecución de los atributos de calidad.
    - patterns (opcional/Recomendado): Lista de patrones que complementan las tácticas arquitecturales.

- alternatives (opcional/Recomendado): Lista de alternativas que se consideran en el momento de tomar decisiones.
- decisions_record: Lista de decisiones que se toman en un determinado momento, para cumplir con un grupo de atributos de calidad específicos.
- reasons: Lista de razones que justifican o expresan el porqué de un grupo de decisiones.

Tareas

El experimento consiste en la realización de tres tareas para cada participante, estas tareas se definen como actividades en el documento adjunto en el directorio del código fuente. En el ítem Materiales y recursos se adjunta el documento con las actividades y el formulario de preguntas.

1. Mantenimiento aditivo
2. Mantenimiento correctivo
3. Documentación del Rationale Arquitectónico

Tarea 1 Mantenimiento aditivo

Se debe crear un módulo de software que permita gestionar los pagos relacionados a un pedido. Este módulo se va a construir de manera iterativa e incremental. Sin embargo, se puede definir la estructura inicial y la organización de los componentes, de tal manera que se puedan implementar las funcionalidades más prioritarias. La primera implementación que se debe realizar consta de verificar si un pedido se puede pagar, realizando una validación con el saldo del método de pago. Esta implementación debe seguir con los siguientes parámetros de entrada y salida:

- Entrada: Número que identifica el método de pago idMetodoPago (Long), tenga en cuenta que MetodoPago no se define todavía en base de datos, pero se puede trabajar con objetos quemados en memoria.
- Salida: Long que representa el saldo de la cuenta

Tarea 2 Mantenimiento correctivo

Suponga que ha pasado tiempo y que se empieza a ver afectado el rendimiento del sistema debido a la gran cantidad de peticiones que realizan los usuarios sobre el sistema. Con un software de pruebas de rendimiento y un monitor de control de peticiones se logra determinar que el componente más afectado es el componente de pagos que usted creó. Además de lo anteriormente mencionado, este componente utiliza servicios externos para consultar datos de los medios de pagos, es por eso que es necesario extraerlo del nivel de la aplicación principal y pasarlo a un entorno independiente, donde tenga recursos propios y pueda responder de

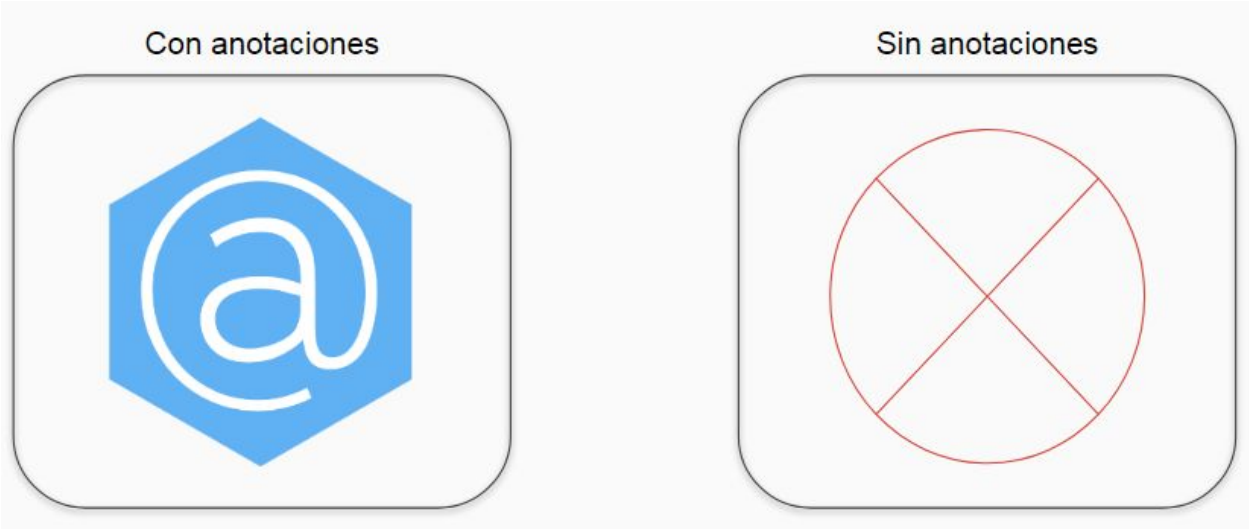
manera más eficiente a la cantidad de peticiones de los usuarios. Se quiere que este módulo sea totalmente independiente del resto de la aplicación, fácil de desplegar, de mantener y que permita el incremento de peticiones de los usuarios de manera controlada

Tarea 3 Documentación del Rationale

Documente los cambios realizados en las tareas anteriores según los artefactos que disponga para la documentación de la arquitectura. Si el tiempo no le alcanza para terminar con las tareas, por favor indique los cambios que planearía implementar lo más detallado posible (diferenciándolos con la palabra clave TODO:) y finalmente, responda las preguntas del formulario, que será entregado una vez indique que finalizó las modificaciones o el tiempo asignado para ellas haya terminado.

Diseño del experimento

Al igual que en el estudio exploratorio los participantes se dividen en dos grupos, uno con anotaciones de código con información del Rationale arquitectónico y otro sin estas anotaciones.



Procedimiento

Los participantes deben recibir una capacitación sobre los conceptos fundamentales relacionados con el Rationale arquitectónico y las anotaciones de código, de igual forma que en el primer experimento. Después de esto, a los participantes se les entrega el código fuente de acuerdo a la selección del grupo que le corresponda, para ello se seleccionan la mitad de los participantes en el orden de llegada y se lanza una moneda para cada uno, en caso de que la moneda caiga cara al participante se le asigna el código con las anotaciones de código, en caso contrario se le asigna el código sin las anotaciones. Los participantes que no lanzan moneda se

les asigna el código fuente dependiendo de cuántos faltan para nivelar el número de participantes por grupo. Después de esto, se les da una introducción de los elementos del sistema y la organización actual del mismo. Una vez terminada la introducción se hace entrega de las actividades que deben realizar y empieza a contar el tiempo para la realización de las tareas. A medida que los participantes terminan se retiran los artefactos del experimento y se capturan los tiempos de cada actividad por participante. Finalmente, se hace entrega de los formularios con las preguntas especificadas en el ítem Materiales y recursos.

Ejecución

El procedimiento establecido en el ítem anterior se lleva a cabo sin ningún contratiempo, los participantes reciben la inducción de los conceptos relevantes, se les entrega el código fuente respectivo, se capturan los tiempos a medida que terminan las actividades y se les entrega el formulario con las preguntas relacionadas con la actividad y el Rationale arquitectónico generado de esta.

Tiempos

Grupo Sin Anotaciones GSA				
Participante	**t1**	**t2**	**t3**	**T(min)**
Javier A.	60	60	20	140
Eduar T.	60	60	20	140
Santiago G.	60	60	20	140
Grupo Con Anotaciones GCA				
Edwin M.	60	60	20	140
Santiago P.	60	60	20	140
Carlos M.	60	60	20	140

Como se puede observar el tiempo es igual para todos los participantes debido a que no fue suficiente para terminar completamente las actividades y se debió cortar en el límite para continuar con las demás actividades hasta finalizar con el tiempo establecido para el experimento. A diferencia del estudio exploratorio anterior, los participantes contaban con el tiempo justo para realizar este experimento y no se podía alargar más de lo pactado, ya que requerían realizar otras actividades personales en el transcurso del día.

Niveles de Correctitud

Grupo Sin Anotaciones GSA				
Participante	**NC1**	**NC2**	**NC3**	**NCT**
Javier A.	400	300	300	1000
Eduar T.	400	400	400	1200
Santiago G.	300	200	300	800
Grupo Con Anotaciones GCA				
Edwin M.	450	200	300	950
Santiago P.	200	400	300	900
Carlos M.	350	300	400	1050

Comprensión en documentación

Grupo Sin Anotaciones GSA						
Participante	**P1**	**P2**	**P3**	**P4**	**P5**	**C**
Javier A.	60	50	100	70	50	330
Eduar T.	30	40	70	90	50	280
Santiago G.	70	80	70	70	40	330
Grupo Con Anotaciones GCA						
Edwin M.	100	70	100	90	60	420
Santiago P.	100	100	100	85	90	475
Carlos M.	100	80	100	100	100	480

Resultados

Luego de tomar los tiempos de cada participante se reciben los sistemas con las modificaciones con el objetivo de evaluar el nivel de correctitud en la realización de las tareas. Para determinar los resultados del experimento se realiza una prueba de hipótesis con la cual se busca confirmar que el uso de las anotaciones de código con información del Rationale Arquitectónico mejora la

mantenibilidad de la Arquitectura en términos de eficiencia, efectividad y comprensión de la Arquitectura y su Rationale, cuando se realizan cambios con un impacto a nivel arquitectural. La prueba de hipótesis que se utiliza es una prueba t student, esta permite determinar si existe una significancia estadística entre dos variables, esto quiere decir que la presencia de la variable independiente afecta positivamente los resultados de las variables dependientes. Para esta prueba es necesario tener dos muestras iguales y se debe suponer igual varianza entre las muestras de cada grupo. En este trabajo los grupos se dividen en: Grupo Con Anotaciones (GCA) y Grupo Sin Anotaciones (GSA).

Cada una de las hipótesis considera una variable independiente en una escala nominal con dos niveles (con anotaciones, sin anotaciones) y una variable dependiente (efectividad, eficiencia o comprensión de la Arquitectura y su Rationale). Para confirmar las hipótesis se utiliza una prueba de t student suponiendo una distribución normal, varianzas y tamaños de muestras iguales, para dos medidas independientes. Debido a que nos interesa saber si los resultados están por encima de la distribución normal se hace uso de una prueba t student de una cola.

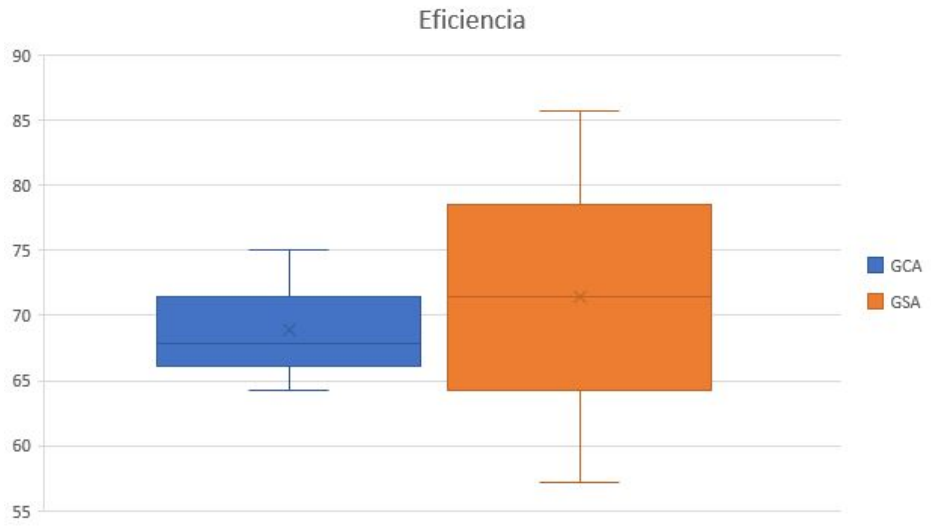
% Eficiencia		**% Efectividad**		**% Comprensión**	
GCA	**GSA**	**GCA**	**GSA**	**GCA**	**GSA**
67.86	71.43	63.33	66.67	84	66
64.29	85.71	60.00	80.00	95	56
75.00	57.14	70.00	53.33	96	66

Una vez calculadas las variables dependientes se hace el contraste de hipótesis mediante la prueba de t student con un nivel de significancia igual a 0.05.

Eficiencia

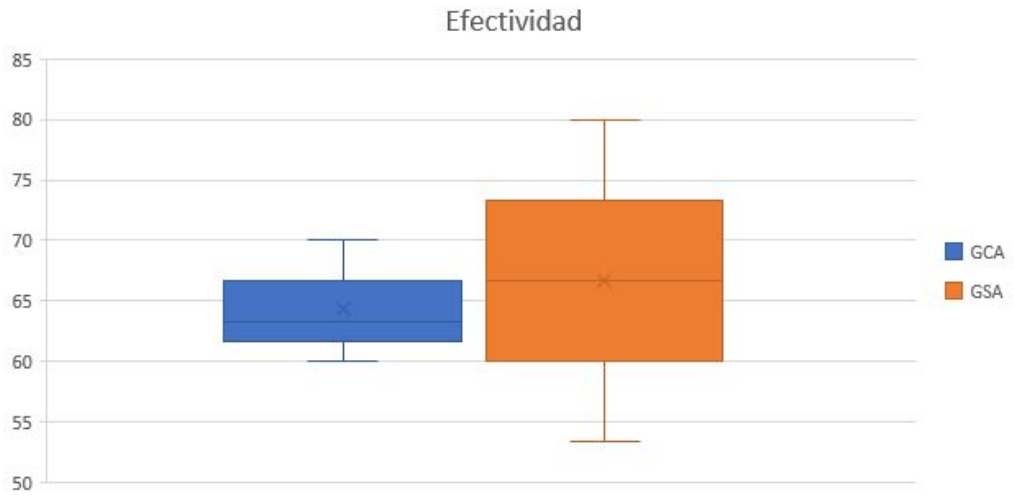
El porcentaje de eficiencia promedio para el grupo con anotaciones de código fue del 69.05% a diferencia del grupo sin anotaciones de código el cual tiene un porcentaje promedio de eficiencia del 71.43%, el cual podría indicar que el grupo sin anotaciones tuvo mejor eficiencia sin utilizar las anotaciones de código fuente. Sin embargo, cuando se realiza la prueba de t-student de una cola con un valor de significancia de 0.05 nos da como resultado un p-valor de 0.41, el cual es mucho mayor que 0.05, por lo tanto se acepta la hipótesis nula y se rechaza la hipótesis

alternativa. En la siguiente imagen, se puede observar que aunque el promedio de eficiencia del grupo con anotaciones de código es menor, los valores se mantienen en el rango del grupo sin anotaciones de código. Otra observación importante de la gráfica es que los valores se encuentran más agrupados en el grupo con anotaciones que en el grupo sin anotaciones, de esto se podría suponer que en un futuro los valores de eficiencia tienden a estar más centralizados si se utilizan las anotaciones de código con información del Rationale arquitectónico.



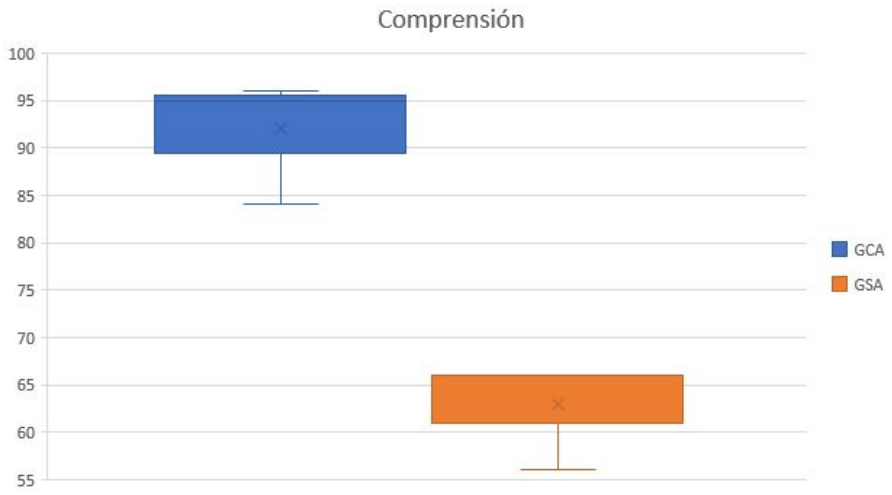
Efectividad

Los resultados muestran que el promedio de efectividad para el grupo con anotaciones de código es de 64.44 y el porcentaje promedio para el grupo sin anotaciones de código es del 66.67. De manera similar a la eficiencia, la efectividad parece afectada de manera negativa por el uso de anotaciones de código fuente con información del Rationale arquitectónico. Sin embargo, cuando se realiza la prueba de t-student con un valor de significancia de 0.05, el p-valor es de 0.41 mayor a 0.05, por lo que se debe rechazar la hipótesis alternativa y aceptar la hipótesis nula, esto significa que el uso de las anotaciones de código con información del Rationale Arquitectónico no tiene un impacto positivo ni negativo sobre la efectividad al realizar actividades de mantenimiento en tareas con un impacto arquitectural. En la siguiente imagen, se puede observar que, al igual que en la eficiencia, los datos están mejor agrupados para el grupo con anotaciones de código que para el grupo sin anotaciones, esto podría suponer que de alguna forma el uso de las anotaciones de código centraliza los valores para la efectividad, pero de ninguna forma tiene un impacto negativo o positivo sobre esta.



Comprensión

El promedio de comprensión para el grupo con las anotaciones de código es de 91.67%, a diferencia del grupo sin anotaciones el cual es de 62.67%. Hasta este momento se podría suponer que si hay impacto positivo en la comprensión de la arquitectura y su Rationale cuando se hace uso de las anotaciones con código fuente. Sin embargo, es necesario validar esta teoría realizando la prueba de t-student con un valor de significancia de 0.0.5. El cual da como resultado un p-valor de 0.002, el cual es mucho menor a 0.05, por lo tanto, aceptamos la hipótesis alternativa y rechazamos la hipótesis nula. Esto significa que el uso de anotaciones de código tiene una significación estadística sobre la comprensión de la arquitectura y su Rationale al realizar cambios con un impacto arquitectural en este experimento. Finalmente, podemos ver en la siguiente imagen que los valores para el grupo con anotaciones de código son mayores y están por fuera del rango de los resultados del grupo sin anotaciones de código, por lo cual, podemos afirmar que el aumento en la comprensión de la Arquitectura y su Rationale se debe al uso de anotaciones de código fuente con información estructurada del Rationale.



Materiales y recursos

En el experimento se entrega directamente la carpeta del código fuente: para los participantes con anotaciones de código con la librería .jar adjunta en las dependencias del proyecto, los reportes generados por la herramienta y el documento de arquitectura. A los participantes sin anotaciones de código se les entrega el código fuente únicamente con el documento de arquitectura.

Los reportes, el código fuente y el documento de arquitectura se puede descargar a través del siguiente repositorio: https://github.com/zahydo/experimento-controlado-ARAT.git. Al final de este documento se anexa el Documento de Arquitectura de Software con los diagramas y el resto de información sobre el código fuente experimental.

A ambos grupos se les anexa el siguiente formato en un documento pdf con las actividades que deben realizar:

Actividades por realizar:

1. Se debe crear un módulo de software que permita gestionar los pagos relacionados a un pedido. Este módulo se va a construir de manera iterativa e incremental. Sin embargo, se puede definir la estructura inicial y la organización de los componentes, de tal manera que se puedan implementar las funcionalidades más prioritarias.
   La primera implementación que se debe realizar consta de verificar si un pedido se puede pagar, realizando una validación con el saldo del método de pago. Esta implementación debe seguir con los siguientes parámetros de entrada y salida:
   a. Entrada: Número que identifica el método de pago idMetodoPago(Long), tenga en cuenta que MetodoPago no se define todavía en BD pero se puede trabajar con objetos quemados en memoria.
   b. Salida: Saldo del método de pago (Long)

2. Suponga que ha pasado tiempo y que se empieza a ver afectado el rendimiento del sistema debido a la gran cantidad de peticiones que realizan los usuarios sobre el sistema. Con un software de pruebas de rendimiento y un monitor de control de peticiones se logra determinar que el componente más afectado es el componente de pagos que ud creó.
   Además de lo anteriormente mencionado, este componente utiliza servicios externos para consultar datos de los medios de pagos, es por eso que es necesario extraerlo del nivel de la aplicación principal y pasarlo a un entorno independiente, donde tenga recursos propios y pueda responder de manera más eficiente a la cantidad de peticiones de los usuarios. Se quiere que este módulo sea totalmente independiente del resto de la aplicación, fácil de desplegar, de mantener y que permita el incremento de peticiones de los usuarios de manera controlada.

3. Documente los cambios realizados en las tareas anteriores según los artefactos que disponga para la documentación de la arquitectura. Si el tiempo no le alcanza para terminar con las tareas, por favor indique los cambios que planearía implementar lo más detallado posible (diferenciándolos con la palabra clave TODO:).

4. Finalmente responda las preguntas del formulario, que será entregado una vez indique que finalizó las modificaciones o el tiempo asignado para ellas.

Finalmente se adjunta el formulario de respuesta por cada uno de los participantes:

Formulario respuesta de Carlos Molano

D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?

Metodo para el punto 1. (com.uniceuca.modelo.ejbs.peticiones / JBPeticionFacade.java). Lo posicione en este carpeta porque es donde se situan los metodos que seran llamados desde las interfaces. Ademas no corresponde a la capa de datos, ni a la capa de vistas.

El cliente para consumir el servicio REST lo ubique en un nuevo paquete com.uniceuca.web.rest el cual es una nueva capa web con nuevas formas de consumo (metodos get) que no son locales (BD local).

E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios?. Justifique su respuesta.

Se busca (disponibilidad a nivel de muchos de usuarios que consumen los servicios, ahora la aplicacion tiene su servicio para verificacion de pagos en un servidor que le evita gastos de trabajo.

Nombre completo: Carlos Adrian Moreno Manrique
Profesión: Estudiante I. Sistemas
Cargo: Tecnico.
Experiencia:
Desarrollo con JAVA EE (BD Postgres), EntityBean JPA) y Angular.

Formulario de preguntas sobre la actividad. GDCA

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.

La organizacion permite que se refleje el codigo de acuerdo a tu funcion (Datos, Modelo y Presentacion) esto ayuda a que exista una mejor coherencia a nivel semantico y ubicar mas facilmente los cabios.

B. ¿Cual es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)?. Justifique su respuesta.

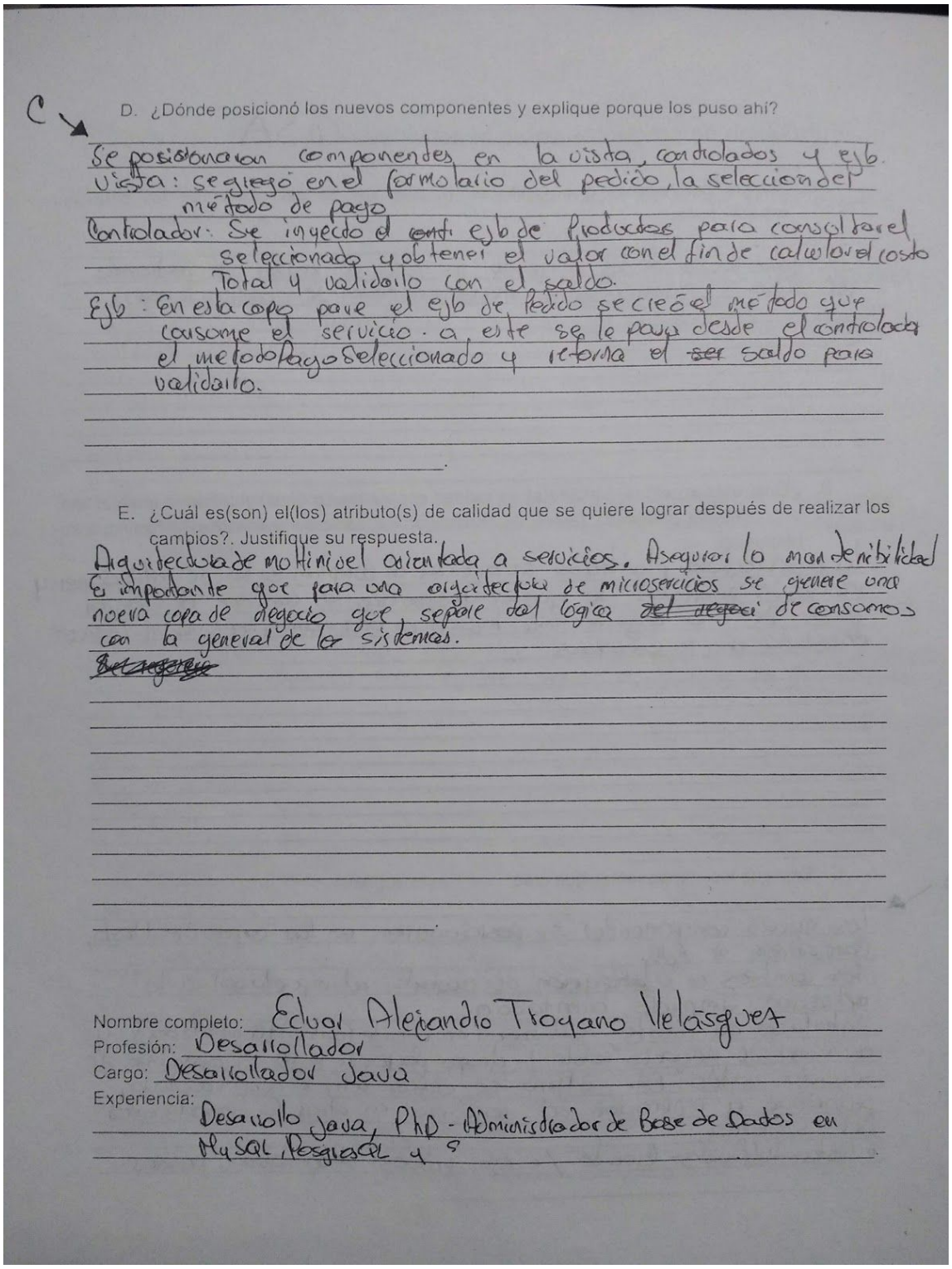
La arquitectura permitiva en un futuro modularizacion mediante microservicios. El atributo es especificamente el mantenimiento.

C. Explique los componentes que creó.

Un componente que verifica que el saldo de la tarjeta (metodo de pago) sea mayor al valor de un pedido, recibe como dtributo un LONG que corresponde al tipo de metodo de pago y retorna un LONG correspondiente al saldo de la tarjeta. (Se encuentra en com.unicauca.modelo.ejbs.pedidos/ TblPedidoUtilidadFacade.java).

El

Formulario de respuestas Eduar Troyano



C

D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?

Se posicionaron componentes en la vista, controlados y ejb.
Vista: se agregó en el formulario del pedido, la selección del método de pago
Controlador: Se inyectó el cont. ejb de Productos para consultar el seleccionado y obtener el valor con el fin de calcular el costo Total y validarlo con el saldo.
Ejb: En esta capa pare el ejb de Pedido se creó el método que consume el servicio. a este se le pasa desde el controlador el metodoPagoSeleccionado y retorna el ~~ser~~ saldo para validarlo.

E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios?. Justifique su respuesta.

Arquitectura de software orientada a servicios. Asegurar la mantenibilidad
Es importante que para una arquitectura de microservicios se genere una nueva capa de negocio que separe del lógica ~~del negocio~~ de consumo con la general de los sistemas.
~~Retroalimentación~~

Nombre completo: Eduar Alejandro Troyano Velásquez
Profesión: Desarrollador
Cargo: Desarrollador Java
Experiencia: Desarrollo Java, PhD - Administrador de Base de Datos en MySQL, Postgresql y s

Formulario de preguntas sobre la actividad. GSA

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.

La organización de los paquetes es dada a partir de patrones de diseño, seguridad y mantenimiento de aplicaciones.

B. ¿Cual es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)?. Justifique su respuesta.

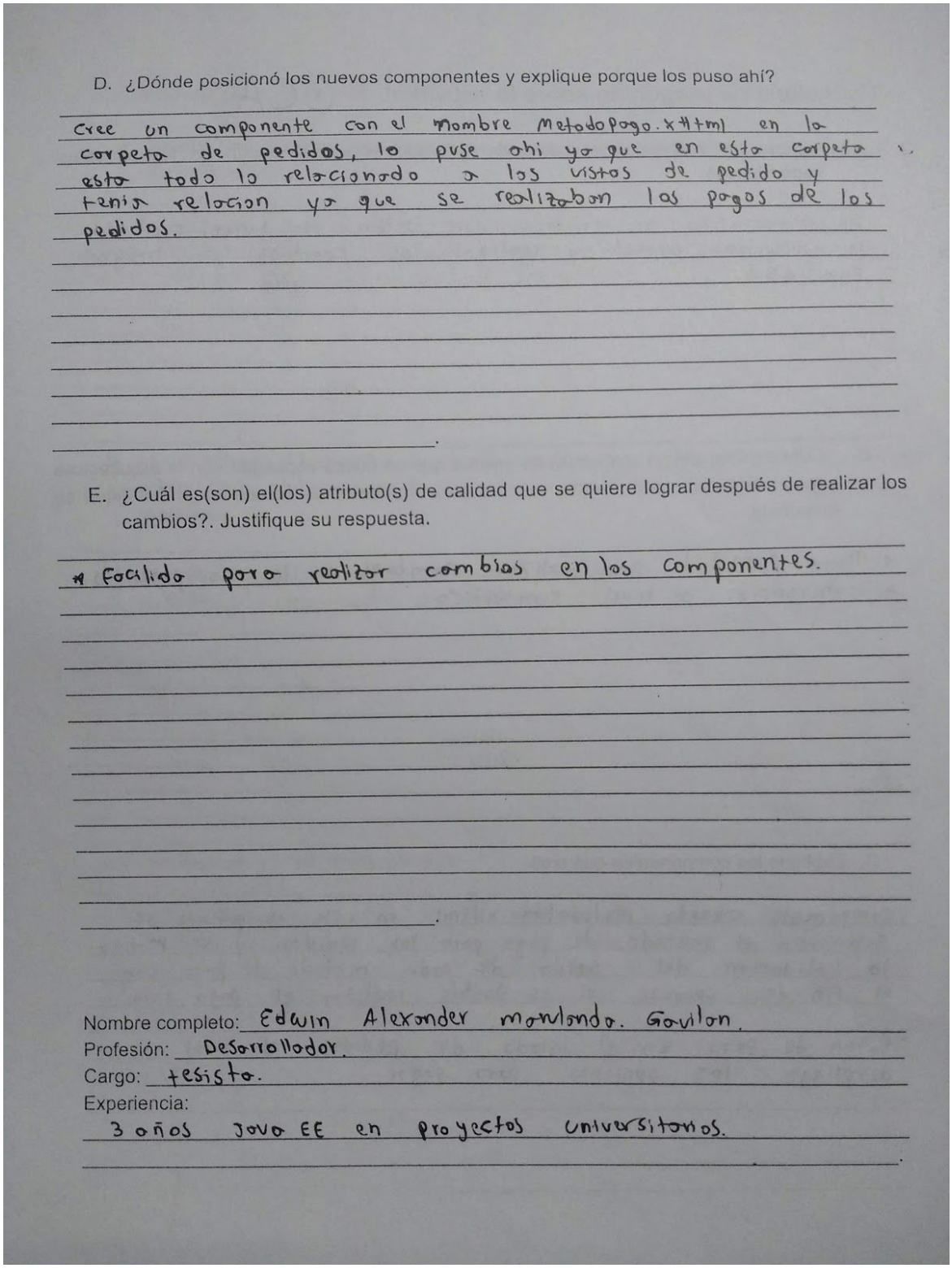
La arquitectura planteada se orienta al cumplimiento de replicaciones y recursos del proyecto. Lo que se busca con los cambios el lograr pasar de una arquitectura inicial a una arquitectura multicapa orientada a microservicios.

C. Explique los componentes que creó.

D →

Los nuevos componentes se posicionaron en las capas de Vista, Controlador y EJB.
Los cambios se establecieron de acuerdo al impacto sobre la aplicación. Impacto orientado a:
Interfaces del cliente validación de dicate (saldo) [Controlador] y a nivel de negocio sobre la capa pagorde EJB, el consumo de servicio Saldo. Este último se ubica allí x arquitectura, seguridad y replicación del consumo en otras funcionalidades del Sistema
• Vista - tblPedidos - Create / • ejbs. pedidos / • controlador. pedidos

Formulario de respuestas Edwin marulanda



D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?

Cree un componente con el nombre MetodoPago.xhtml en la carpeta de pedidos, lo puse ahi ya que en esta carpeta esta todo lo relacionado a los vistos de pedido y tenia relacion ya que se realizaban los pagos de los pedidos.

E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios?. Justifique su respuesta.

* Facilidad para realizar cambias en los componentes.

Nombre completo: Edwin Alexander manlanda. Gaviln.
Profesión: Desarrollador.
Cargo: tesista.
Experiencia:
3 años java EE en proyectos universitarios.

Formulario de preguntas sobre la actividad. GCA

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.

Se organiza de esta forma con el fin de tener facilidad de encontrar errores y realizar los cambios con mayor facilidad

B. ¿Cual es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)?. Justifique su respuesta.

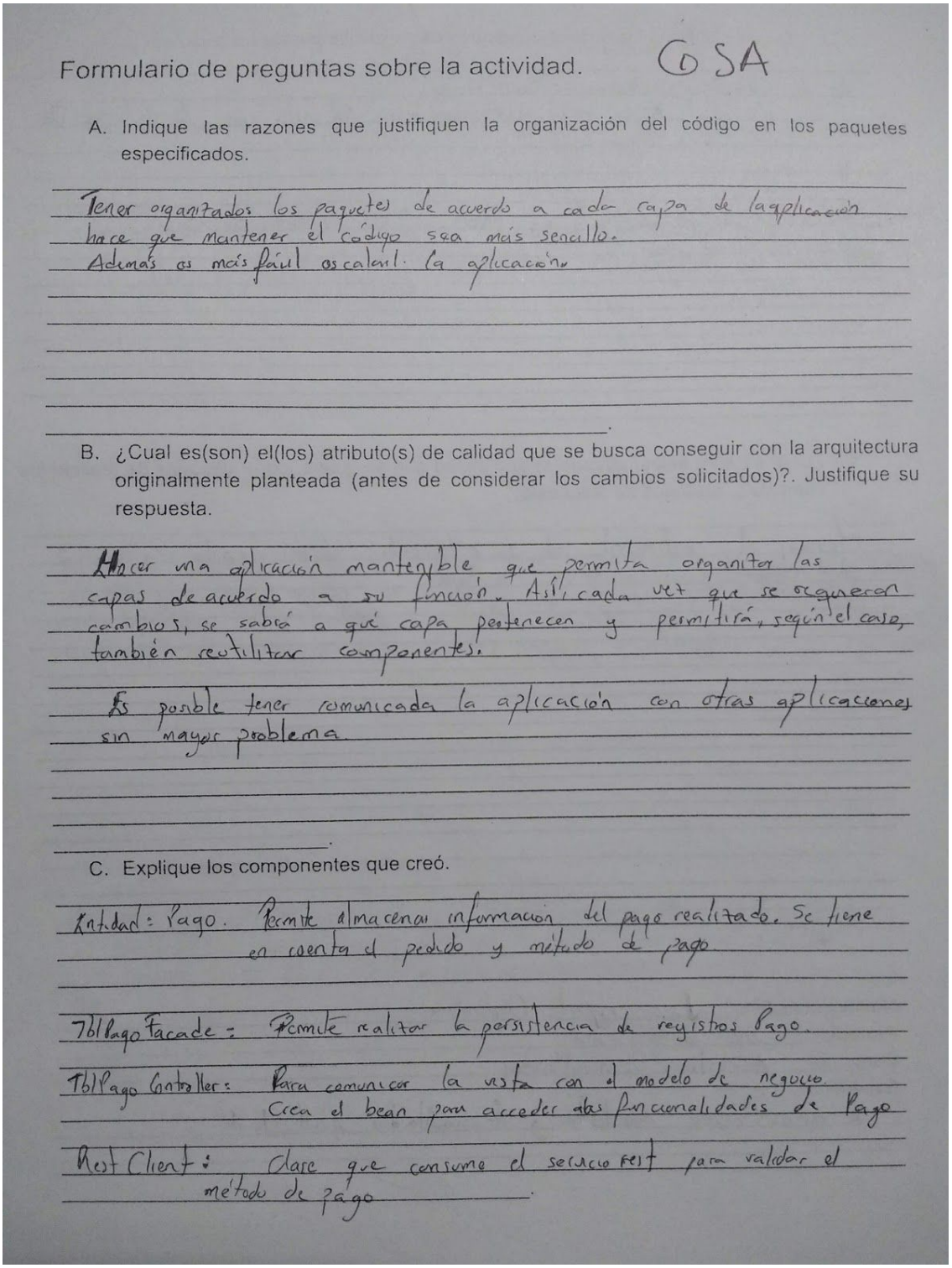
* mayor facilidad para realizar cambios en los componentes.
* coherencia a nivel semantico

C. Explique los componentes que creó.

Componente creado MetodoPago.XHtml, en este componente se selecciona el metodo de pago para los pedidos y se realiza la validación del saldo de cada metodo de pago con el fin de conocer si es posible realizar el pago o no.

boton de pago en el listado de pedidos el cual despliega las opciones para pagar.

Formulario de respuestas Javier Ágredo



Formulario de preguntas sobre la actividad. GOSA

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.

Tener organizados los paquetes de acuerdo a cada capa de la aplicación hace que mantener el código sea más sencillo. Además es más fácil escalar la aplicación.

B. ¿Cual es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)?. Justifique su respuesta.

Hacer una aplicación mantenible que permita organizar las capas de acuerdo a su función. Así, cada vez que se organicen cambios, se sabrá a qué capa pertenecen y permitirá, según el caso, también reutilizar componentes.

Es posible tener comunicada la aplicación con otras aplicaciones sin mayor problema.

C. Explique los componentes que creó.

Entidad: Pago. Permite almacenar información del pago realizado. Se tiene en cuenta el pedido y método de pago

TblPagoFacade: Permite realizar la persistencia de registros Pago.

TblPagoController: Para comunicar la vista con el modelo de negocio. Crea el bean para acceder a las funcionalidades de Pago

RestClient: Clase que consume el servicio Rest para validar el método de pago

D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?

En la capa de presentación - controlador.
Es la capa que comunica la vista con la lógica de negocios.

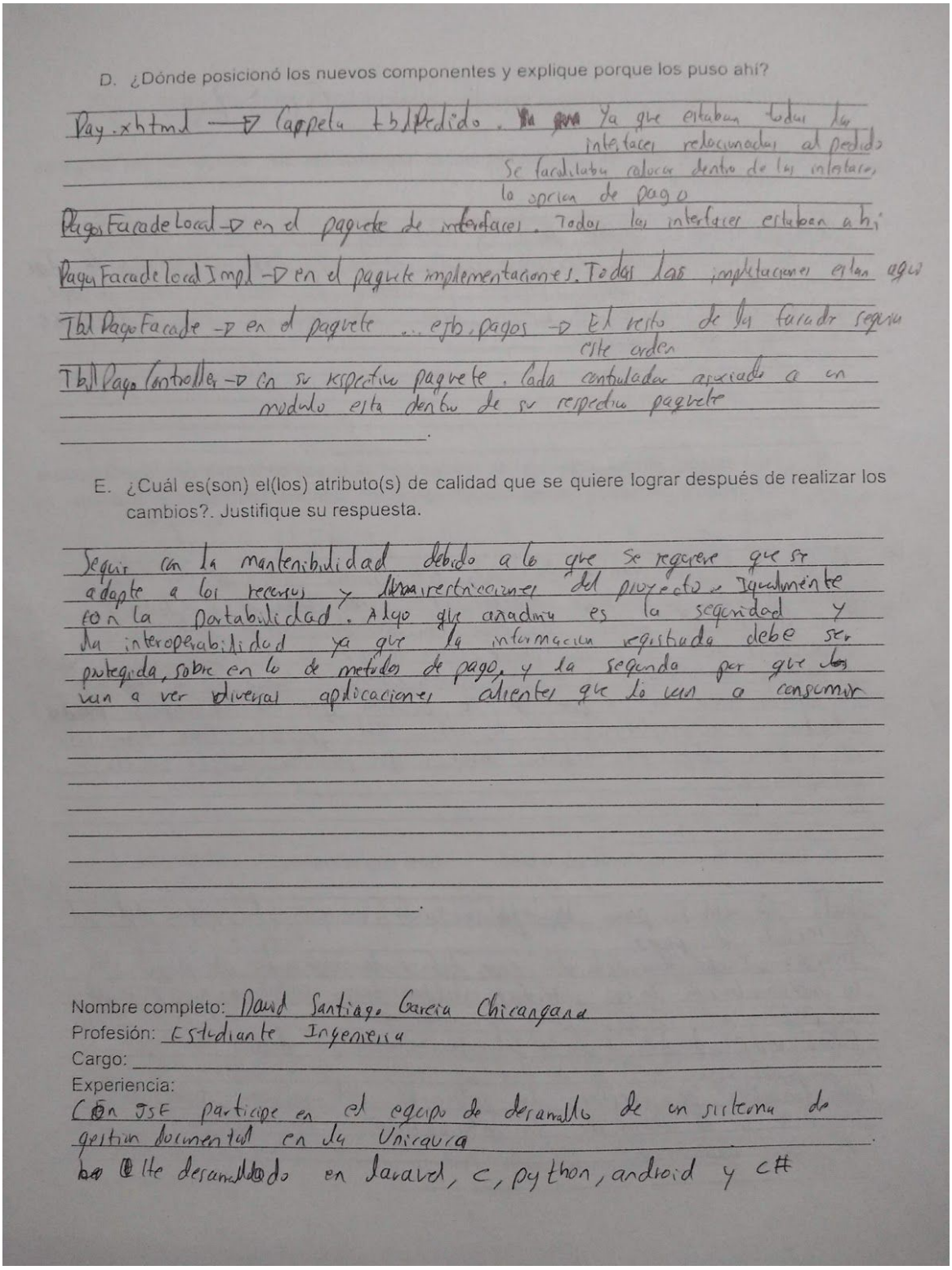
Implementación de interfaz en Modelo. Que permita gestionar la lógica de negocios.

E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios?. Justifique su respuesta.

Mejorar el rendimiento de la aplicación disminuyendo la carga al delegar procesamiento a otro servicio.

Nombre completo: Javier Alfredo Ordoñez
Profesión: Ing. de Sistemas
Cargo: Analista / Desarrollador
Experiencia:
2 años como analista y desarrollador full stack

Formulario de respuestas Santiago Garcia



D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?

Pay.xhtml → Cappeta tblPedido. Ya que estaban todas las interfaces relacionadas al pedido. Se facilitaba colocar dentro de las interfaces la opcion de pago

PagosFacadeLocal → en el paquete de interfaces. Todas las interfaces estaban ahi

PagoFacadeLocalImpl → en el paquete implementaciones. Todas las implementaciones estan aqui

TblPagoFacade → en el paquete ...ejb.pagos → El resto de las faradr seguia este orden

TblPagoControlle → en su respectivo paquete. Cada controlador asociado a un modulo esta dentro de su respectivo paquete

E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios?. Justifique su respuesta.

Seguir con la mantenibilidad debido a lo que se requiere que se adapte a los recursos y restricciones del proyecto. Igualmente con la portabilidad. Algo que añadiria es la seguridad y la interoperabilidad ya que la información registrada debe ser protegida, sobre en lo de metodos de pago, y la segunda por que van a ver diversas aplicaciones clientes que lo van a consumir

Nombre completo: David Santiago Garcia Chicangana
Profesión: Estudiante Ingenieria
Cargo:
Experiencia:
Con JSF participe en el equipo de desarrollo de un sistema de gestion documental en la Unicauca
He desarrollado en laravel, c, python, android y c#

Formulario de preguntas sobre la actividad. CDSA

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.

El diagrama muestra el estilo y la razon del organizacion de los componentes. Lo hacen para estructurar el sistema de manera sea portable y mantenible. Ademas, el sistema debe adaptarse a las restricciones y recursos del proyecto.

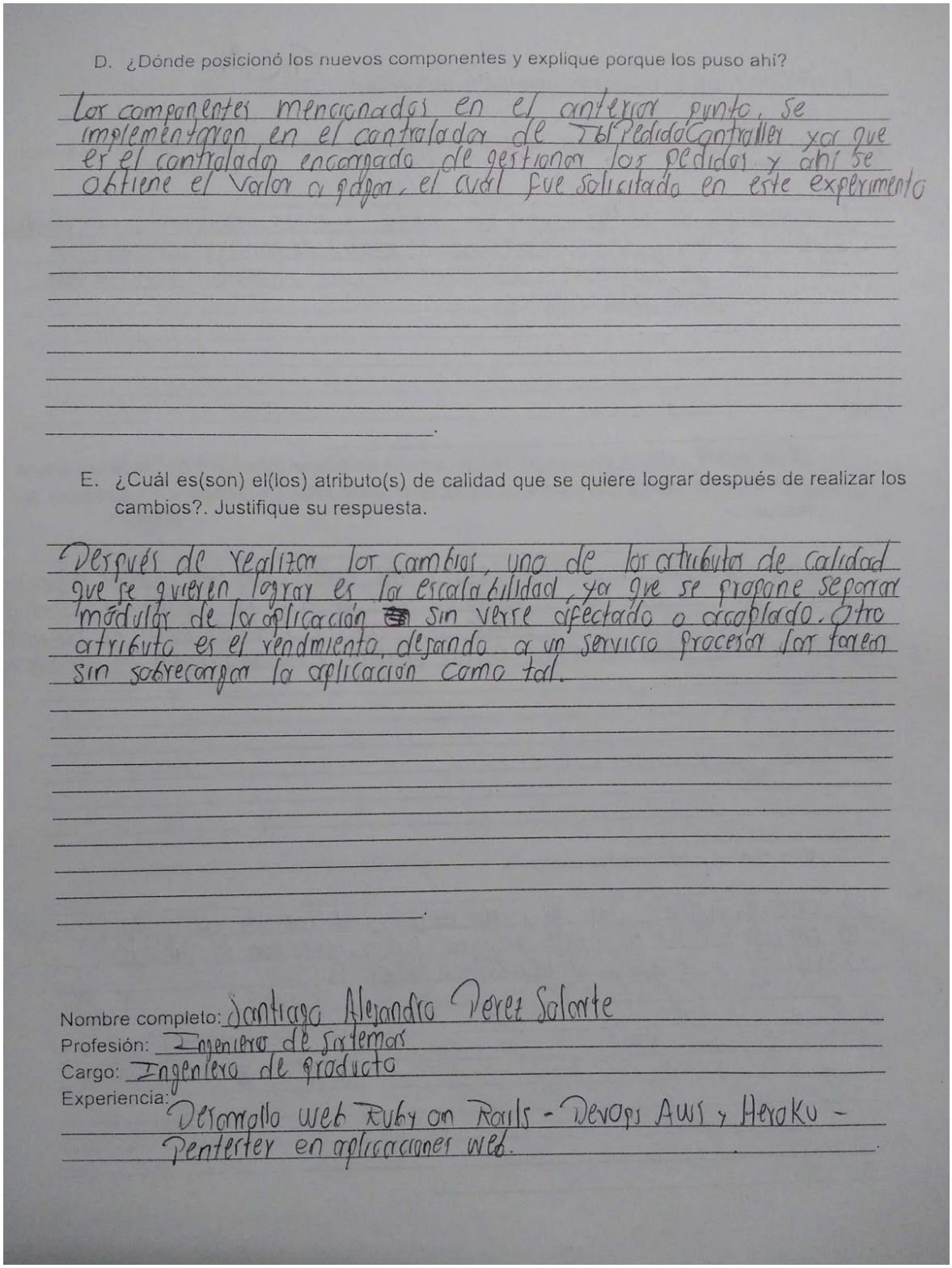
B. ¿Cual es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)?. Justifique su respuesta.

Portabilidad y mantenibilidad. El primero pedina ser por la funcionalidad que pretende realizar el sistema (~~toma de~~ pedidos). Para facilitar el acceso a esta funcionalidad desde diferentes dispositivos como los smartphone. Otra razon es la de exponer los servicios de manera transparante a las applicaciones clientes q' hagan uso de ellos. La segunda seria por lo que se refiere que el sistema se ~~debe~~ adaptar a las restricciones y los recursos del proyecto. Para lograr esto el sistema debe ~~ser~~ facilitar cambios que permitan cumplir con esta adaptación

C. Explique los componentes que creó.

Vista Pago.xhtml para desplegar una interfaz para la captura del id del metodo de pago
Interfaz Pagos FacadeLocal para definir los metodos de pago
la implementacion de esa interfaz donde tengo el objeto facade de pago
PagoFacade donde hago lo del manejo de los datos
Pago Controller para gestionar las peticiones del usuario a traves de la interfaz y el retorno de la información

Formulario de respuestas Santiago Perez



D. ¿Dónde posicionó los nuevos componentes y explique porque los puso ahí?

Los componentes mencionados en el anterior punto, se implementaron en el controlador de TblPedidoController ya que es el controlador encargado de gestionar los pedidos y ahí se obtiene el valor a pagar, el cual fue solicitado en este experimento.

E. ¿Cuál es(son) el(los) atributo(s) de calidad que se quiere lograr después de realizar los cambios?. Justifique su respuesta.

Después de realizar los cambios, uno de los atributos de calidad que se quieren lograr es la escalabilidad, ya que se propone separar módulos de la aplicación sin verse afectado o acoplado. Otro atributo es el rendimiento, dejando a un servicio procesar las tareas sin sobrecargar la aplicación como tal.

Nombre completo: Santiago Alejandro Perez Solarte
Profesión: Ingeniero de Sistemas
Cargo: Ingeniero de producto
Experiencia: Desarrollo web Ruby on Rails - Devops AWS y HeroKu - Pentester en aplicaciones web.

Formulario de preguntas sobre la actividad. GCA

A. Indique las razones que justifiquen la organización del código en los paquetes especificados.

la organización del código separada por las 3 capas Modelo - Vista - Controlador brindan un mejor entendimiento al momento de realizar cambios. Por otro lado los componentes se organizan de acuerdo a cada responsabilidad del negocio.

B. ¿Cual es(son) el(los) atributo(s) de calidad que se busca conseguir con la arquitectura originalmente planteada (antes de considerar los cambios solicitados)?. Justifique su respuesta.

Se busca conseguir mantenimiento esto con el fin de que al momento de realizar un cambio en el código del proyecto no se vea comprometido con problemas de coherencia semántica. Por otro lado se busca el reusamiento desacoplando modulos para evitar cargas exageradas en la aplicación.

C. Explique los componentes que creó.

Se creó el cliente rest el cual recibía el método de pago y la cantidad que el cliente deseaba pagar. esto con el fin de verificar si se podría realizar/efectuar el pago.

Universidad del Cauca

Documento de Arquitectura de Software

shopping

Santiago Hyun Dorado
19-08-2018

Historial de cambios

Fecha	Versión	Descripción	Autor
19/08/2018	1.0		Santiago Hyun Dorado

Tabla de contenido

Tabla de Figuras

1. Introducción

Este documento tiene como finalidad explicar en profundidad la organización fundamental de los componentes que dan origen a la funcionalidad requerida por los interesados en la construcción del software. En esta sección se define el propósito de este documento, el alcance, algunas definiciones, abreviaturas y acrónimos, y finalmente se muestra el resumen de las secciones posteriores.

1.1. Propósito

La organización fundamental de los componentes de software es una de las primeras tareas que se deben realizar en el desarrollo de software junto con el análisis del negocio. Un buen análisis provee buenas bases para establecer un diseño que se adecúe de manera eficiente a las necesidades del negocio, es por eso que es de gran importancia entender el modelo de negocio y definir cuáles son los puntos en donde se requiere apoyo a nivel arquitectural para complementar la funcionalidad a nivel de calidad del producto. El propósito principal de este documento es preservar las decisiones realizadas a nivel de diseño relacionadas con la arquitectura del software, con el objetivo de tener una base de conocimiento que permita entender o recordar con más agilidad la organización de los componentes y las responsabilidades encargadas de representar la funcionalidad del software.

1.2. Alcance

La definición de arquitectura de software a nivel general cuenta con un alto grado de abstracción, esta se define como la organización fundamental de los componentes que en conjunto forman un sistema. Debido a esto, la arquitectura de software se puede representar de diferentes maneras dependiendo de la perspectiva del observador. Desde el origen del desarrollo de software se pensaron y se estandarizaron algunos meta-modelos que permiten abstraer y modelar la arquitectura a través de diagramas, siendo uno de los lenguajes de modelado más conocidos y utilizados UML. La información contenida en este documento permite entender la estructura del sistema desde diferentes perspectivas, además, permite también describir explícitamente las razones de las decisiones arquitecturalmente importantes implementadas en el proceso de desarrollo. La representación de la arquitectura se expresa en el modelo 4+1 vistas ya que este permite observar la arquitectura desde diferentes perspectivas y para diferentes audiencias, además para cada vista de este modelo existen diagramas bien definidos que facilitan la descripción de cada una de ellas.

Este documento permanece en constante actualización debido a que el sistema se encuentra actualmente en desarrollo y no se ha generado ninguna versión para producción. Sin embargo, este artefacto brinda las bases para establecer la organización de los componentes y definir las decisiones de diseño que se toman respecto a las necesidades a nivel funcional y no funcional.

1.3. Definiciones, acrónimos y abreviaturas

SAD: Software Architecture Document

GUI (Graphic User Interface): Interfaz Gráfica del Usuario

POO: Programación Orientada a Objetos

ISO (International Organization for Standardization): Organización Internacional de Normalización.

ARAT (Architectural Rationale Annotations Tool): Es una herramienta software basada en anotaciones de código fuente para documentar las razones arquitecturales de un sistema desarrollado en Java.

Java: Es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems.

Stakeholder: Según N. Rozanski et al [2] un stakeholder en una arquitectura de software "es una persona, grupo o entidad con intereses o preocupaciones sobre la realización de la arquitectura".

MVC (Model View Controller): Es uno de los patrones de diseño arquitectural más utilizado en el desarrollo web, empezó como un Framework desarrollado por Trygve Reenskaug alrededor de 1970 [3].

OMG (Object Management Grpup): Es un consorcio internacional de estándares tecnológicos sin fines de lucro, de membresía abierta [4].

UML (Unified Modeling Language): El lenguaje de modelado unificado (UML) es un lenguaje gráfico creado por el OMG para visualizar, especificar, construir y documentar los artefactos de un sistema de software [5].

1.4. Referencias

DAS: https://www.csun.edu/engineering-computer-science

Java: https://www.java.com/es/download/faq/whatis_java.xml

Stakeholder: https://www.viewpoints-and-perspectives.info/home/stakeholders/

MVC: https://msdn.microsoft.com/en-us/library/ff649643.aspx

ISO: https://www.iso.org/home.html

OMG: https://www.omg.org/

UML: http://www.uml.org/

1.5. Resumen

Este documento se compone de las siguientes secciones:

- Sección 1: Provee una introducción relacionada con la arquitectura de software del Juego N en línea.
- Sección 2: Describe los objetivos a nivel general de la Arquitectura del sistema.
- Sección 3: Describe la representación de la arquitectura en diferentes vistas.
- Sección 4: Desglosa la representación de la arquitectura en diagramas para cada vista.
- Sección 5: Referencias bibliográficas usadas en la creación de este documento.

2. Objetivos de la Arquitectura

La arquitectura de este sistema va dirigida a cumplir con requerimientos funcionales que den cumplimiento a las necesidades del negocio expresadas por los interesados del proyecto. Uno de los principales objetivos de la arquitectura es conseguir una aplicación que se adapte a las restricciones y los recursos del proyecto, en este caso es de carácter obligatorio utilizar Java como lenguaje de desarrollo tanto a nivel de Back-End como a nivel de Front-End debido a que la zona en la que se desarrolla el proyecto la oferta de desarrolladores con conocimientos en este lenguaje de desarrollo y sus herramientas relacionadas es alta. Debido a lo anterior la arquitectura planteada se basa en tecnologías como EJB's, JSF, Primefaces y JPA. Los EJB's representan la capa de negocio, Java Server Faces junto con PrimeFaces permiten la interacción con el usuario a través de la capa de vista y finalmente con JPA se tiene una última capa a nivel de acceso a datos. Uno de los objetivos principales de la estructura actual de los componentes software es facilitar la transición de una arquitectura en un nivel, a una arquitectura multinivel orientada a microservicios. Con el objetivo de conseguir que el sistema pueda brindar portabilidad hacia futuro, exponiendo los servicios de manera transparente a las aplicaciones clientes que hagan uso de ellos.

3. Representación de la Arquitectura

3.1. Vistas arquitecturales

La representación del sistema se puede ver desde diferentes puntos de vista, en el Modelo 4+1 Vistas propuesto por Kruchten p. et al [6] se definen 5 perspectivas que permiten extender el entendimiento de la arquitectura a las diferentes personas involucradas en el desarrollo de un sistema. En la Figura 1 se puede observar la organización de las vistas y su relación.

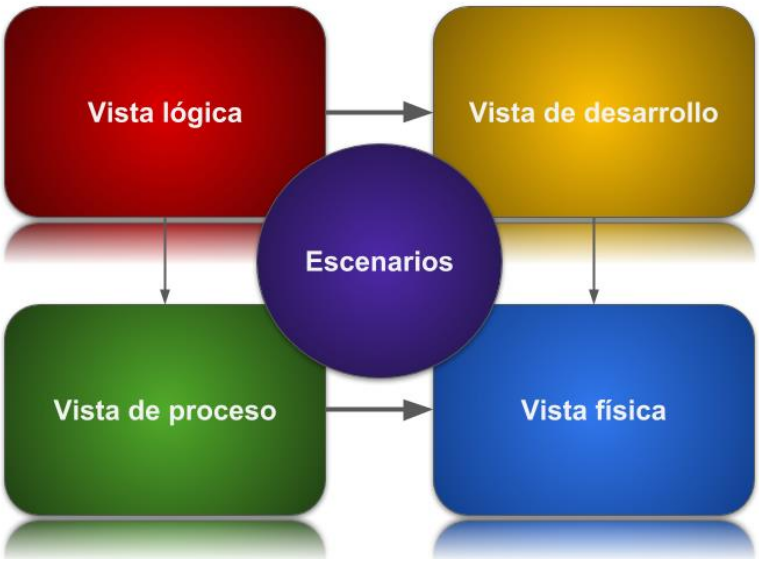


Figura 1 Modelo 4+1 vistas

3.1.1. Vista de escenarios

Conocida también como Vista de Casos de Uso, brinda información a nivel gráfico sobre los requerimientos funcionales de un sistema. Esta vista permite establecer las acciones y los roles que existen en el desarrollo de un sistema. Cada rol definido en el sistema debe tener una cantidad de casos de uso correspondientes a los requerimientos establecidos.

Audiencia: Stakeholders

Artefactos relacionados: Diagrama de Casos de Uso

3.1.2. Vista lógica

La Vista Lógica describe un sistema mediante el paradigma orientado a objetos, esta vista muestra las abstracciones diseñadas para estructurar el sistema junto con sus atributos y las operaciones que definen su comportamiento. Además, muestra cómo se relacionan cada una de ellas para formar partes más grandes de un sistema complejo.

Audiencia: Diseñadores

Artefactos relacionados: Diagrama de clases

3.1.3. Vista de desarrollo

Conocida también como Vista de Componentes, representa la arquitectura en términos de componentes software generalmente constituidos por una o más clases, estos componentes se relacionan entre sí para definir el comportamiento esperado por los usuarios finales. Estos componentes suelen ser software, sin embargo, se pueden dar casos donde los algunos de los componentes sean físicos o de hardware.

Audiencia: Desarrolladores

Artefactos relacionados: Diagrama de componentes, Diagramas de paquetes

3.1.4. Vista de proceso

Esta vista permite observar las tareas individuales que realizan las abstracciones software para cumplir con un determinado proceso. Los requisitos del usuario se pueden comprender como tareas que un sistema debe cumplir, esta vista permite desglosar esas actividades en partes más pequeñas con el objetivo de determinar los procesos de manera gráfica y ordenada. La complejidad de los procesos determina el nivel de abstracción adecuado para realizar el diseño del modelo.

Audiencia: Integradores, Desarrolladores

Artefactos relacionados: Diagrama de actividades, Diagrama de secuencia

3.1.5. Vista física

Es importante conocer la distribución de los componentes físicos que constituyen un sistema, la Vista Física permite observar la topología y las comunicaciones que se establecen entre los nodos físicos que conforman la arquitectura hardware del sistema.

Audiencia: Líderes de despliegue

Artefactos relacionados: Diagrama de despliegue

En la sección Descomposición de la arquitectura se muestran los diagramas utilizados en el Juego N en línea para cada tipo de vista.

3.2. Patrones de diseño arquitectónicos

El patrón de diseño arquitectural implementado en el desarrollo del Juego N en línea es el patrón *Modelo Vista Controlador MVC*. Por medio de este patrón podemos realizar una separación de la de interacción del usuario con la Interfaz Gráfica en tres diferentes componentes. En la Figura 2 se puede observar la interacción entre los componentes del patrón.

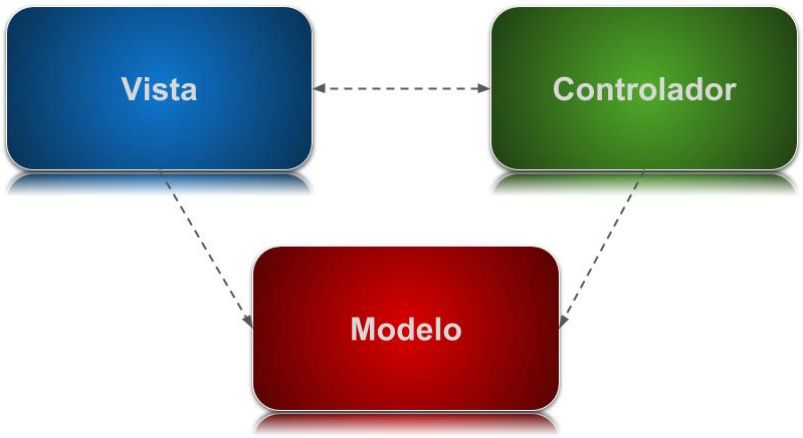


Figura 2 Patrón MVC

Modelo: Representa la información sobre el dominio del negocio, los datos y el comportamiento del sistema.

Vista: Es un componente que representa la *Interfaz Gráfica del Usuario GUI* con los datos del modelo, este componente captura las acciones y los datos y acude al controlador adecuado para procesarlos.

Controlador: Captura los datos ingresados por el usuario, manipula el modelo y actualiza la vista de forma correcta.

En la sección *Vista lógica* se muestra con más detalle las clases y los paquetes que componen este patrón de diseño arquitectural.

3.3. Estilo arquitectural

Los estilos arquitecturales son soluciones para resolver un problema relacionado con requerimientos NO funcionales o atributos de calidad, cada estilo arquitectural representa la experiencia de los diseñadores en un dominio de un problema específico. Las necesidades de calidad cambian de un sistema a otro por lo tanto los estilos arquitecturales son muy diversos y es muy difícil que contemplen todos los atributos de calidad requeridos [7]. Para el desarrollo del sistema Shopping los requisitos NO funcionales de mayor prioridad son la portabilidad y la mantenibilidad. La portabilidad es la característica de un sistema de ejecutar las instrucciones en diferentes dispositivos y la mantenibilidad según la Norma ISO 25000 [8], es la facilidad para efectuar cambios, reutilizar componentes o agregar funcionalidad al sistema.

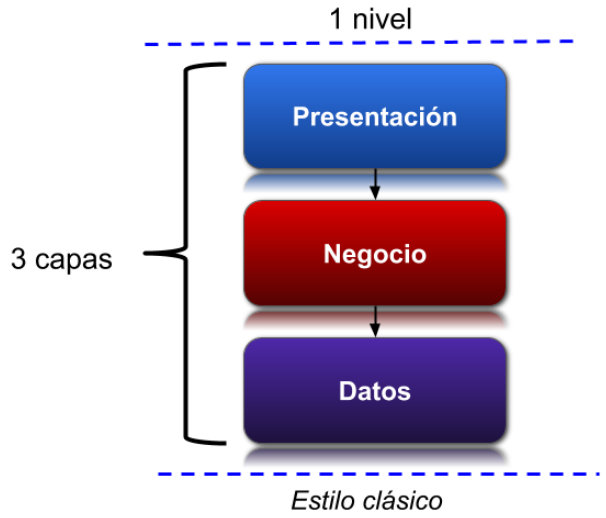


Figura 3 Estilo arquitectural clásico

En la Figura 3 se puede observar que el Juego N en línea sigue el estilo clásico de un nivel y tres capas propuesto por G. Florijn et al [7].

Capa de Presentación: Se encarga de controlar el flujo de trabajo entre el usuario y la GUI, también es responsable por mostrar la información del modelo. En el Juego N en línea esta capa contiene los componentes *Vista* y *Controlador* del *Patrón MVC.*

Capa de Negocio: Define los atributos y el comportamiento de las abstracciones de software en términos del contexto del negocio. En esta capa se puede encontrar las entidades que representan el *Modelo* del *Patrón MVC*.

Capa de datos: Esta capa es la encargada de acceder a los datos y establecer la persistencia de la información en el sistema.

En la Figura 4 se puede observar la organización de las capas y los componentes del *MVC.*

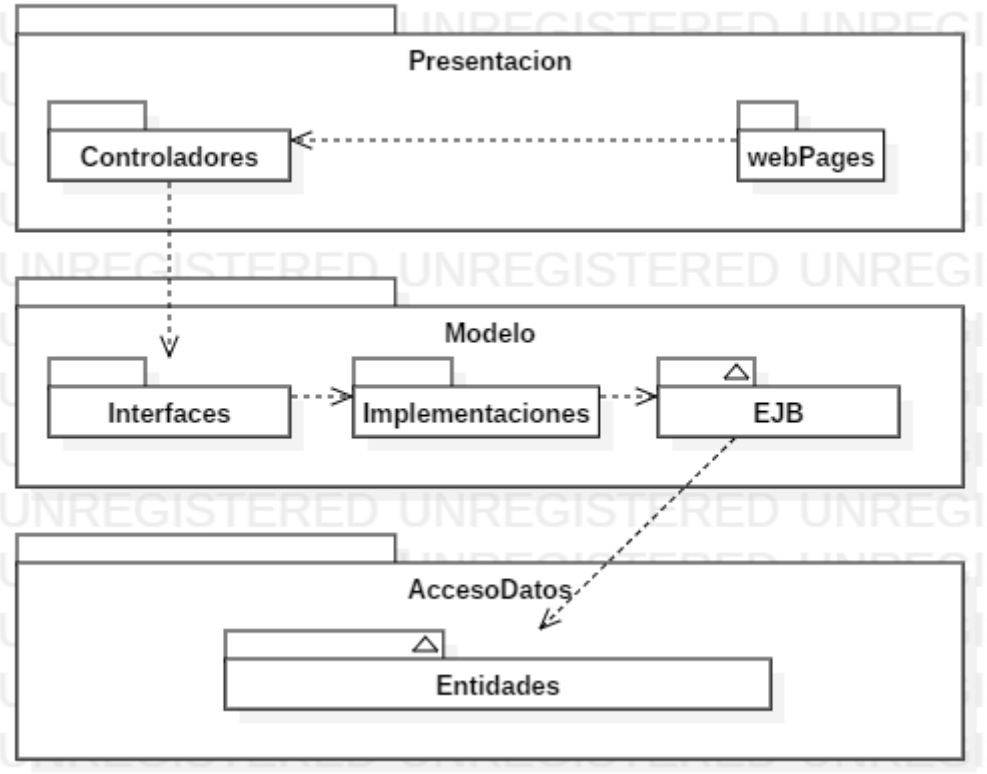


Figura 4 Organización Capas y componentes MVC

4. Descomposición de la arquitectura

La representación de la arquitectura se puede definir desde diferentes vistas adecuadas para una determinada audiencia. Estas vistas están relacionadas con uno o más diagramas UML que detallan con más información la arquitectura del sistema.

4.1. Vista de escenarios (casos de uso)

Las funcionalidades a nivel general se describen en la Figura 5. El jugador en el contexto del juego real es la persona que utiliza un tablero con n filas por m columnas y una cantidad x de fichas con el objetivo de realizar una secuencia de fichas continua del mismo color. Esta persona es la encargada de ubicar la ficha en una columna del tablero y soltarla, la evaluación del ganador se hace de manera visual, rectificando que haya una cantidad x de fichas contiguas del mismo color.

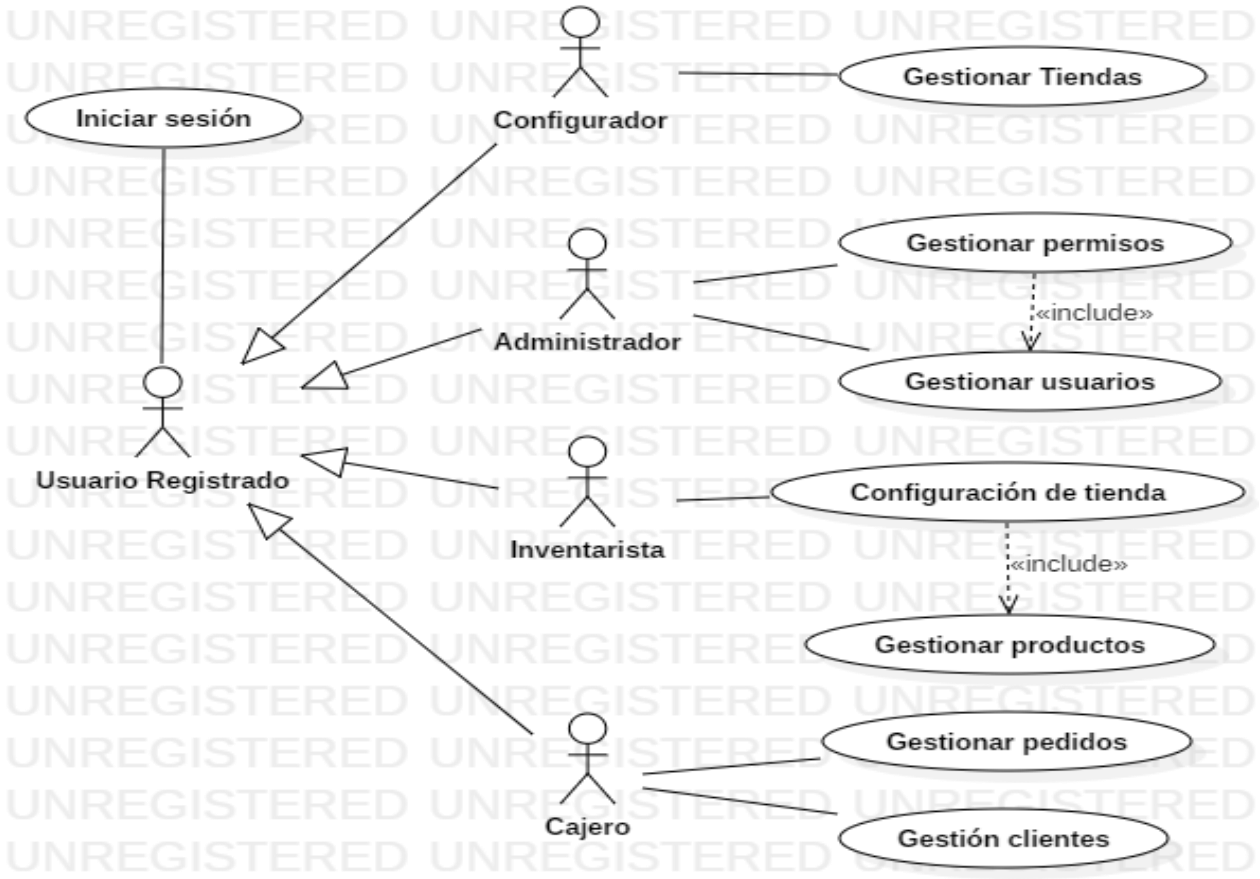


Figura 5 Diagrama de Casos de uso

ID	CU01
Nombre	Iniciar sesión
Actores	Usuario registrado
Sinopsis	Inicio de sesión en el aplicativo
Pre-condición	Tener usuario y contraseña activos en el sistema
Curso típico de eventos	1. Digitar nombre de usuario 2. Digitar contraseña 3. Ingreso al sistema
Cursos alternativos	En caso de que el sistema valide los datos ingresados por la persona y no encuentre ningún usuario con esas credenciales se debe notificar al usuario por medio de un mensaje.
Extensiones	Ninguno
Prioridad	Media

% Completado	100

ID	CU02
Nombre	Gestionar tiendas
Actores	Configurador
Sinopsis	Configurar la información de las diferentes tiendas(sucursales) de la empresa
Pre-condición	Debe haber iniciado sesión (CU01)
Curso típico de eventos	1. Después de que el configurador haya iniciado sesión se muestra la pantalla principal con el menú de opciones en la parte izquierda. 2. El configurador selecciona en el menú configurar tienda y se despliega una lista de tiendas asociadas al sistema 3. El configurador tiene las opciones de Crear, Editar, Eliminar o Ver la información de las tiendas.
Cursos alternativos	El configurador decide no realizar ningún cambio y cierra sesión mediante un botón ubicado en la parte superior derecha de la pantalla.
Extensiones	Crear tienda, Editar tienda, Buscar tienda.
Prioridad	Baja
% Completado	100

ID	CU03
Nombre	Gestionar Usuarios
Actores	Administrador
Sinopsis	Gestionar los usuarios que acceden al sistema, asignarles permisos y definir el rol respectivo para cada usuario.
Pre-condición	Se debe haber iniciado sesión (CU01)
Curso típico de eventos	1. Después de iniciar sesión el sistema muestra la pantalla principal, en la que se muestra el menú de opciones en la parte izquierda. 2. Usuario selecciona la opción 'Gestionar Usuarios' y se despliega una lista con las opciones de 'Gestionar roles', donde se definen los roles que hay en el sistema, la 'Gestión de permisos' donde se asignan los permisos a los cuáles cada usuario tiene acceso y finalmente la 'Gestión de usuarios' donde se Agregan, Modifican, Visualizan o Eliminan a los usuarios del sistema.
Cursos alternativos	
Extensiones	Gestionar Roles, Gestionar Permisos, Agregar, Editar, Buscar y Eliminar Usuarios
Prioridad	Baja
% Completado	70

ID	CU04
Nombre	Configurar tienda

Actores	Inventarista, Configurador
Sinopsis	El usuario relaciona los productos que tiene cada tienda
Pre-condición	Deben existir productos y tiendas configurados previamente
Curso típico de eventos	1. El sistema despliega la lista de Tiendas que existen en la base de datos 2. El usuario selecciona una de las tiendas y se despliega un diálogo dónde se muestra el nombre de la tienda y el producto respectivo que se quiere agregar 3. El sistema relaciona los productos a la tienda.
Cursos alternativos	El usuario no modifica relaciona productos a ninguna tienda y cierra sesión
Extensiones	
Prioridad	Baja
% Completado	60

ID	CU05
Nombre	Gestionar productos
Actores	Inventarista, Configurador
Sinopsis	El Inventarista/Configurador/Cocinero pueden gestionar las diferentes categorías de los productos así como también pueden relacionar los diferentes ingredientes que tiene un producto.
Pre-condición	Deben existir categorías e ingredientes previamente configurados
Curso típico de eventos	1. El usuario selecciona del menú la opción 'Gestionar productos' 2. El sistema despliega un submenú del cual se tienen las opciones 'Gestionar Categorías', 'Gestionar Ingredientes' y 'Gestionar productos'. 3. El usuario selecciona alguno de las anteriores opciones y el sistema muestra un listado de elementos según la opción seleccionada. 4. El usuario puede Crear, Buscar, Editar o eliminar Categorías, Ingredientes o Productos.
Cursos alternativos	El usuario decide no realizar ninguna opción y cierra sesión.
Extensiones	Gestionar Categorías, Gestionar Ingredientes
Prioridad	Baja
% Completado	60

ID	CU06
Nombre	Gestión de Clientes
Actores	Cajero
Sinopsis	El Cajero puede gestionar la información de los clientes habituales.
Pre-condición	Se debe haber iniciado el juego (CU01)
Curso típico de eventos	1. El cajero ingresa a registrar la información de los clientes 2. El sistema le muestra una lista de clientes 3. El usuario puede agregar, editar, buscar o eliminar clientes.

Cursos alternativos	
Extensiones	Agregar cliente, Editar cliente, Buscar cliente, Eliminar Cliente
Prioridad	Baja
% Completado	100

ID	CU07
Nombre	Gestión de Pedidos
Actores	Cajero
Sinopsis	El Cajero puede gestionar la información de los pedidos que se realizan en el sitio.
Pre-condición	Se debe haber iniciado el juego (CU01)
Curso típico de eventos	1. El sistema muestra una lista de pedidos al usuario. 2. El cajero puede agregar, editar, modificar o cancelar pedidos
Cursos alternativos	
Extensiones	Agregar , Editar, Buscar, Eliminar pedido
Prioridad	Alta
% Completado	70

4.2. Vista lógica (diseño)

En esta vista se muestra la estructura lógica del sistema en términos de abstracciones y entidades que dan forma al sistema. La composición de las clases se forma de los siguientes elementos:

- Interfaz: En la cual se definen los métodos relacionados con un componente
- Implementación: Clase que implementa los métodos definidos en la interfaz de algún componente
- Ejb: Entidad que manipula las entidades para cumplir con las reglas de negocio
- Entidad: Abstracción que representa los objetos reales en el dominio del negocio.

A continuación, se muestran las jerarquías de los componentes definidos hasta ahora, en cada interfaz se definen todas las acciones que se pueden realizar sobre un componente:

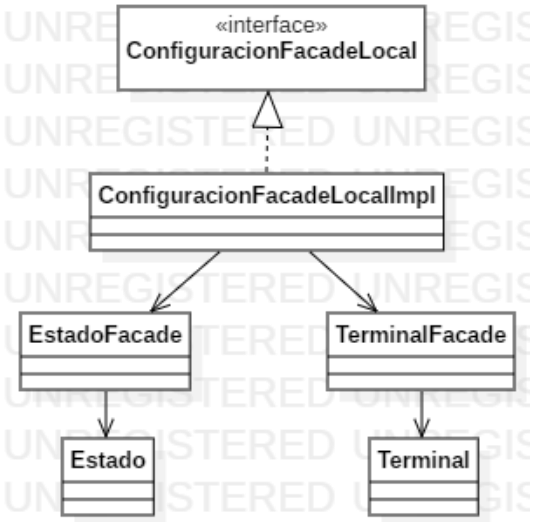


Figura 6 Diagrama de clases del componente Configuración

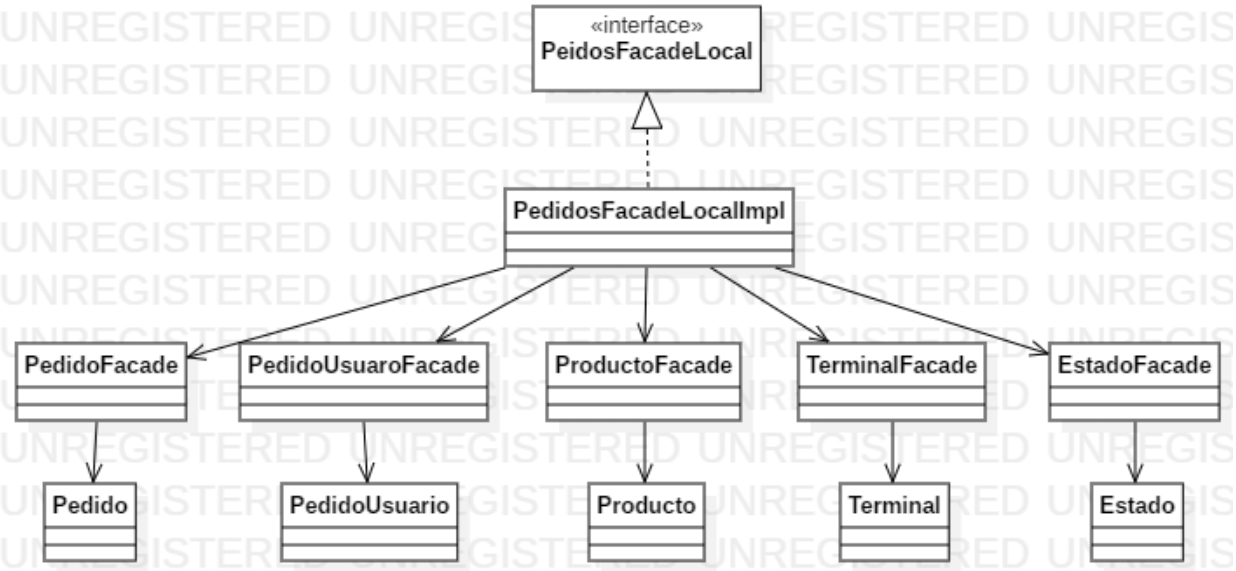


Figura 7 Diagrama de clases del componente Pedidos

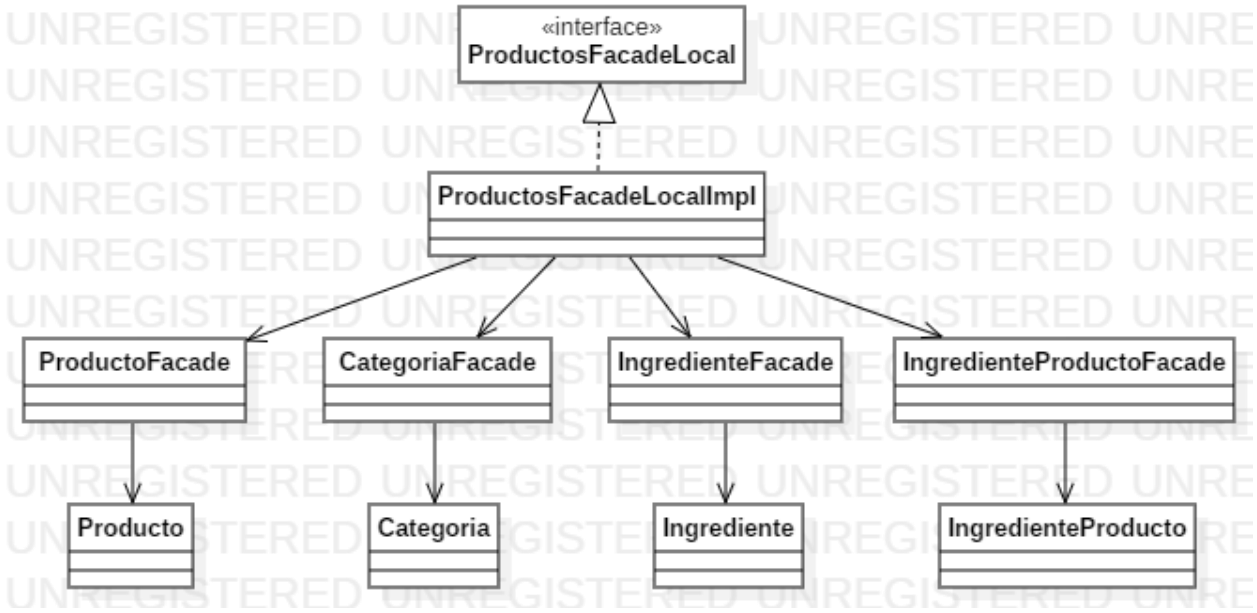


Figura 8 Diagrama de clases del componente Productos

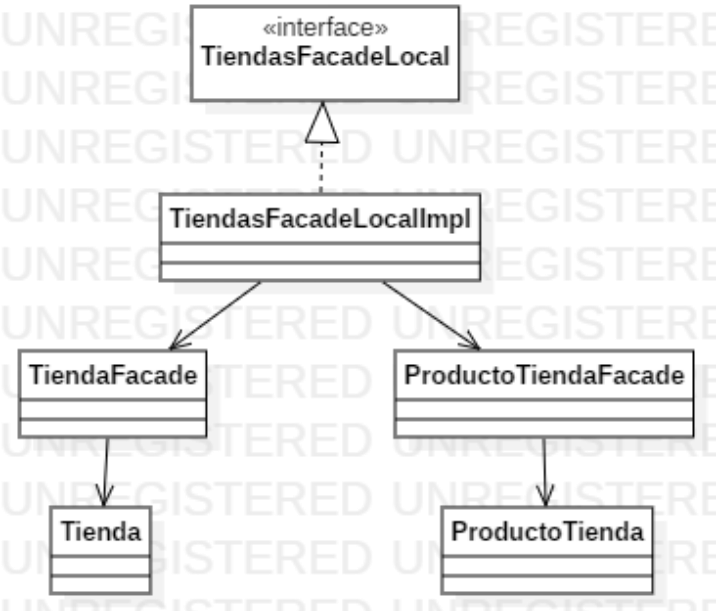


Figura 9 Diagrama de clases del componente Tiendas

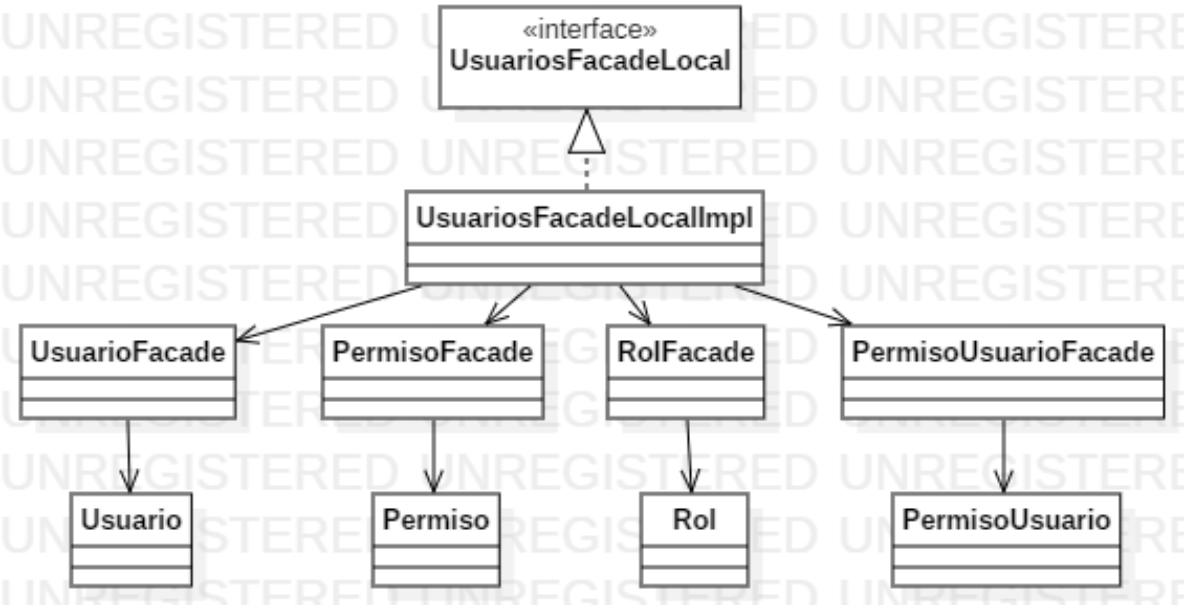


Figura 10 Diagrama de clases del componente Usuarios

4.3. Vista de proceso (actividades)

En esta vista se muestra la secuencia de actividades que son prioritarias en el sistema, en la Figura 11 se puede observar la interacción cuando un usuario quiere realizar un pedido.

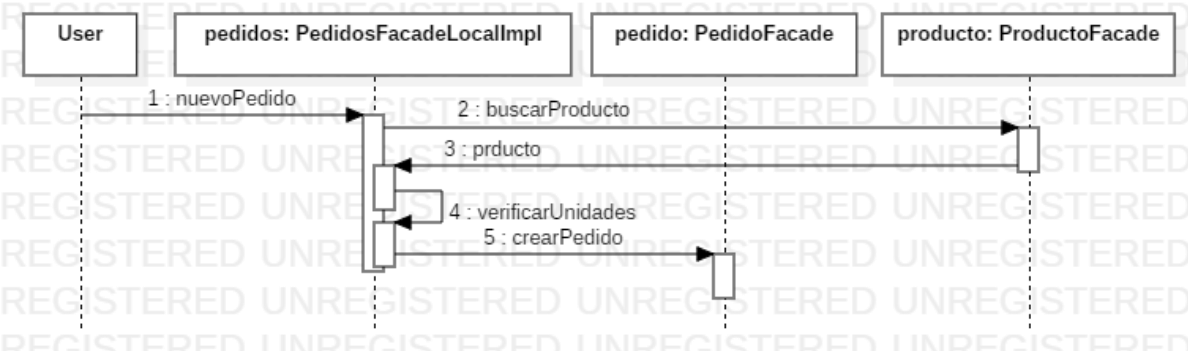


Figura 11 Diagrama de secuencia Realizar pedido

4.4. Vista de desarrollo (componentes)

En esta vista se pueden observar los 5 componentes principales que conforman el sistema, las interfaces por las cuáles se comunican los componentes y las relaciones que se configuran entre sí.

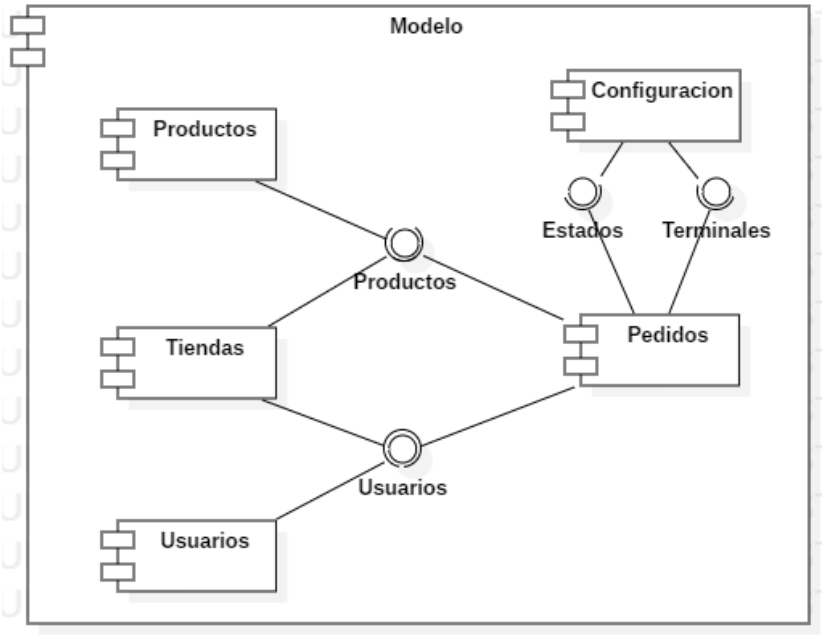


Figura 12 Diagrama de componentes

4.5. Vista física (despliegue)

El sistema Shopping se empaqueta en un archivo .war el cual es instalado en un Servidor Glassfish, este se conecta mediante jdbc con puerto 1521 a la base de datos instalada en un servidor temporal para asistir de repositorio de datos.

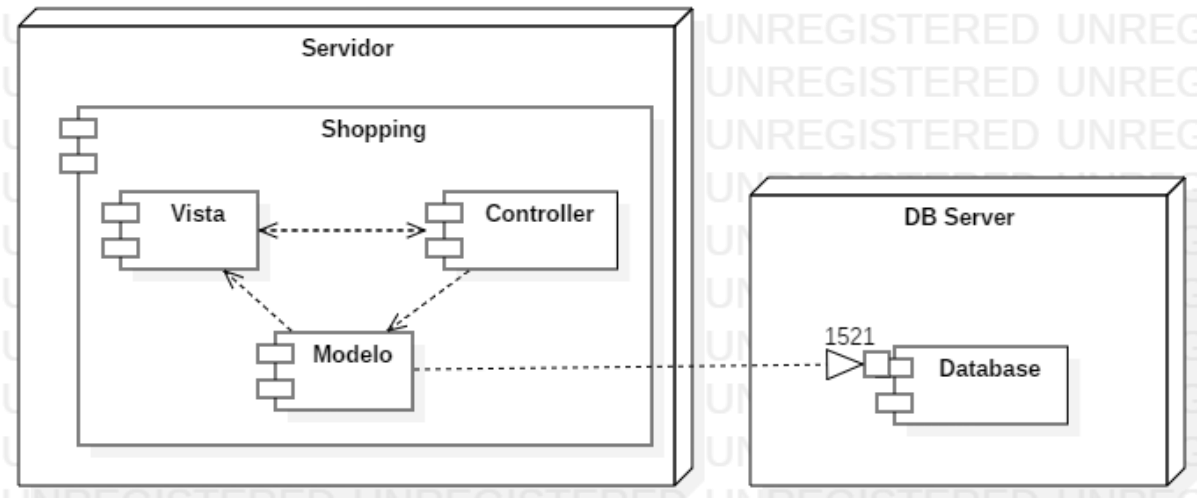


Figura 13 Diagrama de despliegue

5. Bibliografía

[1] E. Robson and E. Freeman, *Head First Design Patterns Poster*. 2005.

[2] N. Rozanski and E. Woods, *Software Systems Architecture*. 2005.

[3] M. Fowler, *Patterns of Enterprise Application Architecture*, vol. 48, no. 2. 2002.

[4] OMG, “Object Management Group.” [Online]. Available: https://www.omg.org/.

[5] Object Management Group, “Unified Modeling Language UML.” [Online]. Available: http://www.uml.org/.

[6] P. Kruntchen, “Architectural blueprints–the” 4+ 1” view model of software architecture,” *IEEE Softw.*, vol. 12, no. November, pp. 42–50, 1995.

[7] G. Florijn, “Architectural styles and patterns,” 2015.

[8] Portal ISO, “ISO/IEC 25010.” [Online]. Available: http://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0.