**SOFTWARE ARCHITECTURE DOCUMENT**

**(SAD)**

Contents

[**1. Context** 1](#_Toc248926143)

[**2. Functional View** 1](#_Toc248926144)

[**3. Process View** 1](#_Toc248926145)

[**4. Non-functional View** 2](#_Toc248926146)

[**5. Architectural Constraints** 2](#_Toc248926147)

[**6. Architectural Principles** 3](#_Toc248926148)

[**7. Logical View** 3](#_Toc248926149)

[**8. Interface View** 3](#_Toc248926150)

[**9. Design View** 4](#_Toc248926151)

[**10. Infrastructure View** 4](#_Toc248926152)

[**11. Deployment View** 5](#_Toc248926153)

[**12. Operational View** 5](#_Toc248926154)

[**13. Security View** 6](#_Toc248926155)

[**14. Data View** 7](#_Toc248926156)

[**15. Technology Selection** 7](#_Toc248926157)

[**16. Architecture Justification** 7](#_Toc248926158)

1. **Context**

El siguiente documento presenta una visión general de la arquitectura de la aplicación, siendo representado a través de una serie de diferentes vistas arquitectónicas que involucran diferentes aspectos de la Aplicación Web.

Future service es un software que permite agilizar y optimizar el Sistema de pedidos en un restaurante, a la vez que implementa funcionalidades adicionales, que sirven para que tanto los empleados como los administradores, puedan tener una herramienta que agilice sus tareas.

**2. Functional View**

**Las principales funcionalidades de nuestro software son:**

1. Toma y notificación de pedidos.

Historia de usuario 1.1: Como mesero quiero tener un listado de productos para ofrecer a los clientes.

Historia de Usuario 1.2: Como despachador quiero ver todos los pedidos que están en cola.

Historia de usuario 1.3: como despachador quiero tener un mensaje al recibir un nuevo pedido.

Historia de usuario 1.3: como despachador quiero notificar al mesero que el producto ya está listo para entregar.

Historia de usuario 1.4: Como mesero quiero definir una mesa como ocupada al registrar un nuevo pedido.

Historia de usuario 1.5: Como mesero quiero agregar de a un producto al pedido.

Historia de usuario 1.6: Como mesero quiero eliminar un producto del pedido.

2. Agregar y consultar productos del inventario.

Historia de usuario 2.1: Como administrador quiero consultar los productos que hay en el inventario.

Historia de usuario 2.2: Como administrador quiero agregar al inventario nuevos productos.

3. Generar factura de cobro.

Historia de usuario 3.1: Como cajero quiero que se me muestre en pantalla el pedido de cada mesa.

Historia de usuario 3.2: Como cajero quiero generar una factura de cobro.

4. Consultar el reporte de ventas.

Historia de usuario 4.1: Como administrador quiero consultar los productos más vendidos.

Historia de Usuario 4.2: Como administrador quiero consultar el total de ventas.

5. Tener una base de datos de los clientes.

Historia de usuario 5.1: Como mesero quisiera poder agregar un nuevo cliente.

Historia de usuario 5.2: Como cliente quisiera acumular puntos de acuerdo al valor de la compra.

Historia de usuario 5.3: Como cliente quisiera poder redimir mis puntos a cambio de algún producto del menú u obtener descuentos.

Historia de usuario 5.4: Como cajero quiero consultar los puntos de un cliente cuando éste me lo solicita.

6. Manejo de la nómina.

Historia de usuario 6.1: Como administrador quiero crear un nuevo despachador.

Historia de usuario 6.2: Como administrador quiero crear un nuevo mesero.

Historia de Usuario 6.3: Como administrador quiero poder pagarle a la nómina.

Historia de Usuario 6.4: Como administrador quiero ingresar con una clave de acceso.

7. Generar una cola de canciones por reproducir a medida que cada cliente las va solicitando.

**4. Non-functional View**

Descrbe los requerimientos no funcionales del sistema dentro de las categorías usabilidad confiabilidad y perfomance

.

Confiabilidad:el sistema no debe fallar en los procesos de la toma y nitifucacion de cada uno de los pedidos realizados ya que esta es la principal función y es crítica para el establecimiento

Usabilidad:el sitio web es simple orientado a los casos de uso por ejemplo administrador mesero cajero despachador disminuyendo la necesidad de capacitación de los usuarios ya que el entorno es amigable con estos

Interoperabilidad:desde cualquier dispositivo con acceso a internet se puede acceder al sitio ya que está diseñado para visualizar de forma óptima todo el contenido

\* ESCALABILIDAD:

El Sistema esta construido con un arquitectura de capas permitiendo la modificación y evolución de este sin tener que alterar demasiado el código

**5. Architectural Constraints**

**En esta vista se presenta las restricciones normativas a la cuales está sujeto el proceso de desarrollo**

**ESTANDARES: la interfaz de usuario debe estar orientada a la web,debe ser posible visualizar el contenido utilizando cualquiera de los navegadores ,google Chrome ,mozilla Firefox ,entre otros**

**Interfaz web**

**Web services:este sistema posee interoperabilidad ya que el administrador poseera una aplicación móvil en la cual podrá tener acceso a informacion**

**TECNOLOGIA:**

**El desarrollo completo del sistema deberá estar realizado en la plataforma eclipse usando el lenguaje de programación JAVA**

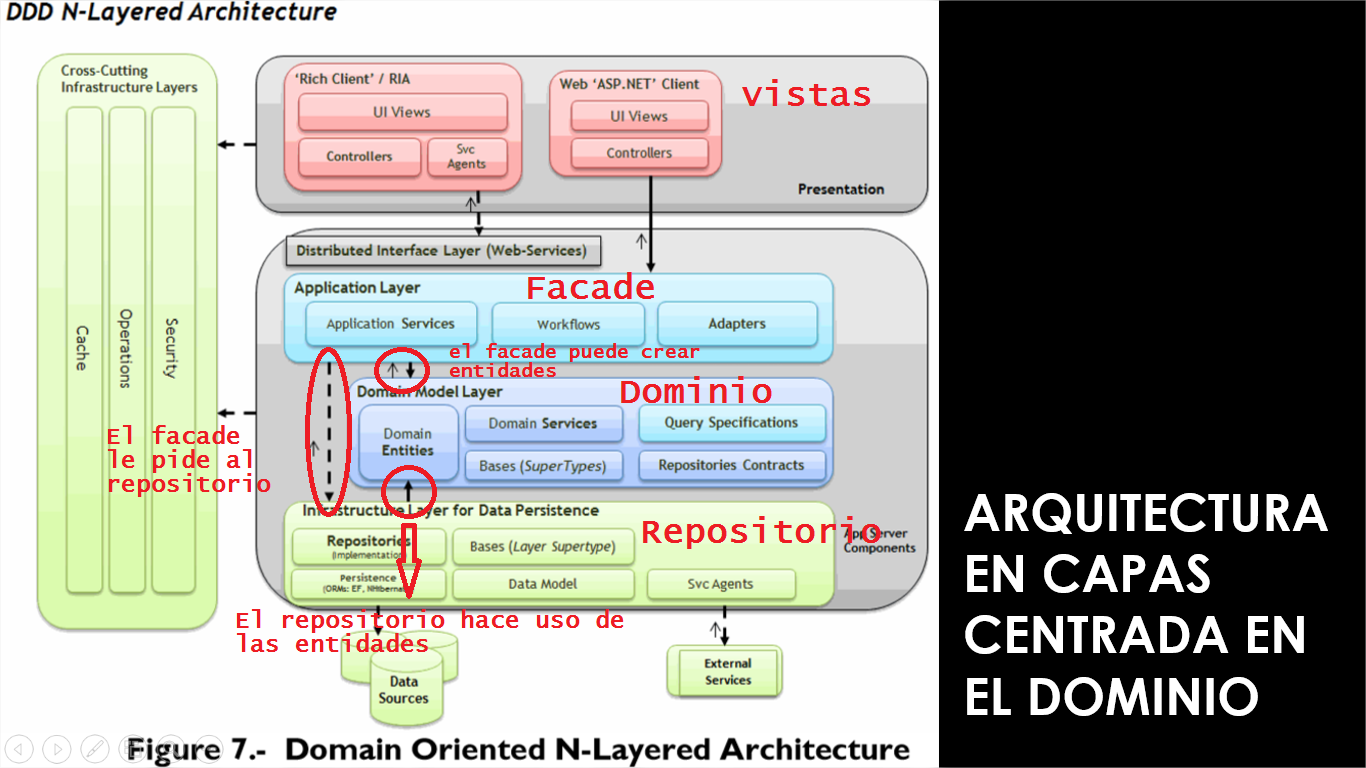
El sistema funcionará con la base de datos que el equipo de trabajo diseñe durante el proceso de desarrollo.

Se utilizará el lenguaje Java para la construcción del software y se empleará el servidor de Base de Datos MySQL; el entorno de desarrollo será la distribución Windows (SO).

**6. Architectural Principles**

Desarrollamos nuestro principio de arquitectura en capas, reconocido como “layers”. De esta manera se busca llegar a una solución que se acerque a la visión de nuestro proyecto: La agilización y optimización de tiempos para los establecimientos. El método de capas nos permite maniobrar mucho más fácil los aspectos más importantes y sobresalientes de la aplicación. Así, en cualquier caso, sea editar, crear una solución o solucionar algún problema, se podrá hacer directamente sobre el problema relacionado, en este aspecto, encontrándolo específicamente en la capa que lo contiene.

**7. Logical View**

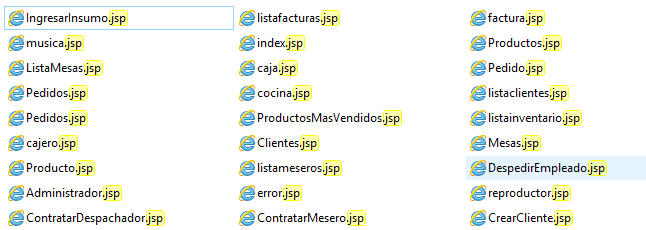
Se usó el modelo MVC más SOA; de esta manera se tienen 4 capas:

1. Vista
2. Servicio
3. Negocio
4. Datos

Como lo podemos observar en las siguientes ilustraciones.

Capa Vista:

Acá se encuentran todas la jsp que se encargan de enseñarle al usuario de una forma amigable todas las funciones requeridas por este. Nos podemos encontrar con jsp’s como la de ProductosMasVendidos.jsp que se encarga de enseñarle al cliente la lista de productos más vendidos en una tabla realizada por HTML5 y un Ajax implementado en JavaScript, ubicado en código.js. También podemos encontrar los controlladores encargados de gestionar los requerimientos del cliente.

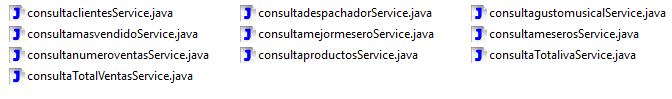


C:\Users\Crisman Carmona\Pictures\proyecto\codigo.PNG

C:\Users\Crisman Carmona\Pictures\proyecto\controllador.PNG

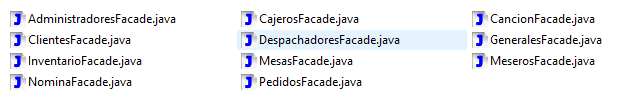
Capa Servicios:

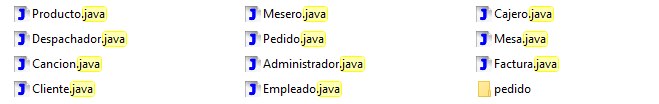
Acá encontramos los servicios que se colocan para el aprovechamiento de las unidades móviles, en este caso para Android. Por ejemplo, consultaclientesService.java, arroja un String con los nombres de todos los clientes, perfecto para el uso del adminitrador desde su Smartphone.



Capa Negocio:

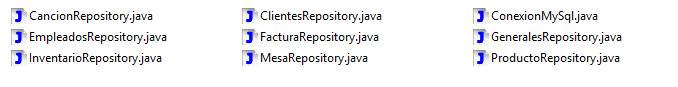
Acá se encuentran los orquestadores del negocio y el negocio en sí. Aquí está la lógica de nuestro negocio y las fórmulas que colocan a correr efectivamente nuestra aplicación y nos acercan a la visión.





Capa Datos:

Con esta capa recuperamos los datos necesarios para ‘jugar’ en nuestra aplicación, de esta manera, no habrían datos quedamos en la app y el usuario tendría los datos guardados en su propia base de datos.



**9. Design View**

The design view is where the lower level implementation details start to make an appearance. For example, this could include information such as how your architectural layering will be implemented through to documenting blueprints/common usage patterns for the technologies/frameworks you have chosen for the implementation. Depending on your team size, you may or may not find including this low level of detail useful.

* Is it well understood how the key use cases will be implemented?
* How are the chosen technologies used and combined?
* Are there common patterns across the architecture?
* If yes, are these well understood and documented?
* Are the diagrams (e.g. UML class and sequence) up to date and do they reflect reality?
* Is there too much detail?
* Are any common wheels being reinvented? If so, why aren’t vendor/open source products being used?
* If the code isn’t enough, is there enough information here to provide the rest of the development team with an overview/the intent of how the designs work?

La vista de diseño es donde los detalles de implementación de bajo nivel comienzan a hacer acto de presencia. Por ejemplo, esto podría incluir información como cómo se implementará la estratificación arquitectónica a través de la documentación de planos / patrones de uso comunes para las tecnologías / marcos que ha elegido para la ejecución. Dependiendo del tamaño de su equipo, que puede o no puede encontrar la inclusión de este bajo nivel de detalles útiles.

• ¿Está bien entendido cómo se implementarán los casos de uso clave?

• ¿Cómo son las tecnologías elegidas usados ​​y combinados?

• ¿Hay patrones comunes a través de la arquitectura?

• Si es así, estas son bien comprendidos y documentados?

• ¿Los diagramas (por ejemplo, de clase UML y secuencia) al día y hacen que reflejan la realidad?

• ¿Hay demasiados detalles?

• ¿Están las ruedas comunes que se reinventan? Si es así, ¿por qué no son proveedores / productos de código abierto que se utilizan?

• Si el código no es suficiente, ¿hay suficiente información para proporcionar el resto del equipo de desarrollo con una visión general / la intención de cómo el diseño de trabajo?

**11. Deployment View**

The deployment view details how the software will be deployed onto the physical infrastructure.

* Is it clear how the software components will be deployed across the hardware elements described in the physical view? (e.g. one-to-one mapping, multiple software components per server, etc)
* If this is still to be decided, what are the options and have they been documented?
* Is it understood how memory and CPU will be partitioned between the processes running on a single hardware node?
* Which components are active-active and which are active-passive?
* Which components can be scaled-out?
* Is it clear how data is replicated across sites?
* Has the rollout and recovery strategy been defined (this might be in a separate document, but referenced)?
* How are the components installed and configured?

La vista despliegue detalla cómo el software se desplegará en la infraestructura física.

• ¿Está claro cómo se desplegarán los componentes de software a través de los elementos de hardware que se describen en el punto de vista físico? (por ejemplo, de uno a uno de mapeo , múltiples componentes de software por servidor , etc)

• Si esto aún no se ha decidido , ¿cuáles son las opciones y se les ha documentado ?

• ¿Es entiende cómo se repartió la memoria y la CPU entre los procesos que se ejecutan en un único nodo hardware ?

• ¿Qué componentes son activo-activo y que son activo-pasivo ?

• ¿Qué componentes se pueden escalar de salida ?

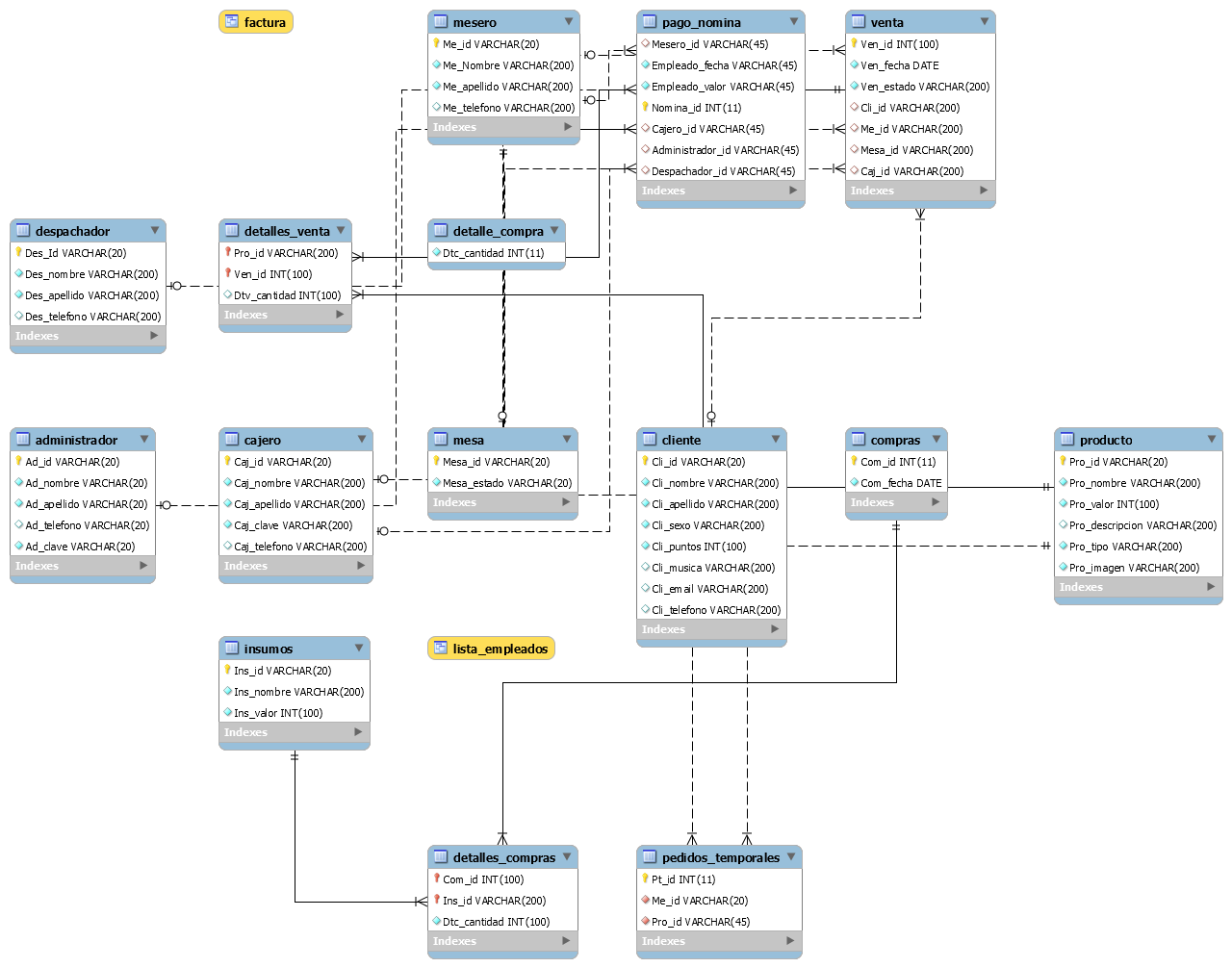
• ¿Está claro cómo se replica datos a través de los sitios ?

• ¿Se ha definido la estrategia de lanzamiento y recuperación ( esto podría ser en un documento separado , pero que se hace referencia ) ?

• ¿Cómo son los componentes instalados y configurados ?

**14. Data View**

Mysql 7



**15. Technology Selection**

Including a technology selection section in your software architecture document gives you somewhere to document the decisions that went into choosing or not choosing technologies. It’s often useful to have a summary of these decisions so that they can be referred to later in the project.

* Is it clear why the selected technologies were chosen?
* If there were options, why were they not chosen?
* Do they all fit in with the constraints outlined previously?
* Are all software and hardware tiers covered?

La inclusión de una sección de selección de la tecnología en su documento de arquitectura de software le da un lugar para documentar las decisiones que iban a elegir o no elegir tecnologías. A menudo es útil tener un resumen de estas decisiones para que puedan ser referidos más adelante en el proyecto.

• ¿Está claro por qué se eligieron las tecnologías seleccionadas ?

• Si había opciones , ¿por qué fueron elegidos no ?

• ¿Todos encajan con las limitaciones señaladas anteriormente ?

• ¿Están cubiertos todos los niveles de software y hardware ?

**16. Architecture Justification**

This section provides a way for you to explicitly state how the non-functional requirements you stated in the non-functional view are satisfied by your architecture. This is an important yet often forgotten piece of the architecture puzzle, and clearly communicating your architecture’s fitness for purpose will help provide everybody with the confidence that your solution will work. If you built an executable reference architecture to prove some of the key non-functional requirements, this is where you can reference that work and the proof of its success.

* For each of the non-functional requirements, is it explicit how the architecture satisfies it?
* In the case of performance and scalability targets, are the test cases and results referenced?
* If true 24x7 availability is required, is redundancy and automatic failover built into all aspects of the architecture?
* Are there any single points of failure?
* What happens if a component fails?
* What happens if an external system you rely on fails?
* Does this affect your availability? Are your transactions ACID?
* Would you have 2PC transactions in-doubt?
* Can you recover from a system failure?
* Who is responsible for system recovery and failover?
* Can you recover in a business continuity scenario?
* Will all data have been replicated between sites?
* How do you tell system components to use alternative resources in the event of DR/BCP?
* Has the architecture been reviewed by in-house security risk teams (if applicable)?

Esta sección proporciona una manera para que usted declare explícitamente cómo los requerimientos no funcionales que declaró en la vista no funcionales están satisfechos por su arquitectura. Esta es una pieza importante, aunque a menudo olvidado del rompecabezas arquitectura, y claramente comunicar la aptitud de su arquitectura para el propósito ayudará a proporcionar a todos con la confianza de que su solución va a funcionar. Si construyes una arquitectura de referencia ejecutable para probar algunos de los requisitos no funcionales clave, aquí es donde usted puede hacer referencia a que el trabajo y la prueba de su éxito.

• Para cada uno de los requisitos no funcionales, es explícita cómo la arquitectura que satisface?

• En el caso de los objetivos de rendimiento y escalabilidad, son los casos de prueba y los resultados de referencia?

• Si se requiere cierto disponibilidad 24x7, es la redundancia y failover automático integrado en todos los aspectos de la arquitectura?

• ¿Hay puntos únicos de fallo?

• ¿Qué pasa si un componente falla?

• ¿Qué sucede si un sistema externo que se basan en un error?

• ¿Afecta esto a su disponibilidad? Son sus transacciones ACID?

• ¿Tendría transacciones 2PC en duda?

• ¿Puede recuperarse de un fallo del sistema?

• ¿Quién es responsable de la recuperación del sistema y de conmutación por error?

• ¿Se puede recuperar en un escenario de continuidad del negocio?

• ¿Todos los datos han sido replicados entre los sitios?

• ¿Cómo se le dice a los componentes del sistema para utilizar recursos alternativos en caso de DR / BCP?

• ¿Se ha revisado la arquitectura por equipos de riesgos de seguridad en la casa (si procede)?

**Referencia**

http://www.codingthearchitecture.com/pages/book/software-architecture-document-guidelines.html