GESTIONATUSPEDIDOS

MEMORIA

Memoria del Proyecto final de la asignatura Ingeniería de Software II

"GestionaTusPedidos"

Profesor: Alberto García Hidalgo
Alumno: Hibjan Cardona Juan Andrés, Prado De Souza Leonardo, Jiahao Ji
Grupo 4.1

INSTRUMENTO DE DOMINIO PÚBLICO EN LOS TÉRMINOS ESPECIFICADOS EN LA LICENCIA DE USO.
PROHIBIDO SU USO COMERCIAL Y CUALQUIER MODIFICACIÓN NO AUTORIZADA.

Índice

Introducción

1.1. Objetivos del proyecto

El proyecto GestionaTusPedidos (GTP) tiene como objetivo facilitar la comunicación entre la cocina y los camareros en cualquier restaurante. Además, ofrece algunas funcionalidades para mejorar la administración de los recursos materiales del negocio.

1.2. Alcance del proyecto

Se proporcionan las funcionalidades básicas de clientes y usuarios. Así, los clientes pueden inscribirse y sumar puntos con pedidos. Los usuarios pueden ser de distintos tipos, cada uno con sus limitaciones (de acuerdo con los casos de uso, ver 2.1 y 2.3), teniendo cada uno su contraseña. Los camareros podrán subir pedidos, modificarlos o borrarlos del estado actual de pedidos pendientes. La cocina podrá cambiar su estado a listo, permitiendo que los camareros o repartidores estén notificados del cambio. Además se ofrece un subsistema de proveedores que permite tener el estado actual de deuda con los distintos proveedores, bien como sus datos.

GTP no gestionará los menús, precios y notaciones de cada pedido/plato, la modificación de la estructura textual deberá ser hecha manualmente por los restaurantes. El software tampoco se encargará de relacionar los pedidos a los repartidores.

Se entregará, junto al software, una estructura básica de archivos en formato JSON, para que el restaurante pueda modificarlos y un manual de uso.

1.3. Estructura de la memoria

Esta memoria está dividida en Introducción, Análisis de requisitos, Diseño del sistema, Implementación y Conclusiones y trabajo futuro, como reflejado inicialmente en el Índice. Además cuenta con una série de apéndices que componen el conjunto de documentos entregados:

- Manual de Uso - Manual que contiene las instrucciones para el uso adecuado del programa;
- Plan de Proyecto - Plantilla inicial utilizada para definir el entorno del software y algunos aspectos de la distribución del trabajo;
- Documento de Pruebas - Documento que contiene las pruebas hechas y su la estructura de evaluación de calidad;
- Diagramas UML - El conjunto de diagramas UML utilizados en el proyecto, en formato .rar;

- Informe del trabajo - Informe requerido por el cliente, evaluando la participación de cada miembro del equipo;

Análisis de requisitos

2.1. Requisitos funcionales

- Inicio de sesión: Los usuarios podrán iniciar sesión en la aplicación utilizando un nombre de usuario y una contraseña proporcionados por el administrador del sistema.

- Gestión de roles: Los usuarios tendrán diferentes funcionalidades disponibles según su rol en la aplicación.

- Funcionalidad de Cocina: El usuario con el rol "cocina" podrá acceder a la funcionalidad de Cocina, que incluirá una tabla de platos por preparar. El usuario podrá marcar los platos que estén listos, lo que los eliminará de la tabla.

- Funcionalidades del Camarero: El usuario con el rol "camarero" tendrá acceso a las funcionalidades de Recogidas, Clientes y Pedidos. En la funcionalidad de Recogidas, se mostrará una tabla de platos cocinados pero aún no recogidos. En la funcionalidad de Clientes, el camarero tendrá las funcionalidades de crear, leer, actualizar y eliminar (CRUD) para gestionar la información de los clientes. En la funcionalidad de Pedidos, también tendrá las funcionalidades CRUD para gestionar los pedidos, en la cual podrá añadir platos al pedido y cada plato podrá tener ingredientes y cantidades modificables. Y con el delete del pedido, se obtiene una cuenta detallada en la que incluye cada plato, su precio, y la cantidad de ese plato, y el total a pagar de esa mesa o pedido delivery.

- Funcionalidades del Administrador: El administrador tendrá acceso a todas las funcionalidades de la aplicación, es decir, Cocina, Recogidas, Clientes, Proveedores y Pedidos.

2.2. Requisitos no funcionales

- Usabilidad: La aplicación es intuitiva y fácil de usar, con una interfaz clara y comprensible para usuarios de diferentes niveles de experiencia técnica. Para lo cual tiene una interfaz minimalista con diferenciación de colores según la operación que el usuario desee realizar. Para así asociar ese color a

la operación y minimizar el tiempo de búsqueda del botón y el margen de error de presionar un botón equivocado.

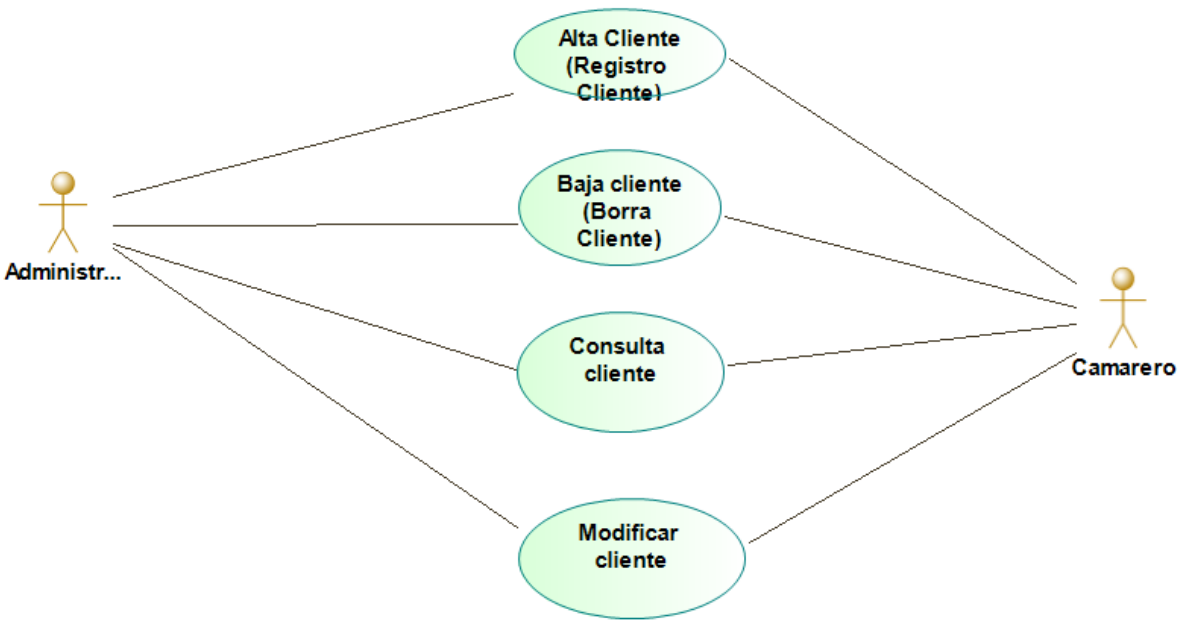
- Rendimiento: La aplicación es capaz de manejar la carga de trabajo esperada, proporcionando una respuesta rápida y tiempos de carga mínimos, a fin de asegurar una experiencia fluida para los usuarios.

- Integración: La aplicación es capaz de integrarse con otros sistemas o plataformas utilizados en el restaurante, como sistemas de pago, sistemas de inventario o servicios de entrega a domicilio, para garantizar una gestión eficiente y sin problemas de los pedidos. Debido a que la arquitectura permite la extensión de estas funcionalidades sin dificultad.

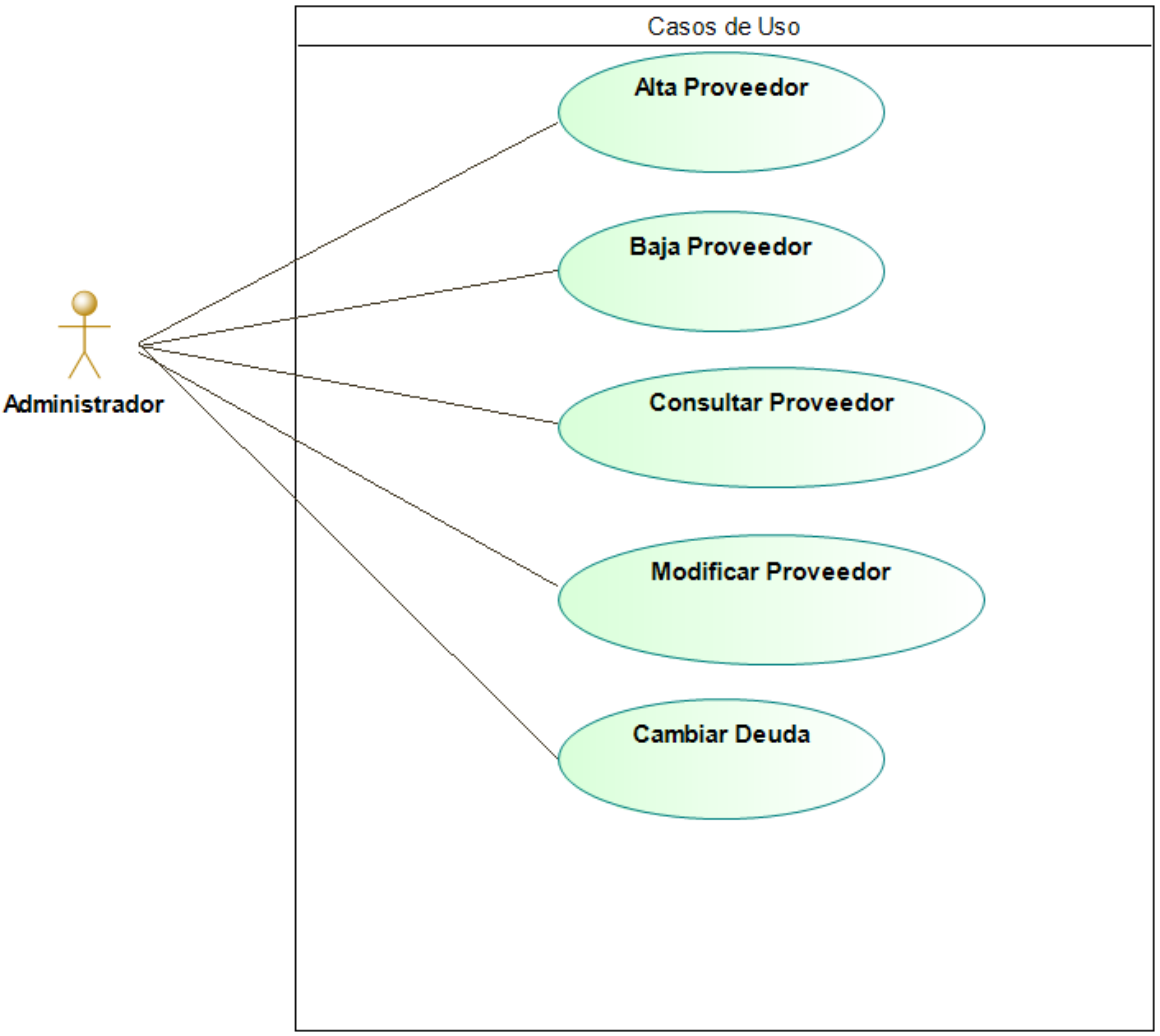
2.3. Casos de uso

A continuación se encuentran los diagramas en los que se hallan los actores con las distintas actividades que pueden realizar estos.

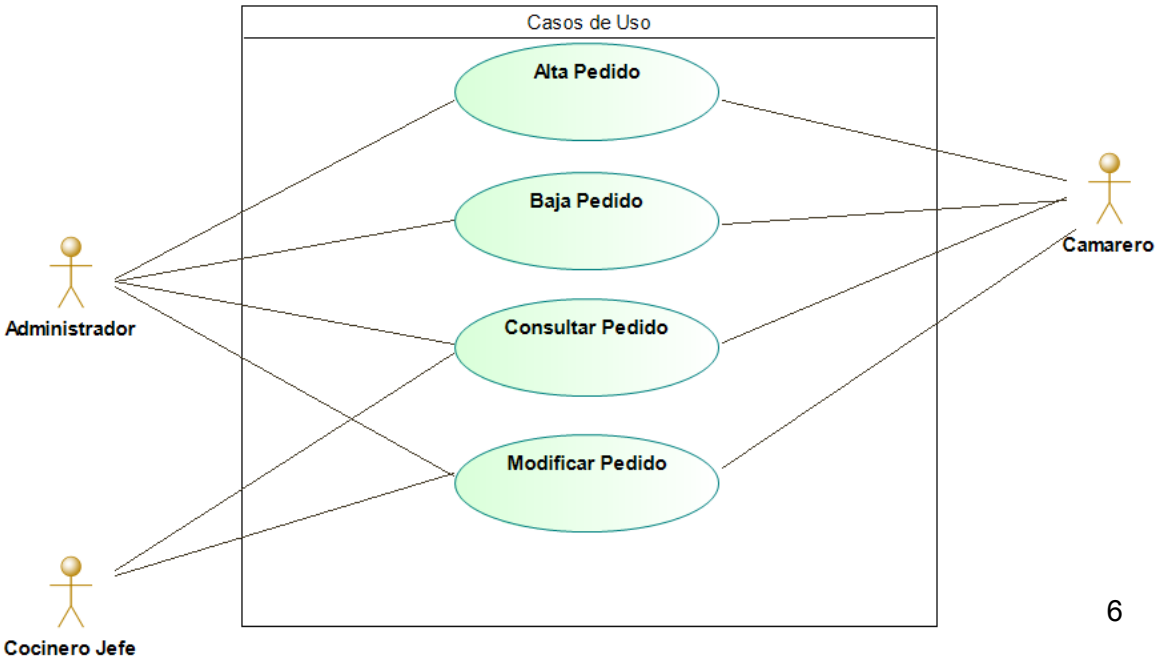
Subsistema Clientes:



Subsistema de Proveedores:



Subsistema de Pedidos:



Diseño del sistema

3.1. Arquitectura del sistema

GTP se basa en una arquitectura multicapas (tres capas). En esta estructura, el sistema se divide en capas, donde cada capa tiene una responsabilidad específica y se comunica con las otras capas. Por lo general, se distinguen tres capas principales: la capa de presentación (GTP lo hace con una interfaz GUI), la capa de lógica de negocio y la capa de persistencia de datos (acceso y almacenamiento de datos, hecho con JSON). La capa de presentación se encarga de la interacción con el usuario, la capa de lógica de negocio gestiona la funcionalidad principal del sistema y la capa de persistencia de datos maneja el almacenamiento y recuperación de información.

3.1.1. Descripción de la arquitectura y patrones utilizados

La implementación de dicha arquitectura se estructura utilizando una serie de patrones de diseño, que se listan a continuación:

- Patrón MVC: en MVC, el modelo representa los datos y la lógica de negocio, la vista se encarga de la presentación y la interfaz de usuario, y el controlador actúa como intermediario entre el modelo y la vista, gestionando las interacciones y el flujo de datos. El patrón fomenta la separación de responsabilidades, lo que facilita la modularidad, la reutilización y la mantenibilidad del código. Además, permite una gestión eficiente de cambios en la interfaz de usuario y en la lógica, ya que los componentes están desacoplados. Hemos elegido este patrón porque concretiza de manera muy directa la arquitectura en multicapas.

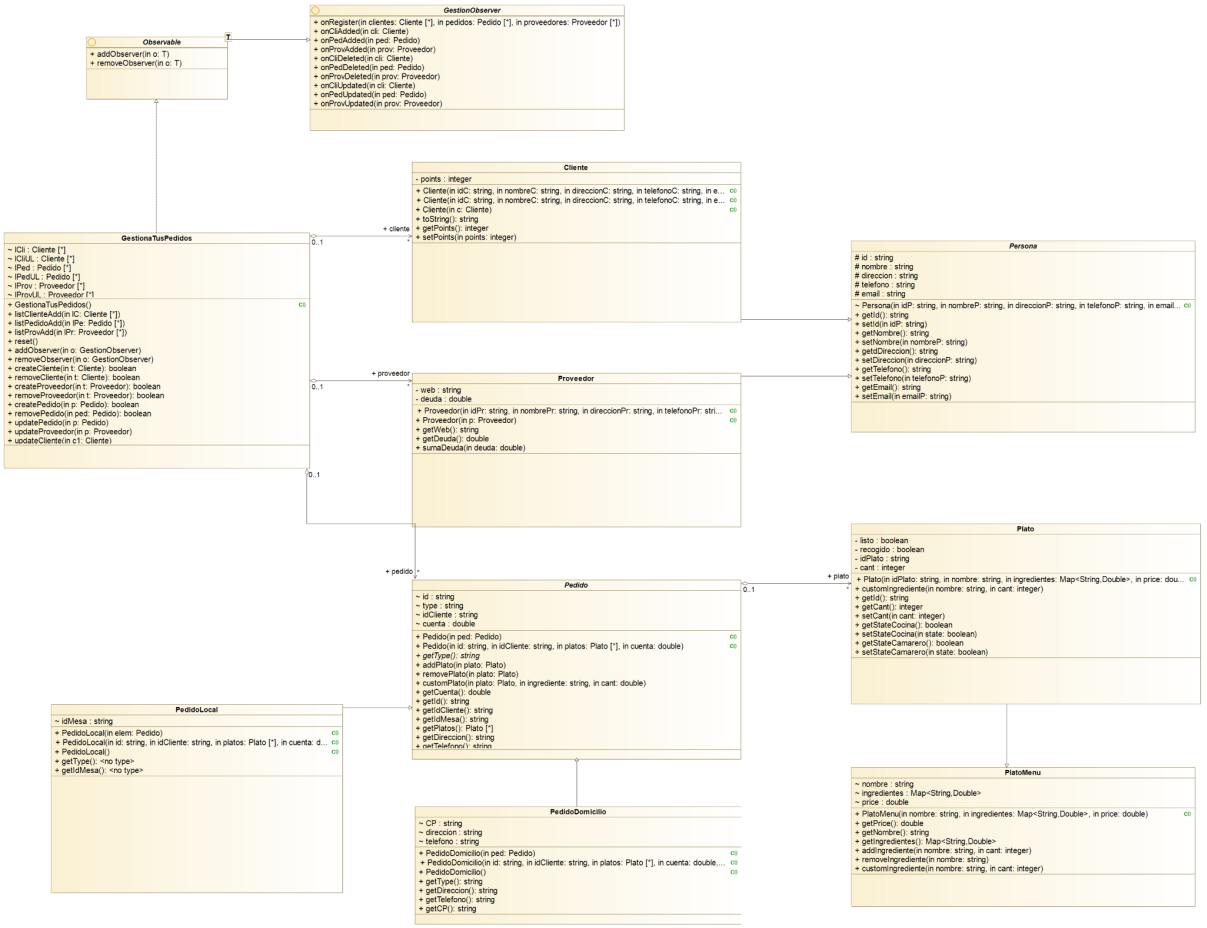
- Patrón Builder y Factory: este patrón separa el proceso de construcción del objeto final de su representación. El Builder define métodos para configurar las propiedades del objeto en cada paso, y al final, se obtiene el objeto completo a través de un método de construcción. Este patrón proporciona un importante aspecto de extensibilidad, especialmente con respecto al subsistema Pedido, pues toda y cualquier complejidad extra se añadirá directamente en su Builder. El patrón de diseño Factory (Factoría) es un patrón creacional que se utiliza para crear objetos sin especificar explícitamente su clase concreta. En este patrón, se define una interfaz común llamada fábrica que declara un método para crear objetos. Las clases concretas que implementan esta interfaz son responsables de crear instancias de objetos concretos. En nuestro caso, juntamos ambos patrones y utilizamos un BuilderBasedFactory, que condensa ambos patrones en la creación de los objetos.

- Patrón Observable: en este patrón, un objeto observable (nuestro modelo), notifica a una lista de objetos interesados, llamados observadores, cuando se produce un cambio en su estado. El observable mantiene una lista de observadores registrados y les envía notificaciones cuando se actualiza. Esto permite una comunicación eficiente y desacoplada entre el sujeto y los observadores, ya que los observadores pueden suscribirse y cancelar su suscripción en cualquier momento. El patrón soluciona la comunicación de cambios de estado entre el View y el Modelo en nuestro proyecto.

- Patrón DAO: el patrón de diseño DAO (*Data Access Object*) se utiliza para separar la lógica de acceso a datos de la lógica de negocio. El objetivo principal del patrón DAO es proporcionar una capa de abstracción entre el software y la fuente de datos, lo que permite un acoplamiento flexible y un mantenimiento más sencillo. El DAO actúa como una interfaz entre la lógica de negocio y los diferentes proveedores de datos, proporcionando métodos estandarizados para realizar operaciones de creación, lectura, actualización y eliminación (CRUD) en los objetos. Esto permite que los cambios en la fuente de datos se realicen sin afectar directamente a la lógica de negocio, y también facilita la sustitución de diferentes fuentes de datos sin alterar la funcionalidad principal de la aplicación. Aquí se utiliza para separar las principales clases del Modelo y su estructura de datos persistente (JSON).

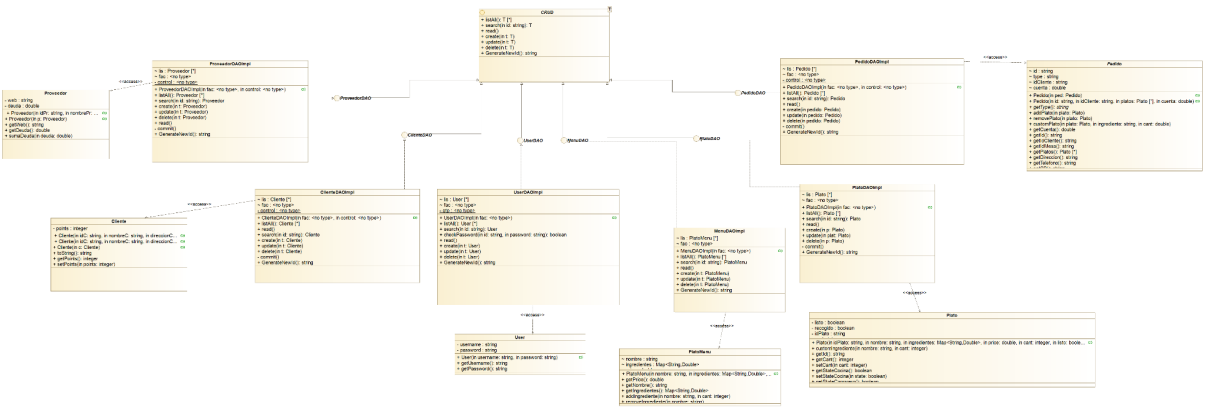
3.2. Diagramas de clases

Sigue el Modelo de Dominio que refleja la relación entre las clases:



3.2.1. Descripción de las clases y sus relaciones

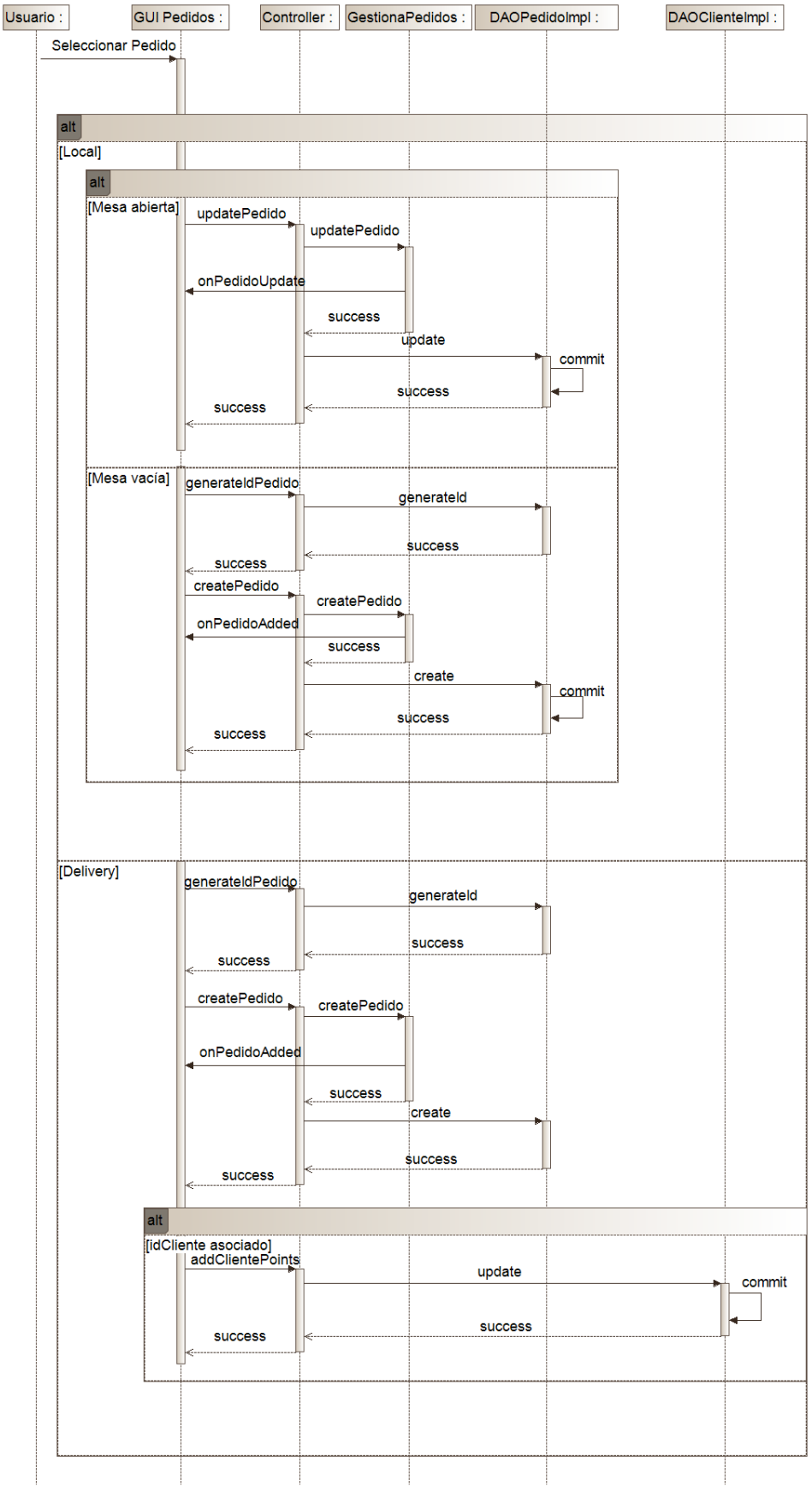
Además del Modelo de Dominio en 3.2. sigue la estructura DAO que el software implementa:



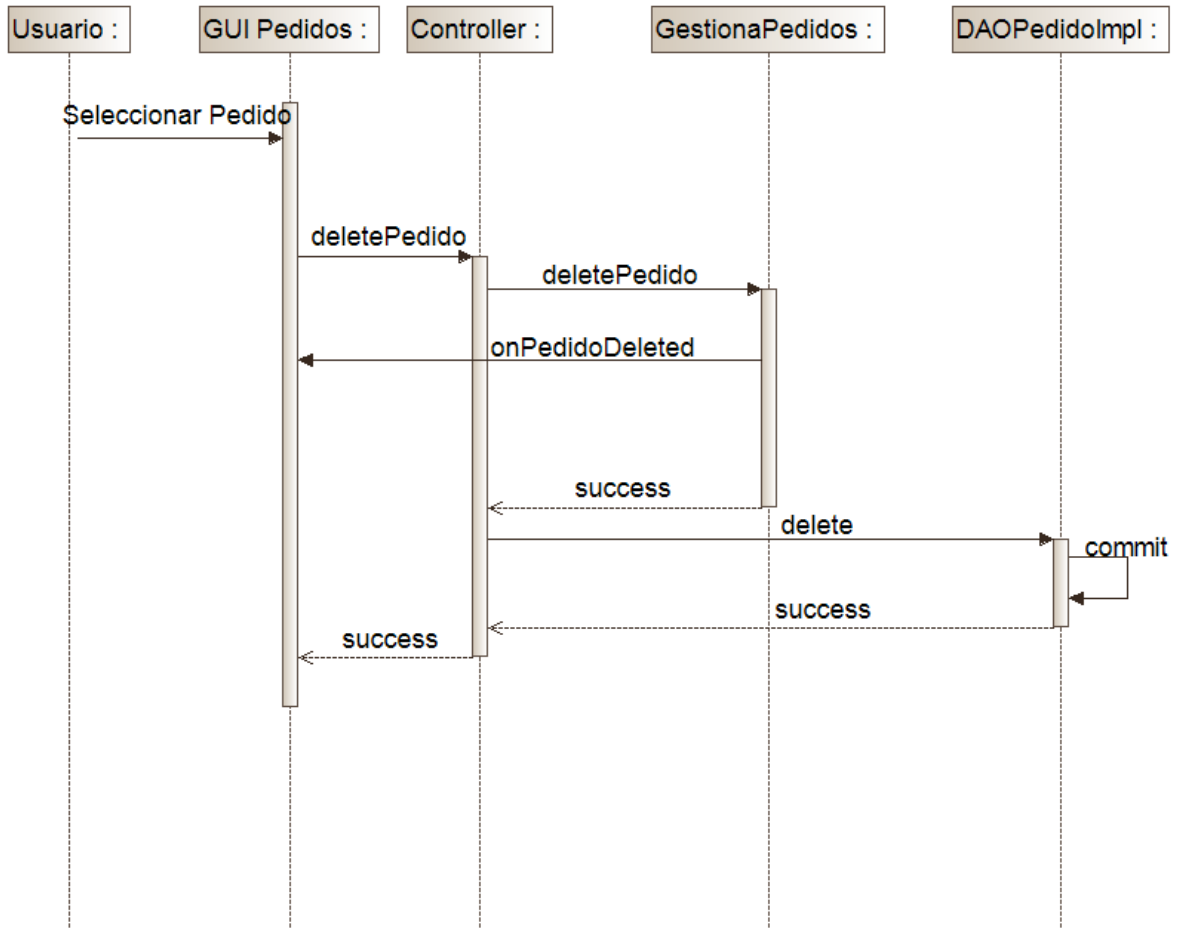
3.3. Diagramas de secuencia

A continuación dejamos los diagramas de secuencia correspondientes a los principales casos de uso de Pedido que condensa la principal estructura de flujo de ejecución:

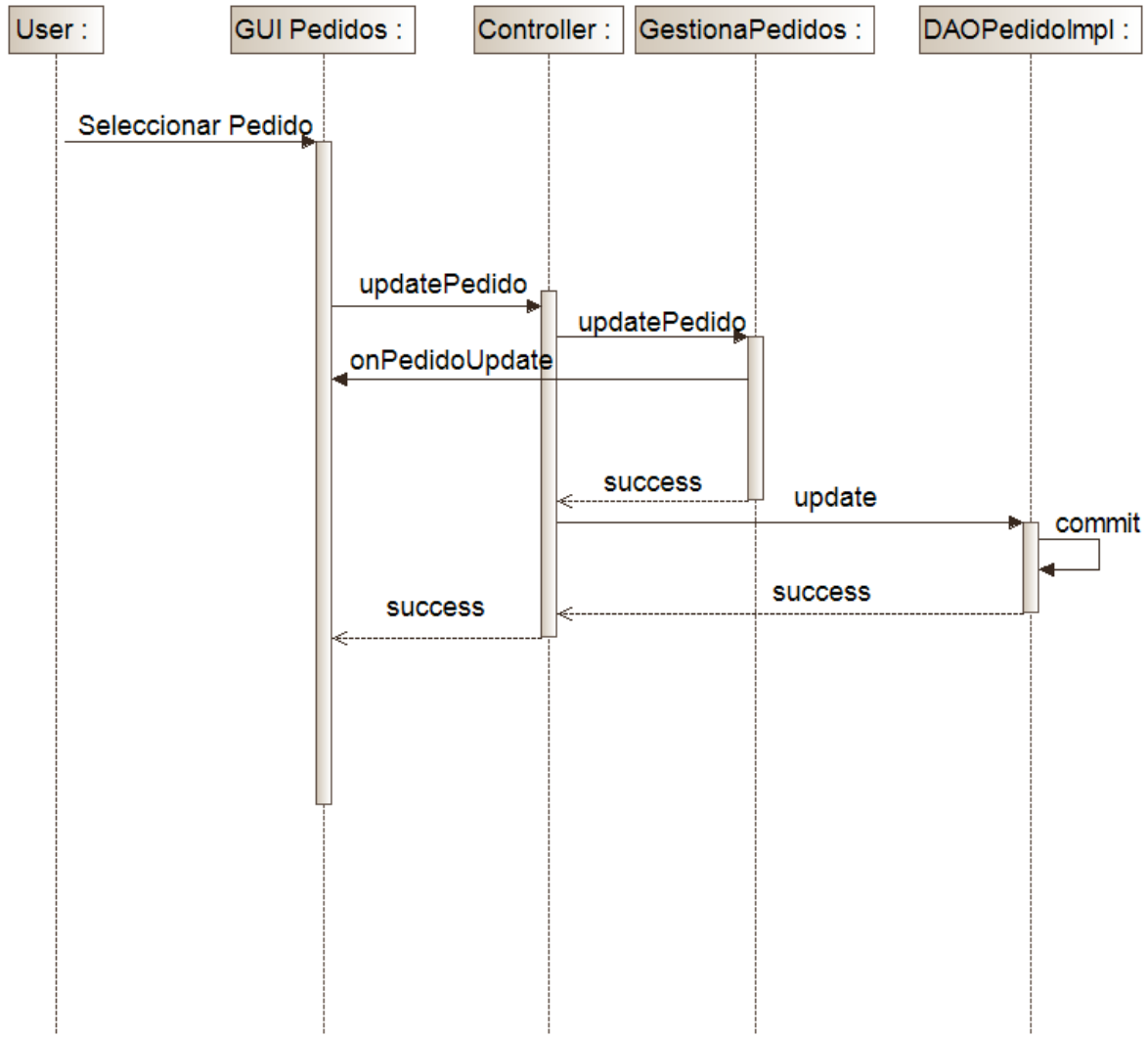
Alta pedido:


Usuario :
GUI Pedidos :
Controller :
GestionaPedidos :
DAOPedidoImpl :
DAOClienteImpl :
Seleccionar Pedido
alt
[Local]
alt
[Mesa abierta]
updatePedido
updatePedido
onPedidoUpdate
success
update
commit
success
success
[Mesa vacía]
generateIdPedido
generateId
success
success
createPedido
createPedido
onPedidoAdded
success
create
commit
success
success
[Delivery]
generateIdPedido
generateId
success
success
createPedido
createPedido
onPedidoAdded
success
create
success
success
alt
[idCliente asociado]
addClientePoints
update
commit
success
success

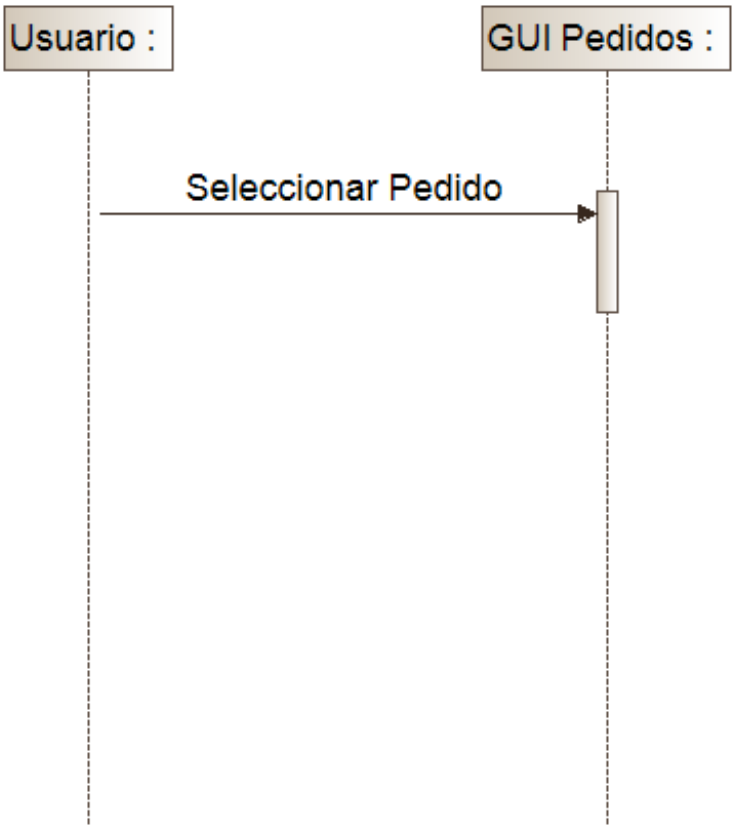
Baja pedido:


Usuario :
GUI Pedidos :
Controller :
GestionaPedidos :
DAOPedidoImpl :
Seleccionar Pedido
deletePedido
deletePedido
onPedidoDeleted
success
delete
commit
success
success

Modificar pedido:



Consultar pedido:



Implementación

4.1. Lenguajes de programación y tecnologías utilizadas

- Para desarrollar la aplicación se utilizó en su integridad Java 17, y en el entorno de desarrollo Eclipse 2022-03 (4.23.0).

- Se utilizan las librerías JRE System Library, JUnit5, y JSON-lib

4.2. Descripción del proceso de desarrollo

El desarrollo se realizó basándose en uno de los modelos de proceso evolutivo: el proceso unificado de desarrollo. El cual se divide en distintas fases, por las que pasa cada flujo de trabajo (requisitos, análisis y diseño, implementación y prueba):

Requisitos:

- Fase de inicio: Definimos los requisitos funcionales y no funcionales, mencionados anteriormente. Definiendo qué funcionalidades debía de suplir

la aplicación, y las herramientas que íbamos a utilizar para desarrollarla, y los distintos subsistemas que tendría.
- Fase de elaboración: Se realizaron bocetos de sendos diagramas UML de Casos de Uso y Diagramas de Actividades.
- Fase de construcción: Se puso por escrito en el Plan de Proyecto las decisiones tomadas con respecto a los requisitos.

Análisis y Diseño:
- Fase de inicio: Se definió la arquitectura multicapa que se implementará, y por tanto se realizaron decisiones sobre los distintos patrones que utilizaremos para hacer el código extensible.
- Fase de elaboración: Se realizó un boceto del diagrama del dominio en el que se viese reflejada la arquitectura y patrones, discutiendo, a su vez, las estructuras de datos y algoritmos que debían de tener las distintas clases para conseguir una considerable extensibilidad y eficiencia. Y se discutió un boceto de la interfaz gráfica y cómo el usuario interactúa con la aplicación.
- Fase construcción: Se definió el diagrama del dominio, y se realizaron los diagramas de secuencia de cada subsistema, retornando en ocasiones a la fase de inicio y elaboración, iterativamente, integrando la GUI.

Implementación:
- Fase de elaboración: Se trasladan los modelos UML a un proyecto Java.
- Fase de construcción: Se discuten especificaciones sobre el desarrollo, y, a la vez que se implementa lo discutido, los encargados de cada subsistema discuten sus implementaciones para asegurar el cumplimiento de los requisitos y diseño, y proponer posibles soluciones a problemas que se enfrentaban en subsistemas ajenos.
- Fase de transición: Se realizan ajustes de la interfaz gráfica y se prepara el proyecto para su integración con las pruebas que se realizarán.

Prueba:
- Fase de elaboración: Se definen las clases JUnit utilizadas para realizar las pruebas y se definen casos de prueba para la GUI.
- Fase de construcción: Se implementan las clases JUnit y se comprueban los resultados equivalentes. Y, los encargados de cada subsistema se encargan de probar todas las funcionalidades referentes a la interfaz para comprobar que funcionen correctamente.
- Fase de transición: Se registran los errores encontrados.

De tal manera, luego de la etapa de Prueba, se vuelve a la primera etapa de Requisitos, en la que se toma en cuenta los resultados de la última etapa, y se continúa el proceso. Pero, esta segunda vez que se itera, las etapas son más reducidas en tiempo y esfuerzo, pues tienen objetivos menos amplios.

Conclusiones y trabajo futuro

5.1. Logros y desafíos enfrentados

Se evalúa que se ha logrado un desarrollo adecuado a las especificaciones del cliente. Se ha invertido considerablemente más en la fase de diseño y arquitectura que en desarrollo por la naturaleza del proyecto, aunque no en demasía. Todavía se hace un balance importante con respecto a organización del personal, ya que se han encontrado una serie de retos con respecto al cumplimiento de plazos y clareza de diseño.

5.2. Propuestas para futuras mejoras o desarrollos

Definimos las propuestas futuras y mejoras en tres partes, la visual, la funcional y la optimización.

- Visual: Embellecer la GUI con íconos, descripciones presentes en el Manual de Usuario y afines con vista a tener una interfaz gráfica más asequible al usuario medio.
- Funcional: invertir en una clase de ingredientes o stock, que trabaje en cooperación con el subsistema de Proveedores. Así, mejorando y ampliando sustancialmente el rango de posibilidades del software.
- Optimización: implementación de hebras para permitir la concurrencia de vários programas ejecutando a la vez.