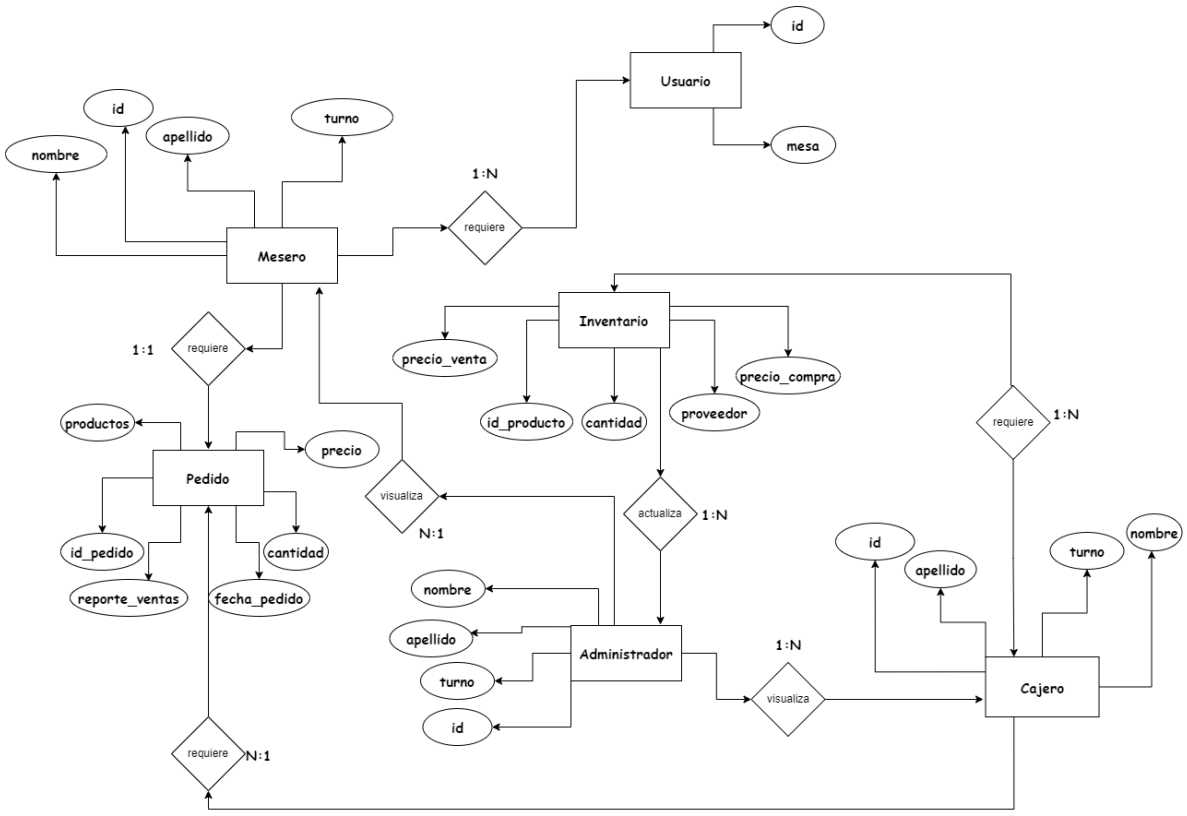
Bar mou

Documento de arquitectura

Modelo ER



Bar
mou

Diagrama de despliegue

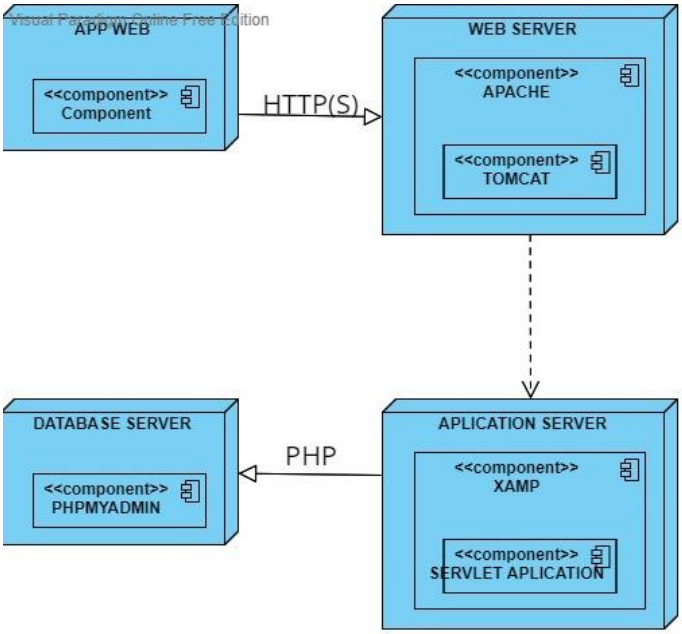


Diagrama de secuencia

Buenas Particas de desarrollo

Algoritmos de implementación

- Ordenamiento rápido Quicksort
- Ordenamiento de burbuja Bubblesort
- Algoritmo Voraz de Greedy

Requerimientos Funcionales

- **Seleccionar el talento y los recursos apropiados**

Captar y seleccionar el talento humano con las destrezas necesarias y experiencia relevante es vital para garantizar el éxito del proyecto. Es importante asignar el trabajo apropiado a la persona indicada e invertir en herramientas que aumentan la productividad y eficiencia del equipo de desarrollo mediante el manejo del software.

- **Escoger el proceso de desarrollo apropiado**

El ciclo de vida del desarrollo del software tiene una fuerte dependencia del proceso elegido. El modelo en cascada, la metodología ágil, el enfoque iterativo en espiral, son todos formas contrastadas de alcanzar el éxito.

- **Hacer presupuestos y estimaciones razonables**

Lo ideal es que no se prolonguen en plazos por hacer estimaciones poco realistas. Una planificación razonable depende de fijar bien los tiempos, el presupuesto, los recursos y los esfuerzos. Lo mejor es usar técnicas de

estimación y presupuestarias contrastadas. Intentar apretar las estimaciones para intentar acortar un proyecto normalmente termina en catástrofe.

- **Fijar hitos más pequeños**

Los grandes proyectos e hitos deben complementarse con mini-hitos para poder hacer mejor seguimiento, más control y mejor gestión de riesgos, y en general para mitigar incidencias de una forma más controlada. Los miembros del equipo deben reunirse para fijar estos mini-hitos y alinearlos con los grandes hitos para cumplir plazos y reducir los retrasos que pueden surgir por las interdependencias de las tareas que tienen asignadas.

- **Definir bien los requisitos**

Documentar de manera efectiva los requisitos es la columna vertebral para poder alinear el producto final con los objetivos empresariales. Es imperativo que se reúnan todas las partes (clientes, los responsables de empresa y los líderes de los equipos) para documentar los requisitos de forma clara y concisa, sin dejar lugar a lagunas o a la improvisación.

- **Definir la arquitectura del sistema**

Un buen arquitecto de aplicaciones garantizará una elección de arquitectura del sistema apropiada, teniendo en cuenta tanto los requisitos como las limitaciones y restricciones, si las hubiera. Buenas prácticas, tales como la identificación de amenazas y anti-patrones dentro del sistema son muy útiles.

- **Implementar el código de manera efectiva**

El uso de módulos más pequeños que están auto-probados, probados unitariamente y que se integran continuamente es una buena práctica muy

extendida. La automatización de herramientas build y la ejecución automatizada de pruebas.

- **Pruebas rigurosas y validación**

La planificación de pruebas, la creación de conjuntos de pruebas y la ejecución de estas son muy importantes con el fin de validar la funcionalidad desarrollada. De hecho, la planificación de las pruebas debe hacerse en paralelo a la fase de desarrollo. Igual de importante es la documentación que hagamos de las pruebas, informar de forma efectiva los errores, el rastreo de los errores y la corrección de estos. El uso de herramientas automatizadas al igual que procesos contrastados que aseguren que los errores se identifiquen en la fase más temprana posible y resueltos con el menor coste.

- **Documentación**

Aun siendo importante el propio software, igual de importante es toda la documentación sobre el que se apoya –el plan del proyecto, requisitos y especificaciones, Diseño de Alto Nivel (HLD), Diseño de Bajo Nivel (LLD), planes de pruebas, informes de las pruebas, informes de estado y la documentación para los usuarios.

- **Planificar sesiones de revisión de código**

Las revisiones de código muchas veces son más efectivas, y sin duda menos costoso, para encontrar errores que si solo hacemos pruebas. Las revisiones de todos los entregables, del código y de la documentación es algo que siempre se debe hacer. Estas sesiones de revisión del código, su gestión y la resolución de

conflictos que puedan darse en las mismas se hacen siguiendo una serie técnicas contrastadas y buenas prácticas

- **Control de calidad**

El control de calidad ayuda a sacar adelante los proyectos de desarrollo sin graves trastornos y de forma más rápida. Desde la detección de fallos hasta el establecimiento de métricas claves, las mejores prácticas en este terreno han demostrado ser un éxito a la hora de determinar si un proyecto está en condiciones de pasar a una nueva fase o si está listo para ser lanzado o entregado al cliente. Se deben fijar métricas y objetivos para asegurar que los requisitos, el diseño, el código, las pruebas y otras tareas vayan coordinadas y alienadas.

- **Instalación y despliegue eficaz**

En multitud de ocasiones cuando ya hemos probado el software de puertas a dentro y todo va bien, de repente el proyecto fracasa en casa del cliente o cuando estamos en fase de implementación y despliegue. Es muy importante tener un buen plan de despliegue y hacer una lista a modo de “checklist” para evitar desastres.

No Funcionales

- **Atributos de calidad**

Características, que, al cumplirse, mejoran en gran medida la calidad del software.

Confiabilidad: Estos requerimientos plantean que las aplicaciones no son perfectas, pero limitan las fallas de la aplicación a determinados valores.

Disponibilidad: Tiempo en que debe estar disponible la aplicación.

Seguridad: Medidas de seguridad con relación a procedimientos que impliquen el uso de información vulnerable como, por ejemplo, las claves de acceso al software.

Mantenibilidad: Facilidad de reparar un defecto en el software.

Portabilidad: El software debe funcionar en determinadas plataformas o bajo ciertas condiciones.

- **Restricciones**

Requerimientos que definen los límites y condicione de cómo una aplicación será diseñada o implementada.

Exactitud: Indica la exactitud con la que se deben prestar los servicios.

Restricciones de herramientas o lenguajes: Lenguajes y herramientas que se deben usar para el desarrollo y puesta en producción de las aplicaciones.

Restricciones de diseño: Son restricciones en el diseño del SW como la necesidad de seguir ciertos estándares.

- **Interfaces externas**

El SW a veces debe interoperar con otras aplicaciones del cliente. Las interfaces pueden ser de hardware, software o de comunicaciones.

- **Interfaces de usuario**

El diseño de las interfaces de usuario a veces se considera como una tarea en la fase de requerimientos. El diseño de interfaces en borrador (wireframes, mockups y prototipos) sirven para que el cliente pueda expresar de una mejor manera lo que quiere.