DESARROLLO

WEB FULL STACK

-INTERMEDIO

Autor de contenido

Andrés Fernando Pineda Guerra

EMPEZAR

Tabla de Contenido

Presentación

En el curso de desarrollador Full Stack como componente intermedio, podrán adquirir las habilidades y lenguajes necesarios para el desarrollo web, enfocándose en sus grandes pilares, como lo son Front End, Back End, Diseño y modelamiento de aplicaciones y documentación de código.
El curso trata temas emergentes tales como, la seguridad informática, desarrollo de aplicaciones móviles, gestión de base de datos, todo esto basado en la metodología Scrum. De la misma manera, se hace énfasis en el manejo de proyectos tanto en los módulos de desarrollo como los módulos de gestión de proyectos de TI.

Objetivos del curso (competencias)

Objetivo general

Formar a los participantes en el desarrollo web en todo el ciclo de vida del software, en donde adquieran los conocimientos básicos para implementar soluciones web.

Objetivo específico

- Conocer los conceptos y teoría básica del desarrollo web.

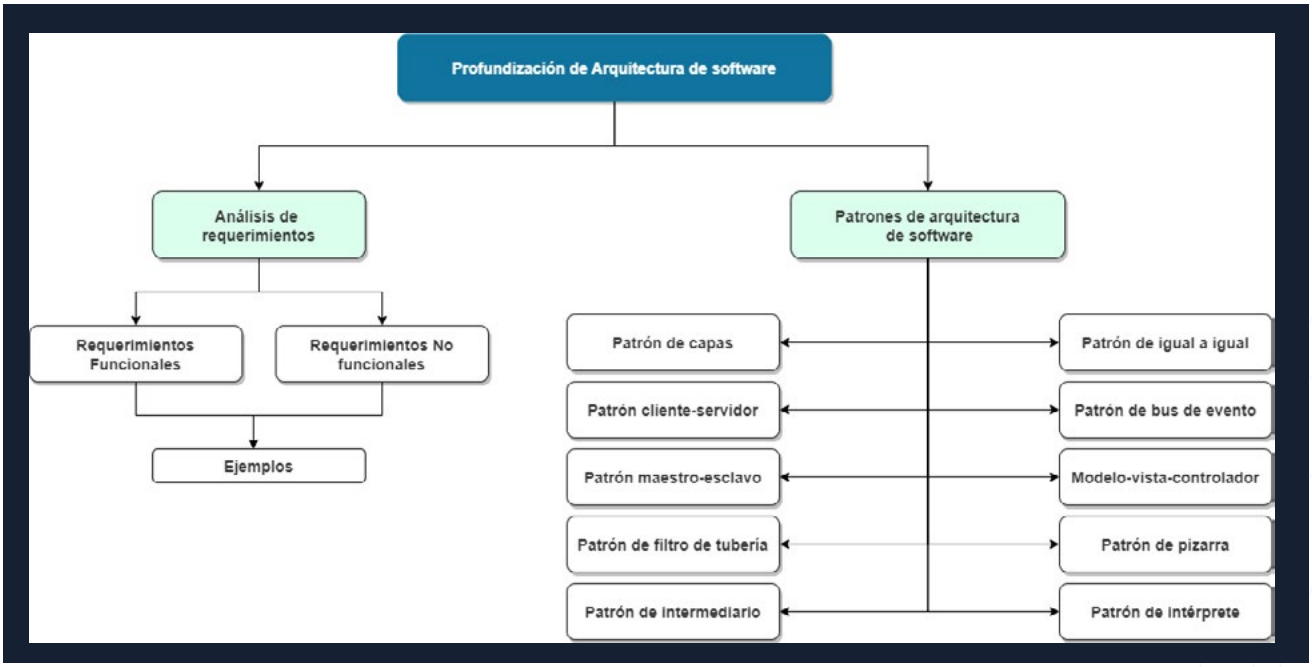
- Identificar y conocer los diferentes lenguajes de programación y herramientas para el desarrollo web.

- Aplicar las diferentes tecnologías web, tendencias y herramientas en el desarrollo de soluciones web enfocadas a proyectos.

- Diseñar, desarrollar e implementar soluciones web básicas en donde se integren los componentes de Front End, Back End, seguridad, redes y buenas prácticas utilizando metodologías ágiles.

- Identificar y conocer los conceptos básicos para el desarrollo móvil, así como aplicar su desarrollo en aplicaciones básicas.

Mapa de contenido de la unidad



Módulo 3

Profundización de Arquitectura de software

Ideas clave

Análisis de requerimientos, requerimientos funcionales, no funcionales, como identificarlos. Características de los requerimientos funcionales y no funcionales y ejemplos.

Patrones de arquitectura de software, patrón de capas, patrón cliente - servidor, patrón maestro esclavo, filtro de tubería, intermediario, igual a igual, bus de evento, modelo vista controlador, patrón de pizarra, patrón intérprete.

3.1. Análisis de requerimientos

Análisis de requerimientos.

Los requerimientos especifican qué es lo que el sistema debe hacer (sus funciones) y sus propiedades esenciales y deseables. La captura de los requerimientos tiene como objetivo principal la comprensión de lo que los clientes y los usuarios esperan que haga el sistema. Un requerimiento expresa el propósito del sistema sin considerar cómo se va a implementar. En otras palabras, los requerimientos identifican el qué del sistema, mientras que el diseño establece el cómo del sistema.

Es el conjunto de técnicas y procedimientos que nos permiten conocer los elementos necesarios para definir un proyecto de software. Es una tarea de ingeniería del software que permite especificar las características operacionales del software, indicar la interfaz del software con otros elementos del sistema y establecer las restricciones que debe cumplir el software.

La especificación de requerimientos suministra al técnico y al cliente, los medios para valorar el cumplimiento de resultados, procedimientos y datos, una vez que se haya construido.

La tarea de análisis de los requerimientos es un proceso de descubrimiento y refinamiento, el cliente y el desarrollador tienen un papel activo en la ingeniería de requerimientos de software. El cliente intenta plantear un sistema que en muchas ocasiones es confuso para él, sin embargo, es necesario que describa los datos, que especifique las funciones y el comportamiento del sistema que desea. El objetivo es que el desarrollador actúe como un negociador, un interrogador, un consultor, o sea, como persona que consulta y propone para resolver las necesidades del cliente.

El análisis de requerimientos proporciona una vía para que los clientes y los desarrolladores lleguen a un acuerdo sobre lo que debe hacer el sistema. La especificación, producto de este análisis proporciona las pautas a seguir a los diseñadores del sistema.

Pasos para a tener en cuenta para un análisis de requerimientos

- Realiza un estudio profundo de la necesidad tecnológica que tiene el negocio.
- Especifica las características operacionales que tendrá el software a desarrollar.
- Tiene en cuenta las diferentes áreas de trabajo: reconocimiento del problema, evaluación, modelado, especificación y revisión.
- Realiza a través de entrevistas, talleres, observación, indagación, revisión documental y demás técnicas específicas.

- Describe el plan del proyecto a seguir.
- Es fundamental entregar el proyecto dentro del tiempo y presupuesto acordados y de los objetivos de negocio.

Características de los requerimientos.

Los requerimientos permiten que los desarrolladores expliquen cómo han entendido lo que el cliente pretende del sistema. También, indican a los diseñadores qué funcionalidad y que características va a tener el sistema resultante. Y además, indican al equipo de pruebas qué demostraciones llevar a cabo para convencer al cliente de que el sistema que se le entrega es lo que solicitó.

Deben ser correctos.
Tanto el cliente como el desarrollador deben revisarlos para asegurar que no tienen errores.

Deben ser consistentes.
Dos requerimientos son inconsistentes cuando es imposible satisfacerlos simultáneamente.

Deben estar completos.
El conjunto de requerimientos está completo si todos los estados posibles, cambios de estado, entradas, productos y restricciones están descritos en alguno de los requerimientos.

Deben ser realistas.
Todos los requerimientos deben ser revisados para asegurar que son posibles.

Deben ser verificables.
Se deben poder preparar pruebas que demuestren que se han cumplido los requerimientos.

Requerimientos funcionales.

Un requisito funcional es una declaración de cómo debe comportarse un sistema. Define lo que el sistema debe hacer para satisfacer las necesidades o expectativas del usuario. Los requisitos funcionales se pueden considerar como características que el usuario detecta. Son diferentes de los requisitos no funcionales, que definen cómo debe funcionar internamente el sistema (p. ej., rendimiento, seguridad, etc.).

Los requisitos funcionales se componen de dos partes: función y comportamiento. La función es lo que hace el sistema (por ejemplo, “calcular la cantidad de usuarios recurrentes”). El comportamiento es cómo lo hace el sistema (p. ej., “El sistema debe calcular la cantidad de usuarios recurrentes mediante plugins de control”).

Tipos de requisitos funcionales

Estos son los tipos de requisitos funcionales más comunes:

- Regulaciones comerciales
- Requisitos de Certificación
- Los requisitos de información
- Funciones administrativas
- Niveles de autorización
- Seguimiento de auditoría
- Interfaces externas
- Administración de datos
- Requisitos legales y reglamentarios

Ejemplos:

Fecha de creación	**07/07/2021**	**Código del requerimiento**	**RF001**
Nombre	Acceso al centro de solicitudes		
Responsables	Ing. Andrés Pineda		
Descripción	El sistema debe permitir el acceso al centro de solicitudes a los usuarios del sitio web de RITA de manera sencilla.		
Justificación	El sistema OSTICKETS o centro de solicitudes es uno de los aspectos más importantes del sitio web de RITA, ya que por este medio se realizan todas las solicitudes.		
Nivel de importancia	5 - ALTO		

Fecha de creación	**07/07/2021**	**Código del requerimiento**	**RF003**
Nombre	Acceso a redes sociales		
Responsables	Ing. Andrés Pineda		
Descripción	El sistema debe permitir acceder a las redes sociales de RITA, tales como facebook, twitter, instagram, LinkedIn y youtube.		
Justificación	Es importante brindar un fácil acceso a las redes sociales, para generar más tráfico.		
Nivel de importancia	5 - ALTO.		

Requerimientos No Funcionales.

Se trata de requisitos que no se refieren directamente a las funciones específicas suministradas por el sistema (características de usuario), sino a las propiedades del sistema: rendimiento, seguridad, disponibilidad. En palabras más sencillas, no hablan de "lo que" hace el sistema, sino de "cómo" lo hace. Alternativamente, definen restricciones del sistema tales como la capacidad de los dispositivos de entrada/salida y la representación de los datos utilizados en la interfaz del sistema.

Los requisitos no funcionales se originan en la necesidad del usuario, debido a restricciones presupuestarias, políticas organizacionales, la necesidad de interoperabilidad con otros sistemas de software o hardware, o factores externos tales como regulaciones de seguridad, políticas de privacidad, entre otros.

Existen diferentes tipos de requisitos y se clasifican según sus implicaciones.

- Requisitos del producto. Especifican el comportamiento del producto, como los requisitos de rendimiento sobre la velocidad de ejecución del sistema y la cantidad de memoria necesaria, los requisitos de fiabilidad que establecen la tasa de fallos para que el sistema sea aceptable, los requisitos de portabilidad y los requisitos de usabilidad.

- Requisitos organizativos. Se derivan de las políticas y procedimientos existentes en la organización cliente y en la organización del desarrollador: estándares en los procesos a utilizar; requisitos de implementación tales como lenguajes de programación o el método de diseño a utilizar; y requisitos de entrega que especifican cuándo se entregará el producto y su documentación.

- Necesidades externas. Se derivan de factores externos al sistema y a su proceso de desarrollo. Incluyen los requisitos de interoperabilidad que definen la forma en que el sistema interactúa con los demás sistemas de la organización; los requisitos legales que deben seguirse para garantizar que el sistema funciona dentro de la ley; y los requisitos éticos. Estos últimos se imponen al sistema para asegurar que será aceptado por el usuario.

Ejemplos:

- RNF001 - El sitio web debe tener énfasis en investigación.
- RNF002 - El sitio web debe tener los principios de diseño, en donde se base en Colores institucionales.
- RNF003 - Protocolos de seguridad HTTPS.

3.2. Patrones de arquitectura de software

Patrones de arquitectura de software
Patrón de capas

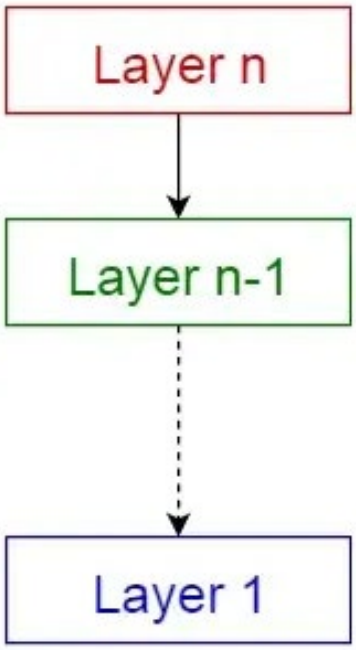
Los patrones de arquitectura en capas son patrones de n niveles donde los componentes están organizados en capas horizontales. Este es el método tradicional para diseñar la mayoría de los programas informáticos y está destinado a ser auto-independiente. Esto significa que todos los componentes están interconectados pero no dependen unos de otros. Cada capa del patrón de arquitectura en capas tiene un papel y una responsabilidad específicos dentro de la aplicación. Por ejemplo, una capa de presentación se encargaría de manejar toda la interfaz de usuario y la lógica de comunicación del navegador, mientras que una capa empresarial se encargaría de ejecutar las reglas empresariales específicas asociadas a la solicitud.

Una de las características poderosas del patrón de arquitectura en capas es la separación de las preocupaciones entre los componentes. Los componentes dentro de una capa específica se ocupan sólo de la lógica que pertenece a esa capa.

Ventajas del modelo de software basado en capa

- Alta compresibilidad porque los componentes pertenecen a capas específicas de la arquitectura, otras capas pueden ser burladas o desviadas, haciendo que este patrón sea relativamente fácil de comprobar..

- Alta facilidad de desarrollo porque este patrón es muy conocido y no es excesivamente complejo de implementar, además la mayoría de las empresas desarrollan aplicaciones separando conjuntos de habilidades por capas, este patrón se convierte en una elección natural para la mayoría de los desarrollos de aplicaciones empresariales.
- Mantenible.
- Fácil de asignar «roles» separados.
- Fácil de actualizar y mejorar las capas por separado.

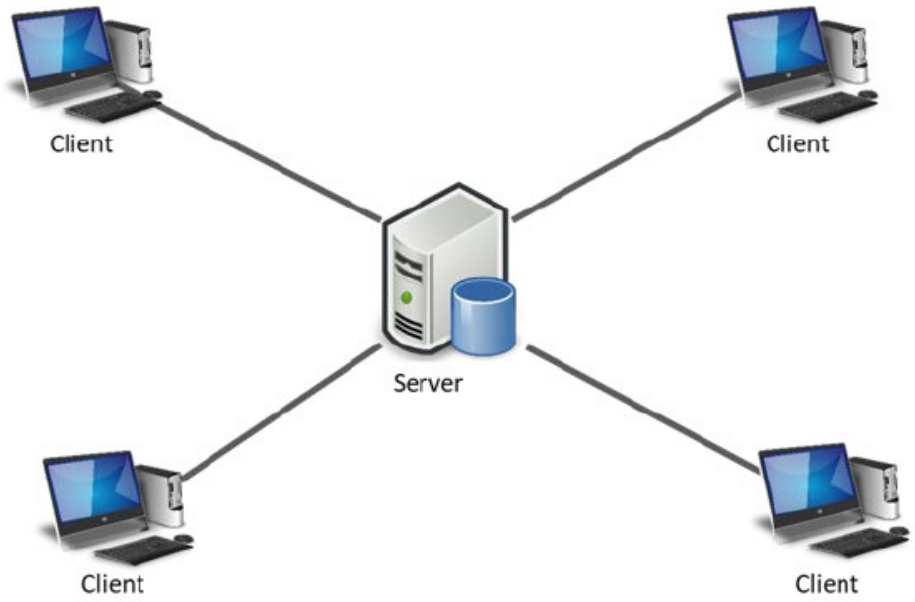


- Alta facilidad de desarrollo porque este patrón es muy conocido y no es excesivamente complejo de implementar, además la mayoría de las empresas desarrollan aplicaciones separando conjuntos de habilidades por capas, este patrón se convierte en una elección natural para la mayoría de los desarrollos de aplicaciones empresariales.
- Mantenible.
- Fácil de asignar «roles» separados.
- Fácil de actualizar y mejorar las capas por separado.

Patrón cliente-servidor

Cliente-Servidor es uno de los estilos arquitectónicos distribuidos más conocidos, el cual está compuesto por dos componentes, el proveedor y el consumidor. El proveedor es un servidor que brinda una serie de servicios o recursos los cuales son consumido por el Cliente.

En una arquitectura Cliente-Servidor existe un servidor y múltiples clientes que se conectan al servidor para recuperar todos los recursos necesarios para funcionar, en este sentido, el cliente solo es una capa para representar los datos y se detonan acciones para modificar el estado del servidor, mientras que el servidor es el que hace todo el trabajo pesado.

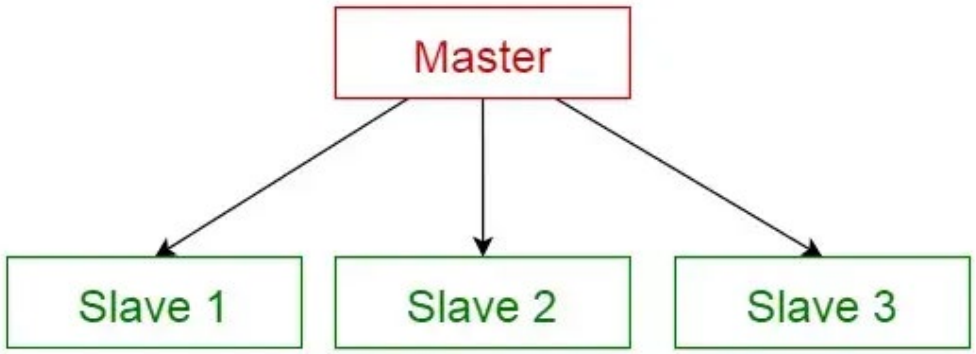


Patrón maestro-esclavo

Este patrón consiste en dos partes; maestro y esclavos . El componente maestro distribuye el trabajo entre componentes esclavos idénticos y calcula el resultado final de los resultados que devuelven los esclavos.

Uso

- En la replicación de la base de datos, la base de datos maestra se considera como la fuente autorizada y las bases de datos esclavas se sincronizan con ella.
- Periféricos conectados a un bus en un sistema informático (unidades maestra y esclava).

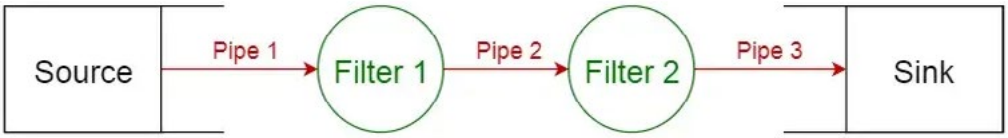


Patrón de filtro de tubería

Se utiliza, sobre todo, para la estructura de sistemas que producen y procesan una secuencia de datos. En la ingeniería de software, el filtro de tubería se aplica cuando los datos de entrada deben transformarse en datos de salida a través de componentes para el cálculo. Los componentes reciben el nombre de 'filtros' conectados entre sí por 'tuberías' que transmiten los datos.

Uso

- Los filtros consecutivos realizan análisis léxico, análisis sintáctico y generación de código.
- Flujos de trabajo en bioinformática.



Patrón de intermediario

Es usado para estructurar sistemas distribuidos con componentes desacoplados (pueden interactuar entre sí). El responsable de coordinar la comunicación entre los componentes es el intermediario. Podemos encontrarlo en software de Message Broker, como la plataforma de software Apache ActiveMQ, por ejemplo.

Este patrón se usa para estructurar sistemas distribuidos con componentes desacoplados. Estos componentes pueden interactuar entre sí mediante invocaciones de servicios remotos. Un componente de intermediario es responsable de la coordinación de la comunicación entre los componentes .

Los servidores publican sus capacidades (servicios y características) a un intermediario. Los clientes solicitan un servicio del intermediario y el intermediario redirecciona al cliente a un servicio adecuado desde su registro.

Uso

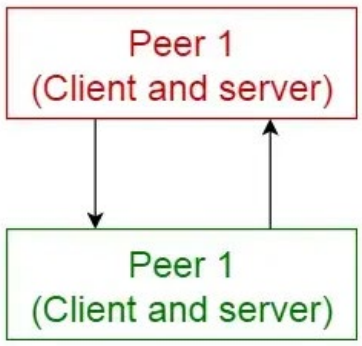
- Software de Message Broker como Apache ActiveMQ , Apache Kafka , RabbitMQ y JBoss Messaging .

Patrón de igual a igual

En este patrón, los componentes individuales se conocen como pares . Los pares pueden funcionar tanto como un cliente , solicitando servicios de otros pares, y como un servidor , proporcionando servicios a otros pares. Un par puede actuar como un cliente o como un servidor o como ambos, y puede cambiar su rol dinámicamente con el tiempo.

Uso

- Redes de intercambio de archivos como Gnutella y G2)
- Protocolos multimedia como P2PTV y PDTP .



Patrón de bus de evento

Este patrón trata principalmente con eventos y tiene 4 componentes principales; fuente de evento , escucha de evento , canal y bus de evento . Las fuentes publican mensajes en canales particulares en un bus de eventos. Los oyentes se suscriben a canales particulares. Los oyentes son notificados de los mensajes que se publican en un canal al que se han suscrito anteriormente.

Modelo-vista-controlador

Es el conocido MVC, que divide una aplicación interactiva en tres partes (modelo, vista, controlador) encargadas de contener la funcionalidad, mostrar la información al usuario y manejar su entrada. Este patrón de arquitectura de software separa los datos y la lógica de negocio de una aplicación de su representación.

- modelo — contiene la funcionalidad y los datos básicos
- vista : muestra la información al usuario (se puede definir más de una vista)
- controlador : maneja la entrada del usuario

Patrón de pizarra

Sus principales elementos son: la pizarra (memoria global estructurada), fuente de conocimiento (módulos especializados) y componente de control (encargado de seleccionar y ejecutar los módulos). Suele utilizarse para el reconocimiento de voz, identificaciones, seguimientos, etc.

Componentes

- pizarra : una memoria global estructurada que contiene objetos del espacio de solución
- fuente de conocimiento : módulos especializados con su propia representación
- componente de control : selecciona, configura y ejecuta módulos.

Patrón de intérprete

Es usado para el diseño de un componente que interpreta programas escritos en un lenguaje y define cómo hacer la evaluación de las líneas de programas. “La idea básica es tener una clase para cada símbolo del idioma”.

Uso

- Lenguajes de consulta de base de datos como SQL.
- Idiomas utilizados para describir los protocolos de comunicación.

Otros materiales para profundizar

Recursos de video

HolaMundo (Director). (2021, junio 18). Los 6 patrones de diseño más utilizados. https://www.youtube.com/watch?v=JI_THVXPToQ

Ingeniería de Software de Élite (Director). (2020, septiembre 15). Cómo deleitar a tus usuarios—Ingeniería de Requerimientos, La Serie. https://www.youtube.com/watch?v=LPM1ehPDpSc

Referencias bibliográficas de la unidad

Requerimientos de Software—ProQuest. (s. f.). Recuperado 3 de enero de 2023, de https://www.proquest.com/openview/5dd30d795aa48c298b-f4352792a93059/1?pq-origsite=gscholar&cbl=2027443

Sommerville, I. (2005). Ingeniería del software. Pearson Educación.

ALCALDÍA MAYOR
DE BOGOTÁ D.C.
SECRETARÍA DE EDUCACIÓN

ATENEA
AGENCIA DISTRITAL PARA LA EDUCACIÓN
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA

BOGOTÁ

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad