

1ºPE Computing & Information technology

Curso 2023-2024



Programming

Entrega Final

Samantha Fernández Peralta

samanthaangelesfp@msmk.university



*The University of
Science & Technology*

Nombre Alumno / DNI	Samantha Fernández /54716803x
Título del Programa	Computing & Information technology
Nº Unidad y Título	UNIT 4/ Programming
Año académico	1ºPE
Profesor de la unidad	Gabriela García
Título del Assignment	Programming
Día de emisión	20-01-2024
Día de entrega	23-01-2024
Nombre IV y fecha	
Declaración del estudiante	<p>Certifico que la presentación del assignment es completamente mi propio trabajo y entiendo completamente las consecuencias del plagio. Entiendo que hacer una declaración falsa es una forma de mala práctica.</p> <p>Fecha: 23-01-2024</p> <p>Firma del alumno: Samantha Fernández</p>



Informe sobre Lenguajes de Programación

Introducción

Los lenguajes, paradigmas y estándares en el mundo de la programación desempeñan un papel esencial al proporcionar las directrices y las herramientas imprescindibles para la expresión de ideas y la creación de software de manera eficaz y coherente.

Establecen una base común que facilita la colaboración, la comprensión y la compatibilidad en el mundo de la programación.

Tipos de Lenguajes de Programación

Los lenguajes de programación se dividen en tres categorías principales según su nivel de abstracción:

Lenguajes de Bajo Nivel:

Descripción: lenguajes cercanos al lenguaje de máquina, permiten un control preciso del hardware. Ofrecen un control excepcional, pero requieren un profundo conocimiento de la máquina.

Ejemplos: lenguaje ensamblador, C en su capacidad de bajo nivel.

Usos comunes: programación de sistemas operativos, control de dispositivos y aplicaciones de alto rendimiento donde la eficiencia es crítica.

Lenguajes de Medio Nivel:

Descripción: Estos lenguajes equilibran la eficiencia y la abstracción, ofreciendo un control considerable sin sacrificar la productividad. Son ideales para aplicaciones que requieren rendimiento y facilidad de desarrollo.

Ejemplos: C, C++, Rust.

Usos comunes: Desarrollo de software de escritorio, aplicaciones de juegos, así como aplicaciones científicas e ingeniería.

Lenguajes de Alto Nivel:

Descripción: lenguajes más cercanos al lenguaje humano, facilitando la escritura de código.

Ejemplos: Python, JavaScript, C#.

Usos comunes: desarrollo de aplicaciones web, aplicaciones móviles, análisis de datos, automatización, investigaciones científicas y escritura de scripts.

Lenguajes compilados e interpretados

Los lenguajes de programación se pueden clasificar en dos categorías principales: compilados e interpretados.

Un lenguaje compilado: es aquel cuyo código fuente se traduce completamente a un lenguaje de bajo nivel, específico de la plataforma de destino, antes de su ejecución.

El proceso de compilación implica tomar el código fuente escrito por el programador y convertirlo en un archivo ejecutable o en un módulo que la máquina puede comprender directamente. Algunos ejemplos de lenguajes compilados son C, C++ y Go.

Un **lenguaje interpretado** no se traduce completamente antes de su ejecución.

En lugar de eso, el programa interpreta y ejecuta el código fuente línea por línea o instrucción por instrucción en tiempo real.

Los lenguajes interpretados generalmente requieren un intérprete o una máquina virtual para ejecutar el código. Algunos ejemplos de lenguajes interpretados son Python, JavaScript, Ruby y PHP.

Lenguajes compilados vs interpretados

La principal diferencia entre los lenguajes compilados e interpretados radica en el proceso de ejecución.

En un lenguaje compilado, el código fuente se traduce de antemano y se ejecuta directamente por la máquina, lo que generalmente resulta en una ejecución más rápida. En un lenguaje interpretado, el código fuente se ejecuta línea por línea o instrucción por instrucción por un intérprete, lo que puede ser más flexible pero también puede ser más lento en términos de rendimiento.

Fuertemente tipado vs. Débilmente tipado

La tipificación se refiere a cómo los lenguajes de programación tratan los tipos de datos. Un lenguaje puede ser fuertemente tipado o débilmente tipado.

En un lenguaje fuertemente tipado: los tipos de datos son estrictamente aplicados y verificados en tiempo de compilación o en tiempo de ejecución.

Esto significa que el tipo de una variable debe ser declarado y que las operaciones entre diferentes tipos de datos pueden estar restringidas o requerir conversiones explícitas. Algunos ejemplos de lenguajes fuertemente tipados son C++, Java y C#.

En lenguaje débilmente tipado: las conversiones y operaciones entre diferentes tipos de datos pueden ser implícitas o realizadas automáticamente por el lenguaje.

Las variables pueden cambiar su tipo dinámicamente durante la ejecución del programa. Algunos ejemplos de lenguajes débilmente tipados son JavaScript y PHP.

La ventaja de un lenguaje fuertemente tipado es que proporciona una mayor seguridad y detección de errores en tiempo de compilación.

Las restricciones de tipo ayudan a prevenir errores comunes y a garantizar la coherencia de los datos.

Por otra parte, un lenguaje débilmente tipado puede ser más flexible y conveniente, ya que permite realizar conversiones y operaciones sin la necesidad de conversiones explícitas.

Algunos lenguajes pueden tener características mixtas o permitir diferentes grados de flexibilidad en la manipulación de los tipos de datos.

Lenguajes de tipado estático vs dinámico

Esta clasificación se refiere a si los tipos de datos son verificados en tiempo de compilación o en tiempo de ejecución.

Los lenguajes de tipado estático realizan la verificación en tiempo de compilación, mientras que los lenguajes de tipado dinámico realizan la verificación en tiempo de ejecución.

C y Java son ejemplos de lenguajes de tipado estático, mientras que Python y JavaScript son ejemplos de lenguajes de tipado dinámico.

Paradigmas de Programación

Los paradigmas de programación son enfoques o estilos de escribir código. Algunos de los principales son:

Paradigma Imperativo:

Descripción: el código se organiza en una secuencia de instrucciones.

Ejemplos: C, Fortran, Pascal.

Características: Utiliza variables y estructuras de control como bucles y condicionales.

Paradigma Declarativo:

Descripción: se enfoca en describir el resultado deseado en lugar de los pasos para llegar a él. Se utilizan declaraciones y expresiones para especificar lo que se quiere, y no cómo lograrlo.

Ejemplos: SQL, HTML.

Características: se centra en la descripción de datos y relaciones.

Paradigma Orientado a Objetos:

Descripción: El programa organiza el código en torno a objetos que contienen datos y métodos. Se centra en la encapsulación, la herencia y el polimorfismo.

Ejemplos: Java, C++, Python.

Características: Abstracción de datos, reutilización de código a través de la herencia y el polimorfismo.

Paradigma Funcional:

Descripción: se basa en funciones que toman entradas y producen salidas. Se enfoca en la inmutabilidad de datos y evita los efectos secundarios.

Ejemplos: Haskell, Lisp.

Características: funciones puras, inmutabilidad, evaluación perezosa y recursión.

Paradigma Lógico:

Descripción: Se basa en reglas lógicas y relaciones entre datos. El programa busca soluciones lógicas en lugar de instrucciones secuenciales.

Ejemplos: Prolog.

Características: Programación basada en reglas y hechos lógicos.

Estándares de Programación

Los estándares de programación son pautas y reglas que establecen la forma en que se debe escribir código para garantizar la consistencia, calidad y eficiencia en el desarrollo de software.

Esto facilita la colaboración en equipo, mejora la legibilidad del código y asegura la interoperabilidad con otros sistemas y aplicaciones.

Seguir estándares puede ayudar a prevenir errores comunes y vulnerabilidades de seguridad.

Algunos estándares populares son:

Estándares de Codificación:

Estos estándares influyen en la presentación y estilo del código fuente, definiendo normativas para la nomenclatura de variables, la indentación, la longitud de líneas y otros aspectos visuales del código.

Un ejemplo es PEP 8, utilizado en Python, que promueve la uniformidad y legibilidad del código.

Estándares de Seguridad:

Establecen prácticas y pautas para proteger el software contra vulnerabilidades, estos estándares ofrecen prácticas para validar entradas, gestionar contraseñas y garantizar la autenticación.

Un ejemplo notable es el OWASP Top Ten, que identifica y aborda las diez principales vulnerabilidades en aplicaciones web.

Estándares de Documentación:

Se centra en la documentación del código, estos estándares incluyen prácticas para comentarios, descripciones de funciones y documentación de API.

Un ejemplo muy utilizado es Javadoc, utilizado en Java, que proporciona una estructura estándar para documentar clases y métodos.

Estándares de Pruebas y Calidad:

Estos estándares definen reglas para realizar pruebas de software, gestionar errores y mantener altos estándares de calidad.

Un ejemplo es la certificación ISTQB (International Software Testing Qualifications Board), que establece estándares para profesionales de pruebas de software.

Estándares de Accesibilidad:

Estos estándares garantizan que el software sea accesible para personas con discapacidades.

Esto puede incluir pautas sobre etiquetas alt en imágenes, semántica en HTML o WCAG (Web Content Accessibility Guidelines), que proporciona pautas para hacer que los contenidos web sean más accesibles.

Beneficios de Adherirse a Estándares

La adhesión a estándares de codificación **mejora la consistencia y legibilidad** del código, facilitando su comprensión y mantenimiento a lo largo del tiempo.

Los estándares proporcionan reglas comunes que **facilitan la colaboración en equipo**. Los desarrolladores pueden entender y trabajar en el código de manera más eficiente.

Seguir estándares promueve la **interoperabilidad**, permitiendo que diferentes componentes y sistemas se integren sin problemas.

Los estándares de seguridad establecen prácticas para proteger el software contra vulnerabilidades, **reduciendo el riesgo de ataques y mejorando la seguridad** global de la aplicación.

La adherencia a estándares de documentación y codificación **agiliza el desarrollo** al proporcionar estructuras y formatos predefinidos.

Consecuencias de No Adherirse a Estándares:

La falta de estándares puede resultar en **código confuso y difícil de mantener**, lo que lleva a un aumento en los costos de desarrollo y corrección de errores.

Sin estándares, **la colaboración en equipo puede volverse caótica**. La falta de consistencia en el código puede generar malentendidos y conflictos entre los miembros del equipo.

La falta de adherencia a estándares **puede provocar incompatibilidades entre diferentes partes del software**, dificultando la integración de componentes y sistemas.

La ausencia de estándares de seguridad **puede dejar el software vulnerable** a ataques, comprometiendo la integridad y privacidad de los datos.

La falta de eficiencia causada por la falta de estándares puede resultar en **retrasos en el desarrollo y en la entrega de proyectos**.

Puede causar **costos adicionales**, ya que corregir problemas derivados de la falta de estándares puede aumentar los costos de desarrollo y mantenimiento a largo plazo.

Conclusión

En resumen: la comprensión profunda de los diversos lenguajes, paradigmas y estándares en el desarrollo de software no solo es esencial, sino que también es un catalizador clave para la eficiencia, la calidad y la innovación.

Estos elementos no solo son herramientas técnicas, sino también fundamentos que influyen en la manera en que concebimos, creamos y mantenemos el software.

La habilidad de navegar este paisaje diverso capacita a los desarrolladores para tomar decisiones informadas, adoptar enfoques más eficientes y contribuir a la evolución constante del arte de programar.

En última instancia, la comprensión de estos elementos no solo mejora la ejecución técnica, sino que también enriquece la creatividad y la adaptabilidad en el siempre cambiante mundo del desarrollo de software.

Referencias:

Tipos de lenguaje de programación:

- Assembler Institute (s.f.). "Tipos de lenguaje de programación." [En línea]. Disponible en: <https://assemblerinstitute.com/blog/tipos-lenguaje-programacion/> (Consultado el 15-12-2023).

Lenguaje de Programación: Conceptos Básicos y Tipos:

- GoDaddy (s.f.). "Lenguaje de Programación: Conceptos Básicos y Tipos." [En línea]. Disponible en: <https://mx.godaddy.com/blog/lenguaje-programacion-conceptos-basicos-y-tipos/> (Consultado el 15-12-2023).

Estándares de Calidad en el Software:

- The Dojo MX Blog (2021). "Estándares de Calidad en el Software." [En línea]. Disponible en: <https://blog.thedojo.mx/2021/10/05/estandares-de-calidad-en-el-software.html> (Consultado el 15-12-2023).

Paradigmas de Programación:

- Platzi Blog (s.f.). "Paradigmas de Programación." [En línea]. Disponible en: <https://platzi.com/blog/paradigmas-programacion/> (Consultado el 15-12-2023).

¿Qué son los Paradigmas de Programación, Cómo Trabajan y Para Qué Sirven?:

- LOV Technology (s.f.). "¿Qué son los Paradigmas de Programación, Cómo Trabajan y Para Qué Sirven?" [En línea]. Disponible en: <https://lovtechnology.com/que-son-los-paradigmas-de-programacion-como-trabajan-y-para-que-sirven/> (Consultado el 15-12-2023).

Lenguajes Compilados vs Interpretados:

- freeCodeCamp Español (s.f.). "Lenguajes Compilados vs Interpretados." [En línea]. Disponible en: <https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/> (Consultado el 15-12-2023).

Informe de Testing y Pruebas de Código

Introducción

El testing y las pruebas de código son cruciales en el ciclo de vida del desarrollo de software. Son esenciales para identificar errores, garantizar la calidad del software y mejorar la confiabilidad de las aplicaciones.

En este informe, se verán los fundamentos básicos del testing y las pruebas de código, los diferentes tipos de pruebas, las técnicas empleadas, la importancia de la automatización de pruebas y la importancia que tienen en el desarrollo de software.

Conceptos Básicos

El testing se refiere al proceso de evaluar un sistema o componente buscando diferencias entre los resultados deseados y los resultados reales.

Las pruebas de código consisten en verificar el comportamiento y funcionalidad de secciones específicas de código.

Mientras el testing implica evaluar el sistema en su conjunto, las pruebas de código se centran en fragmentos específicos.

El objetivo principal de realizar pruebas es garantizar la calidad del software y reducir los errores en producción.

Sus beneficios son:

- El poder detectar errores y fallos en el software.

- Mejora la calidad del software

- Ahorrar tiempo y costos, ya que detecta los errores y problemas en etapas tempranas del desarrollo.

Tipos de Pruebas

Algunas de las pruebas más comunes son:

Pruebas unitarias: se enfocan en probar componentes individuales del software, como funciones o métodos. Estas pruebas se realizan a nivel de código y suelen ser automatizadas utilizando frameworks como JUnit (para Java) o pytest (para Python).

Pruebas de integración: verifican que la interacción entre los diferentes componentes de un sistema sea el correcto. Evalúan cómo se combinan los módulos y cómo se comunican entre sí. Algunas herramientas asociadas son: Selenium y Apache JMeter.

Pruebas de sistema: se ejecutan en el sistema completo para evaluar su cumplimiento de los requisitos y especificaciones. Pueden incluir pruebas funcionales, pruebas de rendimiento y pruebas de seguridad, entre otras. Algunas herramientas asociadas son: Appium, TestComplete.

Pruebas de aceptación: verifican si el software cumple con los criterios de aceptación definidos por el cliente o usuario final. Se enfocan en validar que el sistema se ajuste a sus necesidades. Algunas herramientas asociadas son: Cucumber y Robot Framework.

Pruebas de carga: evalúan el rendimiento del sistema bajo cargas de trabajo específicas, así como un alto número de usuarios concurrentes. Ayudan a identificar cuellos de botella y problemas de rendimiento. Algunas herramientas asociadas son: Gatling y LoadRunner.

Pruebas de estrés: prueban los límites del sistema, sometiéndolo a condiciones extremas para evaluar su estabilidad y respuesta. Algunas herramientas asociadas son: Apache JMeter y Siege.

Técnicas de Testing

Existen diversas técnicas de testing que se utilizan para mejorar la calidad y efectividad de las pruebas. Las más comunes son:

Test Driven Development (TDD): esta técnica implica escribir las pruebas antes que el código de producción, lo que mejora la calidad del software y mantiene un enfoque centrado en los requisitos.

Behavior Driven Development (BDD): se centra en la colaboración entre desarrolladores, testers y partes interesadas para crear pruebas que se basen en comportamientos y requisitos del sistema.

Pruebas de regresión: detectan cualquier regresión o cambio no deseado en el software después de realizar modificaciones o adiciones al código existente.

Pruebas exploratorias: exploran el software de manera apropiada, sin seguir un plan predefinido. Permite descubrir errores y comportamientos inesperados mediante la interacción directa con la aplicación.

Análisis estático: se basa en revisar el código fuente o la documentación sin ejecutar el programa. Se utilizan para identificar posibles problemas.

Beneficios y retos asociados a cada técnica

Los beneficios de **TDD** incluyen una mayor calidad del código, una cobertura de pruebas más completa y una mejor comprensión de los requisitos por parte del equipo de desarrollo.

Como reto, puede haber un tiempo adicional necesario para escribir las pruebas antes de implementar la funcionalidad.

El **BDD** fomenta la colaboración y la comunicación entre todos los miembros del equipo, lo que conduce a una mejor comprensión de los requisitos y una mayor calidad del software.

Como reto, puede requerir una inversión inicial de tiempo y esfuerzo para definir los escenarios de prueba en un lenguaje natural comprensible para todas las partes interesadas.

Las pruebas de regresión, ayudan a garantizar que las modificaciones o adiciones al código no introduzcan errores en áreas previamente funcionales.

Como reto, tiene como desafío el mantener una suite de pruebas de regresión actualizada y ejecutarlas de manera eficiente para evitar retrasos en el desarrollo.

Las pruebas exploratorias, permiten descubrir problemas y comportamientos inesperados que no se pueden planificar de antemano.

Como reto, la cobertura de pruebas puede ser limitada y depende en gran medida de la experiencia y habilidades del tester.

Automatización de Pruebas

La automatización de pruebas implica el uso de herramientas y frameworks para ejecutar pruebas de manera eficiente y repetitiva.

Algunos de sus beneficios son:

Se **ahorran tiempo y recursos**, ya que las pruebas automatizadas se ejecutan más rápido y requieren menos recursos humanos en comparación con las pruebas manuales.

Permite **aumentar la cobertura de las pruebas**, se pueden ejecutar un conjunto más amplio de pruebas en menos tiempo.

Mejora de la calidad del software, ya que se reducen los errores humanos y se mejora la precisión de las pruebas.

Algunas herramientas y frameworks populares para la automatización de pruebas son:

Selenium: permite la grabación y reproducción de acciones del usuario y es compatible con varios lenguajes de programación.

Appium: se utiliza para la automatización de pruebas de aplicaciones móviles en plataformas Android e iOS y permite la escritura de pruebas en lenguajes como Java, C#, Python, entre otros.

Cucumber: es una herramienta de BDD que permite escribir pruebas en un lenguaje natural comprensible por todas las partes interesadas, es compatible con varios lenguajes de programación y se puede integrar con otras herramientas de automatización.

JUnit: es un framework de pruebas unitarias para Java. Proporciona una estructura para escribir y ejecutar pruebas de manera eficiente.

Casos de Uso y Ejemplos

Algunos ejemplos prácticos y casos de uso donde se aplican distintos tipos de pruebas y técnicas de testing en proyectos reales son:

En un **proyecto de desarrollo de software**, se pueden realizar **pruebas unitarias** para validar la funcionalidad de métodos y funciones individuales.

Por ejemplo, se pueden escribir pruebas unitarias para verificar el correcto cálculo de impuestos en una aplicación de contabilidad.

En un **sistema de comercio electrónico**, se pueden realizar **pruebas de integración** para garantizar que los diferentes componentes, como el carrito de compras, el pago y la gestión de inventario, funcionen correctamente juntos.

En un **proyecto de desarrollo de un sistema de gestión de inventario**, se pueden realizar **pruebas de sistema** para verificar que todas las funcionalidades del sistema, como la creación de pedidos, la actualización de inventario y la generación de informes, funcionen correctamente y cumplan con los requisitos establecidos.

Conclusión

Las pruebas y el testing son componentes fundamentales en el desarrollo de software, ya que garantizan la calidad y confiabilidad del producto final.

Gracias a las pruebas, se pueden identificar y corregir errores antes de que el software esté en manos de los usuarios, lo que ayuda a evitar problemas y vulnerabilidades que podrían afectar negativamente la experiencia del usuario y la reputación de la empresa.

Gracias a el testing, se puede validar que el software cumple con los requisitos y funcionalidades esperadas, mejorando así la satisfacción del cliente.

Referencias

Tipos de Testing en Desarrollo de Software:

- Programación y Más (s.f.). "Tipos de Testing en Desarrollo de Software." [En línea]. Disponible en: <https://programacionymas.com/blog/tipos-de-testing-en-desarrollo-de-software> (Consultado el 18-01-2024).

Fundamentos de Testing: Preguntas y Respuestas:

- Campus MVP (s.f.). "Fundamentos de Testing: Preguntas y Respuestas." [En línea]. Disponible en: <https://www.campusmvp.es/recursos/post/fundamentos-de-testing-preguntas-y-respuestas.aspx> (Consultado el 18-01-2024).

Software Testing:

- Tecnologías de la Información (s.f.). "Software Testing." [En línea]. Disponible en: <https://www.tecnologias-informacion.com/software-testing.html> (Consultado el 18-01-2024).
- Atlassian (s.f.). "Automated Testing." [En línea]. Disponible en: <https://www.atlassian.com/es/continuous-delivery/software-testing/automated-testing> (Consultado el 18-01-2024).

Informe sobre frameworks

Introducción

Los frameworks son un conjunto de herramientas y bibliotecas de software que proporcionan una estructura y funcionalidades predefinidas para el desarrollo de aplicaciones.

En el caso de una tienda online, los frameworks pueden ser de gran utilidad, ya que agilizan el proceso de desarrollo y proporcionan características específicas para este tipo de sitios web.

En este informe, explicaremos qué son los frameworks y mencionaremos algunos de los más utilizados y que más aplicarían a una tienda online.

¿Qué son los frameworks?

Un framework es una infraestructura de software que proporciona una base sobre la cual se pueden construir y desarrollar aplicaciones. Estos frameworks ofrecen una serie de herramientas, bibliotecas y patrones de diseño predefinidos, lo que permite a los desarrolladores ahorrar tiempo y esfuerzo al no tener que desarrollar desde cero todas las funcionalidades comunes de una aplicación.

Los frameworks también promueven una estructura organizada y modular, lo que facilita el mantenimiento y la escalabilidad del proyecto.

Frameworks para una tienda online

Algunos de los frameworks más populares que se pueden utilizar para desarrollar una tienda online son:

WooCommerce: es un plugin de WordPress que brinda todas las funcionalidades necesarias para construir una tienda online. Ofrece opciones de gestión de productos, carrito de compras, procesamiento de pagos y más. WooCommerce es ampliamente utilizado y cuenta con una gran comunidad de apoyo.

Shopify: es una plataforma completa para crear y gestionar tiendas online. Proporciona una interfaz intuitiva y fácil de usar, así como numerosas plantillas y complementos para personalizar la tienda. También ofrece herramientas de gestión de inventario, procesamiento de pagos y opciones de marketing.

Magento: es un framework de comercio electrónico de código abierto que ofrece una amplia gama de funcionalidades para tiendas online.

Es muy personalizable y escalable, lo que lo hace adecuado para tiendas con necesidades más complejas. Cuenta con una comunidad activa y una gran cantidad de extensiones disponibles.

PrestaShop: es un framework de comercio electrónico de código abierto que permite crear tiendas online de manera sencilla. Ofrece una amplia selección de temas y módulos para personalizar la tienda, así como opciones de gestión de productos, pagos y envíos.

Django: es un framework de desarrollo web de Python que se enfoca en la eficiencia y la facilidad de uso. Proporciona una estructura sólida y segura para el desarrollo de aplicaciones web, y es adecuado para tiendas online que requieren una gestión compleja de datos y una integración con otros sistemas.

Además de los frameworks mencionados anteriormente, también existen otros frameworks populares que se pueden considerar, como **React, Angular y Vue.js**.

React, Angular y Vue.js son frameworks de JavaScript que se utilizan principalmente para construir interfaces de usuario interactivas y dinámicas

Estos frameworks son altamente flexibles y permiten crear experiencias de usuario modernas. Son adecuados para tiendas online que requieren una interacción intensiva del usuario y una experiencia fluida.

Selección del framework adecuado

La elección del framework para una tienda online dependerá de varios factores, como los requisitos específicos del proyecto, el tamaño y la complejidad de la tienda, así como el nivel de experiencia y preferencias del equipo de desarrollo.

Es importante evaluar cuidadosamente las funcionalidades proporcionadas por cada framework, su escalabilidad, la facilidad de personalización y el soporte de la comunidad al momento de seleccionar el framework adecuado para una tienda online.

Elegir el framework correcto ayudará a garantizar un desarrollo más rápido y eficiente de la tienda online, adaptándose a las necesidades específicas del proyecto.

Control de Versiones con Git y GitHub

El control de versiones es una práctica esencial en el desarrollo de software. Git y GitHub son unos de los sistemas de control de versiones más populares y ampliamente utilizados.

Git y GitHub son herramientas ampliamente utilizadas en el desarrollo de software para gestionar el control de versiones y colaborar en proyectos de programación. Aunque están relacionados, tienen roles y funciones diferentes. A continuación, te proporciono información sobre cada uno y cómo pueden ayudar a tu código:

Git: es un sistema de control de versiones distribuido diseñado para rastrear cambios en el código fuente durante el desarrollo de software.

Git permite a los programadores trabajar en proyectos de forma colaborativa y realizar un seguimiento de los cambios realizados en el código.

Cada vez que se realiza una modificación, Git registra un nuevo "commit", que representa un conjunto de cambios en el código. Esto permite a los desarrolladores volver atrás en el tiempo y acceder a versiones anteriores del código en caso de errores o necesidades de referencia.

Es especialmente útil para trabajar en equipo, ya que permite a los desarrolladores fusionar sus cambios de forma segura y gestionar conflictos cuando dos o más personas realizan modificaciones en la misma parte del código.

GitHub: es una plataforma basada en web que proporciona servicios de alojamiento para repositorios Git. Permite a los desarrolladores almacenar y compartir sus proyectos de software, colaborar con otros programadores y realizar un seguimiento de los cambios utilizando Git.

También ofrece características adicionales, como seguimiento de problemas, administración de proyectos, integración continua y despliegue (CI/CD), wikis y más. Lo que hacen de GitHub sea una herramienta poderosa para la colaboración y la gestión de proyectos de software.

Diferencias entre Git y GitHub

La principal diferencia entre Git y GitHub radica en su función y ubicación. Git es un sistema de control de versiones distribuido que se ejecuta en la máquina local del desarrollador, mientras que GitHub es una plataforma en línea que aloja repositorios Git y proporciona herramientas adicionales para la colaboración.

Git se utiliza principalmente en la máquina local para realizar seguimiento de cambios y gestionar el control de versiones. Por otra parte, GitHub es una plataforma en línea que actúa como un repositorio remoto para proyectos Git. Permite a los desarrolladores compartir su código con otros, colaborar en proyectos, revisar cambios y realizar un seguimiento de problemas y tareas asociadas con el desarrollo del software.

Cómo puede ayudar en mi código

Tanto Git como GitHub pueden ser muy beneficiosos para mi código y mi flujo de trabajo en el desarrollo de software. Algunas formas en que pueden ayudar son:

Git te permite **realizar un seguimiento de los cambios en tu código** a lo largo del tiempo. Puedes crear commits para registrar cambios específicos y revertir a versiones anteriores si es necesario, lo que facilita la colaboración y te brinda una mayor tranquilidad al tener un historial completo de cambios.

GitHub permite a los desarrolladores **trabajar en equipo de manera más eficiente**. Puedes compartir tu código con otros, crear ramas para desarrollar características o solucionar problemas sin afectar la rama principal, fusionar cambios de forma segura y revisar los cambios realizados por otros miembros del equipo.

En GitHub, tienes una **copia de seguridad** remota de tu código. En caso de pérdida de datos en tu máquina local, puedes clonar el repositorio desde GitHub y restaurar tu trabajo.

GitHub es una plataforma utilizada por una gran comunidad de desarrolladores de todo el mundo. Puedes explorar **proyectos de código abierto**, contribuir a ellos y aprender de otros desarrolladores. También puedes mostrar tus proyectos y recibir comentarios y contribuciones de la comunidad.

Conclusión

Los frameworks son herramientas clave en el desarrollo de una tienda online, ya que ofrecen una base sólida y funcionalidades predefinidas. Los frameworks como WooCommerce, Shopify, Magento y PrestaShop brindan soluciones completas y personalizables. Además, frameworks como React, Angular, Vue.js y Django también son opciones populares que brindan capacidades específicas para desarrollar interfaces de usuario dinámicas y gestión de datos eficiente.

Al seleccionar un framework, es importante considerar los requisitos específicos del proyecto, la escalabilidad, la facilidad de personalización y el soporte de la comunidad.

Elegir el framework adecuado ayudará a garantizar un desarrollo más rápido y eficiente de la tienda online.

Git y GitHub son herramientas esenciales para el desarrollo de software. Git te permite gestionar el control de versiones y realizar un seguimiento de los cambios en tu código, mientras que GitHub proporciona una plataforma en línea para alojar repositorios Git, colaborar con otros desarrolladores y aprovechar características adicionales para la gestión de proyectos.

Referencias

Revista Ingeniería y Tecnología: Framework:

- UNIR Research Platform (s.f.). "Revista Ingeniería y Tecnología: Framework." [En línea]. Disponible en: <https://unirfp.unir.net/revista/ingenieria-y-tecnologia/framework/> (Consultado el 19-01-2024).

Blog Assembler Institute: Framework de Programación:

- Assembler Institute (s.f.). "Blog Assembler Institute: Framework de Programación." [En línea]. Disponible en: <https://assemblerinstitute.com/blog/framework-programacion/> (Consultado el 19-01-2024).

Tutoriales de Frameworks:

- Dongee (s.f.). "Tutoriales de Frameworks." [En línea]. Disponible en: <https://www.dongee.com/tutoriales/frameworks/> (Consultado el 19-01-2024).

10 Criterios para Elegir el Framework Correcto:

- Jonathan Melgoza (s.f.). "10 Criterios para Elegir el Framework Correcto." [En línea]. Disponible en: <https://jonathanmelgoza.com/blog/10-criterios-para-elegir-el-framework-correcto/> (Consultado el 19-01-2024).



*The University of
Science & Technology*

Informe sobre el Diseño Responsive en una Tienda online

Introducción

El diseño responsive es una técnica que permite que una tienda online se visualice y funcione correctamente en dispositivos móviles, tabletas y computadoras de escritorio. En este informe, explicaremos qué es el diseño responsive y discutiremos algunas de las maneras de implementarlo en una tienda online.

¿Qué es el diseño responsive?

El diseño responsive es una técnica de diseño web que se enfoca en crear sitios web que se adapten y respondan a diferentes tamaños y resoluciones de pantalla.

En lugar de crear múltiples versiones de un sitio web para dispositivos específicos, el diseño responsive utiliza un conjunto de reglas y estilos CSS flexibles que permiten que el contenido se ajuste automáticamente al tamaño de la pantalla del dispositivo en el que se está visualizando.

Esto garantiza una experiencia de usuario consistente y optimizada, independientemente del dispositivo utilizado.

Maneras de hacer responsive una tienda online

Algunas maneras de hacer responsive una tienda online son:

Hacer un diseño adaptable: el diseño adaptable es la base del diseño responsive. Se trata de crear una estructura flexible que se ajuste y reorganice automáticamente según el tamaño de la pantalla.

Utilizando técnicas como grids fluidos, unidades de medida relativas (por ejemplo, porcentajes en lugar de píxeles) y media queries en CSS, el diseño adaptable permite que los elementos y el contenido se redistribuyan y escalen de manera adecuada en diferentes dispositivos.

Optimizar las Imágenes: las imágenes pueden representar un desafío en el diseño responsive debido a su tamaño y carga. Es importante optimizar las imágenes para diferentes resoluciones y tamaños de pantalla. Esto se puede lograr utilizando formatos de imagen adecuados, como el formato WebP, y técnicas como el lazy loading (carga diferida) para cargar las imágenes a medida que el usuario las visualiza.

Utilizar menús de navegación adecuados: en una tienda online responsive, es esencial tener en cuenta la navegación en dispositivos móviles.

Los menús desplegados tradicionales pueden ser difíciles de usar en pantallas pequeñas, por lo que en su lugar, es mejor utilizar técnicas como menús hamburguesa, navegación en pestañas o acordeones para optimizar la experiencia de navegación en dispositivos móviles.

Tener una tipografía legible: la legibilidad del texto es crucial en una tienda online. Es importante elegir tamaños de fuente adecuados y asegurarse de que el texto se muestre correctamente en diferentes dispositivos y tamaños de pantalla. También se tiene que tener en cuenta el espaciado y el contraste para garantizar una lectura cómoda.

Realizar pruebas en diferentes dispositivos: Esto permite identificar posibles problemas y realizar ajustes necesarios para asegurar una experiencia de usuario consistente y satisfactoria en todos los dispositivos.

Conclusión:

El diseño responsive es esencial para garantizar que una tienda online se adapte y proporcione una experiencia de usuario óptima en diferentes dispositivos y tamaños de pantalla.

Utilizando técnicas como el diseño adaptable, optimización de imágenes, navegación móvil amigable, tipografía legible y pruebas exhaustivas, es posible lograr una tienda online que se vea y funcione de manera excelente en cualquier dispositivo.

Al implementar un diseño responsive, se mejora la accesibilidad, se aumenta la satisfacción del usuario y se maximiza el potencial de conversión en una tienda online.

Referencias

Diseño Responsive:

- HubSpot (s.f.). "Diseño Responsive." [En línea]. Disponible en: <https://blog.hubspot.es/website/disenio-responsive> (Consultado el 20-01-2024).

¿Qué es el Diseño Responsive?:

- 40 de Fiebre (s.f.). "¿Qué es el Diseño Responsive?" [En línea]. Disponible en: <https://www.40defiebre.com/que-es/disenio-responsive> (Consultado el 20-01-2024).

¿Qué es un Diseño Responsive? Características y Ventajas:

- Isidro de la Rosa (s.f.). "¿Qué es un Diseño Responsive? Características y Ventajas." [En línea]. Disponible en: <https://www.isidrodelarosa.com/que-es-un-disenio-responsive-caracteristicas-y-ventajas/> (Consultado el 20-01-2024).