Bachiller: Yusneidi Hidalgo

C.I: 30.210.451

Investigación

¿Qué es un modelo de negocio?

Un modelo de negocio es la estructura que describe cómo una empresa crea, entrega y captura valor para ser rentable. Define cómo se ofrecen productos o servicios a un público objetivo, cómo se obtienen ingresos y cómo se gestionan los costes para lograr beneficios. Es un plano fundamental para el funcionamiento y la estrategia de cualquier organización.

Un modelo de negocio en programación se refiere a cómo una empresa crea, ofrece y monetiza su software para generar ingresos. Los modelos comunes incluyen SaaS (Software como servicio), donde se vende acceso por suscripción, la publicidad para generar ingresos a través de anuncios, las licencias de pago único y el modelo Freemium, que ofrece una versión básica gratuita con opciones Premium de pago

Elementos clave

¿Qué se ofrece? (Oferta): Los productos o servicios que la empresa pone a disposición de sus clientes.

¿Para quién se ofrece? (Usuarios): El segmento de mercado y los clientes objetivo a los que se dirige la empresa.

¿Cómo se llega a los clientes? (Canales): Las vías y estrategias para llegar a los clientes y entregar la oferta.

¿Cómo se generan ingresos? (Finanzas): Las estrategias para monetizar la oferta y generar beneficios.

¿Cómo se estructura la empresa? (Infraestructura): Los recursos (humanos, físicos, intelectuales), actividades clave y socios estratégicos necesarios para operar.

Utilidad del modelo de negocio

Para emprendedores: Sirve como una guía para definir y estructurar un nuevo proyecto empresarial.

Para empresas existentes: Ayuda a los gestores a guiar la dirección estratégica, orientar las operaciones y optimizar los recursos.

Para inversores: Proporciona una visión clara y concisa de las operaciones clave y la visión a largo plazo de la organización.

Ejemplos de modelos de negocio

Suscripción: Los clientes pagan una tarifa recurrente para acceder a un producto o servicio (ej: servicios de streaming).

Freemium: Se ofrece una versión básica gratuita y se cobra por funciones o características adicionales (ej: almacenamiento en la nube).

Marketplace: Una plataforma que conecta compradores y vendedores, obteniendo ingresos por comisión (ej: Airbnb, Amazon).

Publicidad: Se ofrecen contenidos gratuitos a cambio de mostrar anuncios (ej: Google).

¿Qué es una versión?

Una versión en programación es una iteración específica de un software, identificada por un número o código, que marca un estado particular de su desarrollo. Las versiones se utilizan para rastrear y administrar cambios, como la corrección de errores, la adición de nuevas funciones o mejoras de rendimiento, y se gestionan mediante sistemas de control de versiones como Git.

Características clave

Identificación única: Cada versión tiene un nombre o número único (por ejemplo, 1.0, 2.5.1-beta) que la distingue de otras iteraciones del mismo software.

Gestión de cambios: Permiten realizar un seguimiento de las modificaciones, facilitando la corrección de errores, la mejora de funciones y la implementación de nuevas características a lo largo del tiempo.

Reversibilidad: Posibilitan la restauración del software a un estado anterior si es necesario.

Información para instalaciones: Ayudan a los programas de instalación a identificar correctamente los archivos existentes y a instalar las actualizaciones apropiadas.

Tipos de cambios: Los esquemas numéricos como el versionado semántico indican la naturaleza del cambio:

Versión mayor: Introduce cambios significativos que rompen la compatibilidad con versiones anteriores.

Versión menor: Añade nuevas funcionalidades de manera retro compatible.

Parche o revisión: Corrige errores sin alterar la funcionalidad existente.

Herramientas de control de versiones

Para gestionar estas versiones de manera eficiente, se utilizan sistemas de control de versiones (VCS, por sus siglas en inglés). Estas herramientas guardan un registro de todas las modificaciones realizadas en el código y permiten a los desarrolladores colaborar, archivar y recuperar diferentes estados del proyecto.

¿Qué es un quiz?

Un "quiz" en programación es un cuestionario corto y rápido utilizado para evaluar el conocimiento o habilidades de una persona en un tema específico de programación. Pueden ser pruebas en línea o dentro de una aplicación, que utilizan preguntas de opción múltiple, verdadero/falso, o preguntas abiertas, y a menudo se usan como herramientas de aprendizaje, evaluación de candidatos o incluso en juegos de preguntas y respuestas.

Características principales

Evaluación rápida: Son pruebas cortas para valorar conocimientos específicos de forma concisa y rápida.

Variedad de formatos: Incluyen preguntas de opción múltiple, verdadero/falso, emparejamiento o respuestas abiertas.

Uso educativo: Se emplean para que los estudiantes practiquen y repasen conceptos de programación.

Evaluación de candidatos: Las empresas pueden utilizarlos para evaluar las habilidades de un candidato para resolver problemas de codificación en un tiempo determinado, permitiendo a los candidatos elegir entre varios lenguajes de programación.

Ejemplo práctico: En C++, se puede crear un juego de preguntas que almacena preguntas, opciones y respuestas correctas en arreglos, comprueba la respuesta del usuario y lleva un registro del puntaje.

¿Qué es un git?

Git es un sistema de control de versiones distribuido que permite a los desarrolladores registrar y gestionar los cambios en un proyecto de software a lo largo del tiempo. Su principal utilidad es que permite rastrear el historial del código,

facilitando la colaboración entre varios programadores, quienes pueden trabajar en paralelo sin sobrescribir los cambios de los demás gracias al uso de ramas.

Características principales de Git

Control de versiones: Git crea una instantánea del proyecto cada vez que se guarda un cambio (un "commit"), permitiendo volver a versiones anteriores si es necesario para corregir errores o revisar el historial.

Distribución: Cada desarrollador tiene una copia completa del repositorio localmente, lo que les permite trabajar sin conexión a internet y sincronizar sus cambios más tarde.

Colaboración en equipo: Facilita el trabajo en equipo al permitir que varios desarrolladores trabajen en el mismo proyecto simultáneamente. Cada uno puede trabajar en su propia "rama" y luego fusionar sus cambios en la rama principal una vez completados.

Flexibilidad: Es compatible con diversos flujos de trabajo y es eficiente tanto en proyectos pequeños como en los de gran tamaño.

Seguridad: Utiliza un algoritmo de "hash" criptográficamente seguro para proteger la integridad del código y el historial de cambios.

Velocidad: Muchas operaciones, como revisar el historial, se realizan localmente, lo que las hace muy rápidas.

Importancia de git

sirve para administrar y controlar versiones de código fuente. Facilita la colaboración en proyectos de software, ya que permite a varios programadores trabajar simultáneamente en diferentes aspectos del código sin generar conflictos.

¿Qué es un git bash?

Git Bash es una aplicación para Windows que emula el entorno de línea de comandos de Linux, instalando la shell Bash y herramientas de Git para que los desarrolladores puedan ejecutar comandos de Unix en su sistema operativo. Proporciona una interfaz de línea de comandos que permite a los usuarios trabajar con Git a través de comandos de texto, como git commit o git push, así como usar utilidades de Unix adicionales como ssh y scp.

Funcionalidades principales

Shell Bash: Incluye la shell Bash, que es una interfaz de línea de comandos popular en sistemas como Linux y macOS.

Herramientas de Git: Proporciona todas las utilidades necesarias para trabajar con Git de forma local y remota.

Comandos de Unix: Permite ejecutar comandos de Unix en Windows, que son comunes en el desarrollo de software y la administración de sistemas.

Scripts de Bash: Permite crear y ejecutar scripts de Bash para automatizar tareas repetitivas, como copias de seguridad, comprobaciones de estado del sistema y procesamiento de datos.

¿Cómo crear una cuenta github?

Para crear una cuenta en GitHub, visita github.com, haz clic en "Sign up" y sigue las indicaciones para ingresar tu correo electrónico, crear una contraseña y elegir un nombre de usuario. Completa la verificación de seguridad que te enviarán a tu email y, opcionalmente, ignora los pasos de personalización para empezar a usar tu cuenta gratuita de inmediato.

¿Cómo subir un repositorio?

Para subir un repositorio, primero crea un repositorio en línea en GitHub y luego inicialízalo localmente con git init. Después, añade los archivos con git add ., confirma los cambios con git commit -m "Mensaje", y vincula tu repositorio local con el remoto usando git remote add origin [URL del repositorio]. Finalmente, sube los archivos con git push -u origin main.

1. Preparación

Crea una cuenta en GitHub: Si aún no tienes una, regístrate en la página de GitHub.

Crea un nuevo repositorio en GitHub: Ve a tu perfil, haz clic en "New" o "Nuevo", ponle un nombre y haz clic en "Create repository".

2. Inicializa el repositorio local

Abre la terminal o Git Bash: en la carpeta de tu proyecto local.

Inicializa Git: Ejecuta el comando git init para crear un nuevo repositorio local.

3. Agrega, confirma y sube los cambios

Agrega los archivos: Usa git add . para agregar todos los archivos nuevos o modificados al "staging area".

Confirma los cambios: Usa git commit -m "mensaje" para guardar los cambios con un mensaje descriptivo.

Cambia la rama a main: Ejecuta git branch -M main para asegurarte de que estás en la rama principal.

Enlaza el repositorio remoto: Copia la URL de tu repositorio de GitHub y ejecuta git remote add origin [URL del repositorio]. Reemplaza [URL del repositorio] con la URL que copiaste.

Sube los cambios: Usa git push -u origin main para subir los cambios a tu repositorio remoto.

4. Actualiza el repositorio

Después de hacer más cambios, repite los pasos 3 (Agrega los archivos, Confirma los cambios y Sube los cambios) para subir tus actualizaciones.

¿Para que sirve, que hace, método de seguridad: git clone, https, ssh?

1. git clone

¿Qué es?

git clone es un comando de Git que se usa para copiar un repositorio remoto (en línea) a tu computadora.

Básicamente, crea una copia local completa del proyecto, con todos sus archivos, historial y ramas.

🔷 ¿Para qué sirve?

Sirve para empezar a trabajar con un repositorio que está alojado en plataformas como GitHub, GitLab o Bitbucket.

Ejemplo:

git clone https://github.com/usuario/proyecto.git

Esto descarga todo el repositorio dentro de una carpeta llamada proyecto

2. HTTPS

¿Qué es?

HTTPS (HyperText Transfer Protocol Secure) es un protocolo seguro de transferencia de datos por internet.

Cuando lo usas con Git, te conectas al repositorio remoto usando tu usuario y contraseña (o token).

¿Método de seguridad?

Usa cifrado SSL/TLS, lo que significa que los datos que viajan entre tu computadora y GitHub están encriptados.

Sin embargo, puede pedirte autenticación cada vez (a menos que uses un token personal o lo guardes en un credential manager).

Ejemplo de uso:

git clone https://github.com/usuario/proyecto.git

Ventajas:

Fácil de usar.

Funciona en cualquier red (firewalls, proxies, etc.).

Ideal si no quieres configurar claves SSH.

Desventajas:

Pide credenciales o tokens más seguido.

Menos práctico si haces muchos push/pull a diario.

3. SSH (Secure Shell)

¿Qué es?

SSH es un protocolo de comunicación segura que usa claves criptográficas (no contraseñas) para autenticarte ante el servidor.

Con Git, te permite conectarte a GitHub de forma segura y automática, sin tener que escribir contraseñas.

¿Método de seguridad?

Usa un par de claves:

🔑 Clave privada: guardada en tu PC.

🔑 Clave pública: registrada en tu cuenta de GitHub.

Solo quien tenga la clave privada puede acceder.

Toda la conexión va cifrada.

Ejemplo de uso:

git clone git@github.com:usuario/proyecto.git

Ventajas:

Más seguro y cómodo a largo plazo.

No pide usuario ni contraseña.

Ideal para desarrolladores que hacen muchos commits/pushes.

Desventajas:

Requiere configuración inicial (crear y registrar claves SSH).

Puede no funcionar en redes corporativas que bloqueen el puerto SSH (22).

¿Cuál es la tercera forma en una tabla de normalización?

En bases de datos relacionales, la tercera forma normal (3FN) o 3NF es una de las etapas de normalización cuyo objetivo es eliminar redundancias y dependencias innecesarias en las tablas para mejorar la organización de los datos.

¿Qué es la Tercera Forma Normal (3FN)?

Una tabla está en tercera forma normal si cumple con:

1. Está en segunda forma normal (2FN).

2. No existen dependencias transitivas entre los atributos no clave.

Definición técnica:

> Una tabla está en 3FN si todos los campos que no son clave dependen directamente de la clave primaria, y no de otro campo no clave.

En palabras simples:

👉 Ninguna columna debe depender de otra columna que no sea la clave primaria.

Ejemplo para entenderlo:

❌ Tabla no normalizada (dependencia transitiva)

ID_Empleado	Nombre	ID_Departamento	Nombre_Departamento
1	Ana Pérez	D1	Ventas
2	José López	D1	Ventas
3	Marta Díaz	D2	Contabilidad

Clave primaria: ID_Empleado

Problema: el Nombre_Departamento depende de ID_Departamento, no directamente de ID_Empleado.

👉 Hay una dependencia transitiva:

ID_Empleado → ID_Departamento → Nombre_Departamento

✅ Tabla en Tercera Forma Normal (3FN):

Tabla Empleados

ID_Empleado	Nombre	ID_Departamento
1	Ana Pérez	D1
2	José López	D1
3	Marta Díaz	D2

Tabla Departamentos

ID_Departamento	Nombre_Departamento
D1	Ventas
D2	Contabilidad

Ahora:

Cada campo depende directamente de su clave primaria.

No hay dependencias transitivas.

No hay datos duplicados del nombre del departamento.

Resumen general de las 3 primeras formas normales:

Forma Normal	Condición Principal	Objetivo
1FN	No hay grupos repetidos, cada celda tiene un solo valor.	Eliminar repeticiones.
2FN	Cumple 1FN y todos los campos dependen completamente de la clave primaria (no parcialmente).	Eliminar dependencias parciales.
3FN	Cumple 2FN y no hay dependencias transitivas.	Eliminar dependencias entre campos no clave.

Base de datos independiente

¿Por qué es necesario? ¿Por qué de esa manera?

¿Qué es una base de datos independiente?

Una base de datos independiente es aquella que no depende directamente de un programa o aplicación específica para poder funcionar o almacenar los datos.

Es decir, los datos están separados del software que los usa.

En otras palabras:

> La base de datos puede seguir existiendo, modificarse o consultarse aunque cambie el programa o la aplicación que la utiliza.

Ejemplo sencillo:

Imagina que tienes un sistema de gestión de ventas:

Antes, los datos estaban dentro del mismo programa (por ejemplo, un Excel con macros o una app vieja).

Si el programa se dañaba o se actualizaba, se perdían los datos.

Ahora, si tienes una base de datos independiente (como MySQL, PostgreSQL o SQL Server):

Los datos están guardados en un servidor aparte.

Puedes cambiar el programa o crear otro (una app web, móvil, o de escritorio) y seguir usando los mismos datos.

¿Por qué es necesaria una base de datos independiente?

Porque separa la lógica del sistema (el programa) de la gestión de los datos, lo que aporta muchas ventajas:

Razón Explicación

1. Seguridad de los datos Los datos están protegidos incluso si la aplicación falla o cambia.

2. Reutilización Diferentes aplicaciones pueden usar la misma base de datos.

3. Mantenimiento fácil Se pueden hacer copias de seguridad, optimizaciones o migraciones sin afectar el software.

4. Escalabilidad Permite crecer (más usuarios, más datos) sin reescribir el sistema.

5. Integridad Las reglas y relaciones entre datos se controlan directamente en la base de datos (por ejemplo, claves primarias y foráneas).

¿Por qué se hace de esa manera?

Porque en el diseño profesional de sistemas se busca el principio de independencia de datos, que se divide en dos tipos:

Tipo de independencia Significa que...

Independencia física Se pueden cambiar aspectos físicos (como ubicación o estructura interna de archivos) sin afectar las aplicaciones.

Independencia lógica Se pueden modificar estructuras lógicas (como tablas o campos) sin tener que reprogramar todo el sistema.

Esto permite que un sistema sea flexible, seguro y duradero.

¿Qué tipo de validaciones se utilizan para que no inserten datos maliciosos?

Tipos de validaciones para evitar datos maliciosos

1.Validación del lado del cliente (frontend)

Se hace antes de enviar el formulario, en el navegador (con HTML o JavaScript).

🔷 Ejemplos:

Campos obligatorios (required)

Longitud mínima/máxima (minlength, maxlength)

Validar formato de correo (type="email")

Validar que un número esté dentro de un rango (min, max)

✅ Ventaja:

Evita errores simples y mejora la experiencia del usuario.

Limitación:

No es suficiente sola, porque el usuario puede desactivarla o manipular el código con herramientas como inspeccionar elemento.

2. Validación del lado del servidor (backend)

Es la más importante, ya que el servidor verifica los datos antes de guardarlos o procesarlos.

🔷 Ejemplos de validaciones comunes:

Verificar que los campos requeridos no estén vacíos.

Asegurarse de que los datos tengan el formato correcto (correo, fecha, número, etc.).

Limitar el tamaño de los textos para evitar desbordamientos.

Comprobar que los datos no contengan código peligroso.

Lenguajes como PHP, Python o Java suelen usar expresiones regulares (regex) para esto.

3. Validación y sanitización de entradas (Input Sanitization)

Este paso limpia el contenido que envía el usuario antes de usarlo.

🔷 Ejemplo:

Eliminar etiquetas HTML, scripts o caracteres especiales que puedan causar ataques como:

XSS (Cross-Site Scripting)

SQL Injection

🔷 En PHP:

$nombre = htmlspecialchars($_POST['nombre']);

$email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);

4. Uso de consultas preparadas (Prepared Statements)

Evita que los atacantes inyecten código SQL en los formularios.

🔷 Ejemplo de SQL Injection malicioso:

SELECT * FROM usuarios WHERE usuario = 'admin' OR '1'='1';

🔷 Solución con consultas preparadas (PHP - PDO):

$stmt = $pdo->prepare("SELECT * FROM usuarios WHERE usuario = :usuario");

$stmt->bindParam(':usuario', $usuario);

$stmt->execute();

Aquí, el valor del usuario no se interpreta como código, sino como dato seguro.

5. Validaciones de tipo y longitud

Asegura que los datos sean del tipo esperado (texto, número, fecha, booleano, etc.) y con un tamaño limitado.

🔷 Ejemplo:

Edad: debe ser un número entero y positivo.

Nombre: solo letras, máximo 50 caracteres.

Contraseña: mínimo 8 caracteres, con letras, números y símbolos.

6. Validación de permisos y roles

Evita que los usuarios accedan a datos o acciones que no les corresponden.

🔷 Ejemplo:

Un usuario normal no puede borrar otros usuarios.

Un administrador sí, pero con autenticación extra.

7. Codificación y escape de salida (Output Escaping)

Antes de mostrar datos del usuario en la web, se deben “escapar” para que no se interpreten como código HTML o JavaScript.

🔷 Ejemplo:

echo htmlspecialchars($comentario);

Así, si alguien intenta insertar:

<script>alert('hack');</script>

Solo se mostrará en texto, no se ejecutará.