

TECNOLOGÍAS DE LA INFORMACIÓN EN LÍNEA

DESARROLLO DE APLICACIONES WEB

3 créditos

**Profesor:**

Ing. Manuel Bravo, Mg.

Ing. Tatiana Cobeña Macías, Mg.

Ing. Edison Solorzano, MG.

Asignatura	Semestre
DESARROLLO DE APLICACIONES WEB	Sexto

Tutorías: El profesor asignado, se publicará en el entorno virtual de aprendizaje online.utm.edu.ec), y sus horarios de conferencias se indicarán en la sección **CAFETERÍA VIRTUAL**.

PERÍODO ACADÉMICO:

MAYO 2023 / SEPTIEMBRE 2023

Tabla de Contenido

Tabla de Ilustración.....	¡Error! Marcador no definido.
Ilustraciones gráficas	1
Resultado de aprendizaje de la asignatura	2
Organización de la lectura para el estudiante por semana del compendio	2
UNIDAD I: PROGRAMACIÓN WEB EN EL SERVIDOR	2
Resultado de aprendizaje de la unidad:	2
Ejes temáticos	2
TEMA 1: MODELO CLIENTE-SERVIDOR.....	4
TEMA 2: MODELO, VISTA, CONTROLADOR (MVC)	13
TEMA 3: ORM	16
TEMA 4: PRUEBAS UNITARIAS	21
TEMA 5: SERVERLESS COMPUTING	29



Ilustraciones gráficas



Sabías que. - La presente imagen dentro del manual mostrara información interesante y novedosa de la asignatura.



Recuerde que. - La presente imagen dentro del manual permite recordar información que es relevante y que vas necesitar en tu vida profesional.



Comprueba tu aprendizaje. - Es un cuestionario de un conjunto de preguntas que se confecciona para obtener información con algún objetivo en concreto. Por cada tema de la unidad se tendrá un cuestionario que el estudiante debe resolver entre preguntas teóricas y prácticas.



Videos. - Para complementar contenidos de la unidad dentro del manual se tiene videos que permitirán al estudiante revisar y explorar conocimientos auditivos y visuales.



Curiosidades. - La presente imagen en el manual mostrara información que debes conocer de la asignatura.



Datos útiles. - La presente imagen en el manual mostrara información que deberás tomar en cuenta en otras unidades de la asignatura o de otras asignaturas en semestres superiores.



Resultado de aprendizaje de la asignatura

Este curso provee a los estudiantes el conocimiento y la experiencia práctica para diseñar e implementar aplicaciones web cumpliendo con los estándares actuales y las buenas prácticas de programación que faciliten su mantenibilidad, escalabilidad y adaptabilidad.



Organización de la lectura para el estudiante por semana del compendio

Semana	Compendio
Semana 1	Páginas 6 - 24
Semana 2	Páginas 24 - 34
Semana 3	Página 34 – 45



DESARROLLO DE APLICACIONES WEB

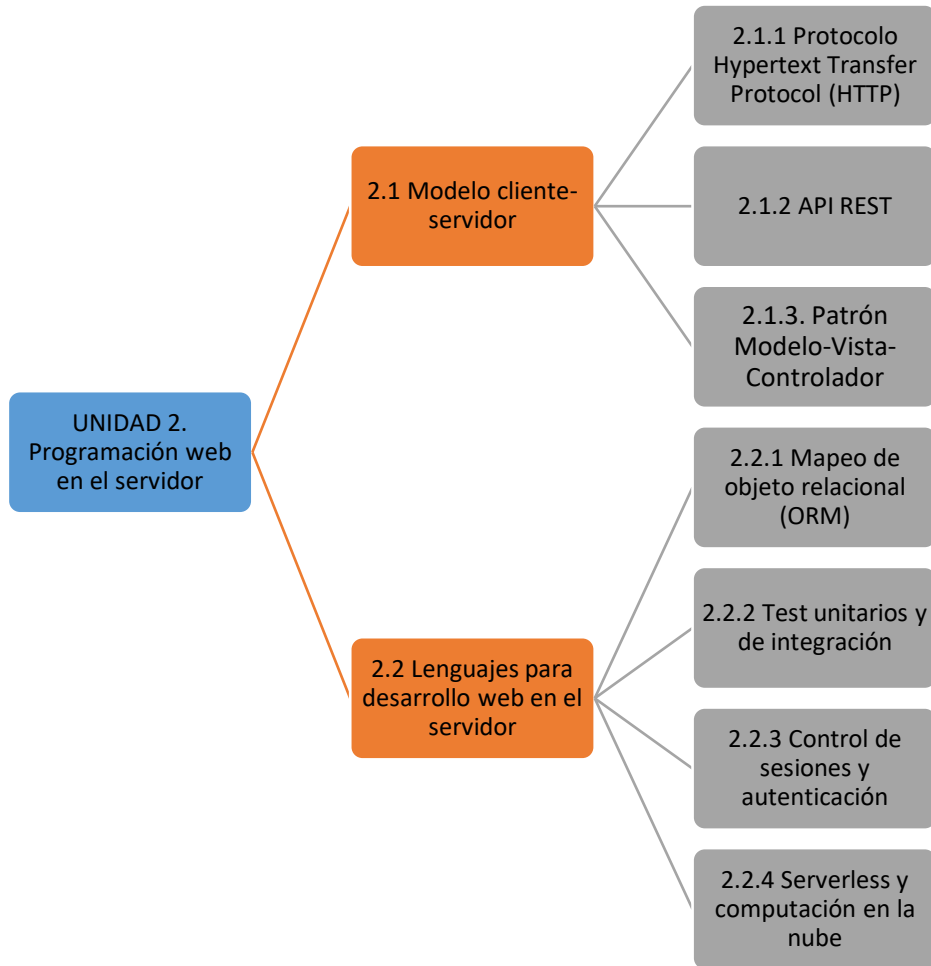


UNIDAD I: PROGRAMACIÓN WEB EN EL SERVIDOR

Resultado de aprendizaje de la unidad: Aplicar el patrón modelo-vista-controlador en el desarrollo de una aplicación web para que el proceso de mantenimiento sea rápido.

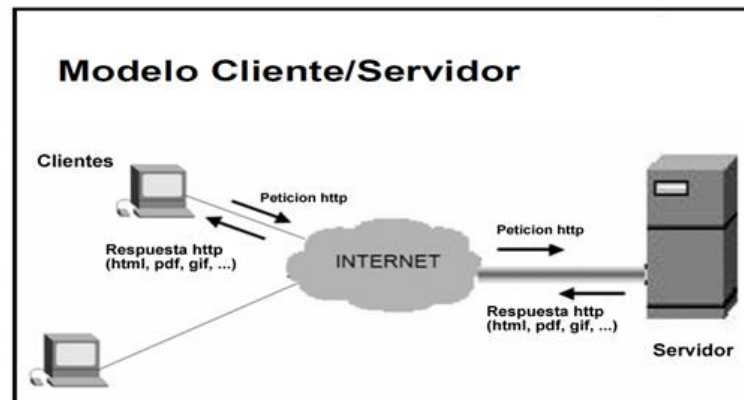


Ejes temáticos



TEMA 1: MODELO CLIENTE-SERVIDOR

Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma. En el modelo cliente servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio). A continuación, la siguiente figura:



Fuente: Elaborado por autores.

La idea es tratar a una computadora como un instrumento, que por sí sola pueda realizar muchas tareas, pero con la consideración de que realice aquellas que son más adecuadas a sus características. Si esto se aplica tanto a clientes como servidores se entiende que la forma más estándar de aplicación y uso de sistemas Cliente/Servidor es mediante la explotación de las PC's a través de interfaces gráficas de usuario; mientras que la administración de datos y su seguridad e integridad se deja a cargo de computadoras centrales tipo mainframe. Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y el o los procesos cliente sólo se ocupan de la interacción con el usuario (aunque esto puede variar). En otras palabras, la arquitectura Cliente/Servidor es una extensión de programación modular en la que la base fundamental es separar una gran pieza de software en módulos con el fin de hacer más fácil el desarrollo y mejorar su mantenimiento.



Sabías que. - Esta arquitectura permite distribuir físicamente los procesos y los datos en forma más eficiente lo que en computación distribuida afecta directamente el tráfico de la red, reduciéndolo grandemente.

CLIENTE

El cliente es el actor que formula los requerimientos y los transmite al servidor a través de un ordenador, generalmente se asocia al Frontend en este lado del modelo. Además, el cliente normalmente maneja todas las funciones relacionadas con la manipulación y despliegue de datos, por lo que están desarrollados sobre plataformas que permiten construir interfaces gráficas de usuario (GUI), además de acceder a los servicios distribuidos en cualquier parte de una red.

Las funciones que se llevan a cabo en el lado cliente, son:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

SERVIDOR

Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se le conoce con el término Backend. El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos.

Las funciones que se llevan a cabo en el lado servidor, son:

- Aceptar los requerimientos de bases de datos que hacen los clientes.
- Procesar requerimientos de bases de datos.
- Formatear datos para transmitirlos a los clientes.
- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

CARACTERÍSTICAS DEL MODELO CLIENTE-SERVIDOR

- Combinación de un cliente que interactúa con el usuario, y un servidor que interactúa con los recursos compartidos. El proceso del cliente proporciona la interfaz entre el usuario y el resto del sistema. El proceso del servidor actúa como un motor de software que maneja recursos compartidos tales como bases de datos, impresoras, módems, etc.

- Las tareas del cliente y del servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, velocidad y capacidades del disco e input-output devices.
- Se establece una relación entre procesos distintos, los cuales pueden ser ejecutados en la misma máquina o en máquinas diferentes distribuidas a lo largo de la red. Existe una clara distinción de funciones basada en el concepto de "servicio", que se establece entre clientes y servidores.
- La relación establecida puede ser de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando su acceso a recursos compartidos.
- Los clientes corresponden a procesos activos en cuanto a que son éstos los que hacen peticiones de servicios a los servidores. Estos últimos tienen un carácter pasivo ya que esperan las peticiones de los clientes.
- No existe otra relación entre clientes y servidores que no sea la que se establece a través del intercambio de mensajes entre ambos. El mensaje es el mecanismo para la petición y entrega de solicitudes de servicio.
- El ambiente es heterogéneo. La plataforma de hardware y el sistema operativo del cliente y del servidor no son siempre la misma. Precisamente una de las principales ventajas de esta arquitectura es la posibilidad de conectar clientes y servidores independientemente de sus plataformas.
- El concepto de escalabilidad tanto horizontal como vertical es aplicable a cualquier sistema Cliente/Servidor. La escalabilidad horizontal permite agregar más estaciones de trabajo activas sin afectar significativamente el rendimiento. La escalabilidad vertical permite mejorar las características del servidor o agregar múltiples servidores.

VENTAJAS DEL MODELO CLIENTE-SERVIDOR

- Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor, es la existencia de plataformas de hardware cada vez más baratas. Esta constituye a su vez una de las más palpables ventajas de este esquema, la posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Además, se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual

contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.

- El esquema Cliente/Servidor facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo, que las máquinas ya existentes puedan ser utilizadas, pero utilizando interfaces más amigables al usuario. De esta manera, podemos integrar PCs con sistemas medianos y grandes, sin necesidad de que todos tengan que utilizar el mismo sistema operacional.
- Al favorecer el uso de interfaces gráficas interactivas, los sistemas contruidos bajo este esquema tienen mayor interacción y más intuitiva con el usuario. En el uso de interfaces gráficas para el usuario, el esquema Cliente/Servidor presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.
- Una ventaja adicional del uso del esquema Cliente/Servidor es que es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes (por ejemplo, los servidores de SQL o las herramientas de más bajo nivel como los sockets o el RPC).
- La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.
- El esquema Cliente/Servidor contribuye, además a proporcionar, a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información relevante a nivel global.

DESVENTAJAS DEL MODELO CLIENTE-SERVIDOR

- El mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software, distribuidas por distintos proveedores, lo cual dificulta el diagnóstico de fallas.
- Se cuenta con muy escasas herramientas para la administración y ajuste del desempeño de los sistemas.
- Es importante que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo, sockets o RPC), lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas.

- Además, hay que tener estrategias para el manejo de errores y para mantener la consistencia de los datos.
- La seguridad de un esquema Cliente/Servidor es otra preocupación importante. Por ejemplo, se deben hacer verificaciones en el cliente y en el servidor.



Fuente: Elaborado por autores.



Curiosidades. – Pese a la existencia de modernas tecnologías y sistemas, el modelo Cliente-Servidor ha seguido vigente en el mercado.

TIPOS DE SERVIDORES

➤ Servidores FTP

Uno de los servicios más antiguos de Internet, File Transfer Protocol permite mover uno o más archivos. Con una de las alternativas más importantes que nos

permite Internet es la transferencia de archivos de una computadora a otra desde cualquier parte del mundo. Para ello utilizamos el protocolo de transferencia de archivos o “FTP”.

➤ Servidores Groupware

Un servidor groupware es un software diseñado para permitir colaborar a los usuarios, sin importar la localización, vía Internet o vía Intranet corporativo y trabajar juntos en una atmósfera virtual.

➤ **Servidores IRC**

Otra opción para usuarios que buscan la discusión en tiempo real, Internet Relay Chat consiste en varias redes de servidores separadas que permiten que los usuarios conecten el uno al otro vía una red IRC.

➤ **Servidores de Listas**

Los servidores de listas ofrecen una manera mejor de manejar listas de correo electrónico, bien sean discusiones interactivas abiertas al público o listas unidireccionales de anuncios, boletines de noticias o publicidad.

➤ **Servidores de Noticias**

Los servidores de noticias actúan como fuente de distribución y entrega para los millares de grupos de noticias públicos actualmente accesibles a través de la red de noticias USENET.

➤ **Servidores Telnet**

Un servidor telnet permite a los usuarios entrar en un ordenador huésped y realizar tareas como si estuviera trabajando directamente en ese ordenador.

➤ **Servidores Proxy**

Los servidores proxy se sitúan entre un programa del cliente (típicamente un navegador) y un servidor externo (típicamente otro servidor web) para filtrar peticiones, mejorar el funcionamiento y compartir conexiones.

➤ **Servidor Web**

Básicamente, un servidor web sirve contenido estático a un navegador, carga un archivo y lo sirve a través de la red.

➤ **Servidor DHCP**

El protocolo de configuración dinámica de host (DHCP, Dynamic Host Configuration Protocol) es un estándar TCP/IP diseñado para simplificar la administración de la configuración IP de los equipos de nuestra red. El estándar DHCP permite el uso de servidores DHCP para administrar la asignación dinámica, a los clientes DHCP de la red, de direcciones IP y otros detalles de configuración relacionados, siempre que los clientes estén configurados para utilizar un servidor DHCP.

➤ **Servidor de Base de Datos**

Da servicios de almacenamiento y gestión de bases de datos a sus clientes. Una base de datos es un sistema que nos permite almacenar grandes cantidades de información.

Por ejemplo, todos los datos de los clientes de un banco y sus movimientos en las cuentas.

➤ **Servidor Clúster**

Son servidores especializados en el almacenamiento de la información teniendo grandes capacidades de almacenamiento y permitiendo evitar la pérdida de la información por problemas en otros servidores.

➤ **Servidores dedicados**

Hay servidores compartidos si hay varias personas o empresas usando un mismo servidor, o dedicados que son exclusivos para una sola persona o empresa.

➤ **Servidores de imágenes**

Recientemente se han popularizado servidores especializados en imágenes, permitiendo alojar gran cantidad de imágenes sin consumir recursos de nuestro servidor web en almacenamiento o para almacenar fotografías personales, profesionales.

➤ **Servidor de Archivo**

Es el que almacena varios tipos de archivos y los distribuye a otros clientes en la red.

➤ **Servidor de impresión**

Controla una o más impresoras y acepta trabajos de impresión de otros clientes de la red, poniendo en cola los trabajos de impresión (aunque también puede cambiar la prioridad de las diferentes impresiones), y realizando la mayoría o todas las otras funciones que en un sitio de trabajo se realizaría para lograr una tarea de impresión si la impresora fuera conectada directamente con el puerto de impresora del sitio de trabajo.

➤ **Servidor de fax**

Almacena, envía, recibe, enruta y realiza otras funciones necesarias para la transmisión, la recepción y la distribución apropiadas del fax.

➤ **Servidor de telefonía**

Realiza funciones relacionadas con la telefonía, como es la de contestador automático, realizando las funciones de un sistema interactivo para la respuesta de la voz, almacenando los mensajes de voz, encaminando las llamadas y controlando también la red o el Internet, p. ej., la entrada excesiva de la voz sobre IP (VoIP), etc.

➤ **Servidor del acceso remoto (RAS)**

Controla las líneas de módem de los monitores u otros canales de comunicación de la red para que las peticiones conecten con la red de una posición remota, responde llamadas telefónicas entrantes o reconoce la petición de la red y realiza la autenticación necesaria y otros procedimientos necesarios para registrar a un usuario en la red.

➤ **Servidor de uso**

Realiza la parte lógica de la informática o del negocio de un uso del cliente, aceptando las instrucciones para que se realicen las operaciones de un sitio de trabajo y sirviendo los resultados a su vez al sitio de trabajo, mientras que el sitio de trabajo realiza la interfaz operadora o la porción del GUI del proceso (es decir, la lógica de la presentación) que se requiere para trabajar correctamente.

➤ **Servidor no dedicado**

Son aquellos que no dedican toda su potencia a los clientes, sino también pueden jugar el rol de estaciones de trabajo al procesar solicitudes de un usuario local.

INTERACCIÓN HUMANO COMPUTADOR (HCI)

Es el proceso de comunicación entre el usuario y el computador. En general, Es la disciplina que estudia el intercambio de información mediante software. Ésta se encarga del diseño, evaluación e implementación de los aparatos tecnológicos interactivos.

PIEZAS CLAVES DEL HCI

Son las **guías** y **estándares** que regulan la comunicación entre el usuario y el computador.

ESTÁNDAR

Un estándar es un requisito, regla o recomendación basada en principios probados y en la práctica.

- Representa un acuerdo de un grupo de profesionales oficialmente autorizados a nivel.
- **Local** (aceptado desde una industria, organización profesional o entidad empresarial).
- **Nacional** (aceptado por una amplia variedad de organizaciones dentro de una nación).
- **Internacional** (consenso entre organizaciones de estándares a nivel mundial).

COMITES DEDICADOS A LA CREACIÓN DE ESTÁNDARES

- ISO (Organización Internacional para Estándares).
- IEC (Comisión Electrotécnica Internacional).
- ANSI (Instituto Nacional Americano para Estándares).
- IEEE (Instituto de Ingenieros Eléctricos y Electrónicos Americano).
- CEN (Comité Europeo para la Estandarización).
- W3C (Consortio para el World Wide Web).

ISO

Fundada en 1947 y con sede en Ginebra (Suiza).

- Federación mundial de cuerpos de estándares nacionales de más de 130 países.
- Abarca casi todos los campos excepto básicamente la ingeniería eléctrica (responsabilidad del IEC, Comisión Electrotécnica Internacional).
- La coordinación entre ISO e IEC es responsabilidad del Comité Técnico de Conexión (JTC).

ANSI

Instituto Nacional Americano para Estándares creado en 1918.

- Instituto de carácter privado, sin ánimo de lucro, dedicado a la representación de los intereses de casi 1000 compañías, organizaciones y agencias del gobierno.
- Su principal misión es ampliar la competitividad de las empresas de los EEUU a través de la generación de estándares.

IEEE

Instituto de Ingenieros Eléctricos y Electrónicos Americano. Asociación para Estándares.

- Su principal misión es desarrollar y publicar estándares generalmente aceptados, que promoverán la teoría y práctica de la ingeniería eléctrica, electrónica e informática, así como el resto de ramas de la ingeniería y artes o ciencias relacionadas.

CEN

Comité europeo para la estandarización.

- Es el principal proveedor de estándares europeos y especificaciones técnicas.

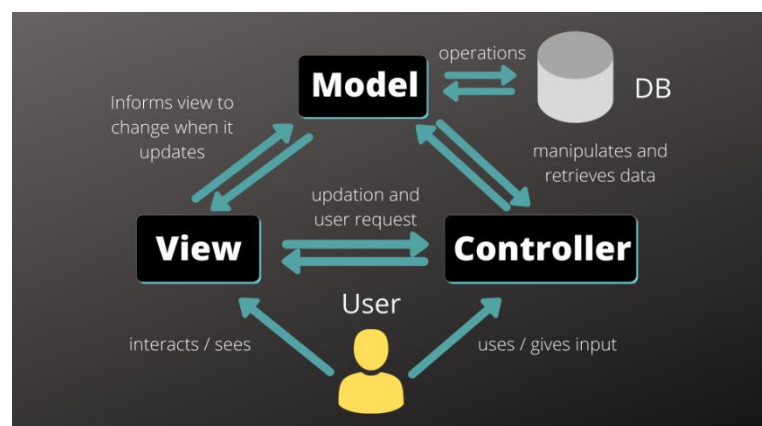
W3C

Consorcio para el World Wide Web creado en 1994 por Tim Berners-Lee en el MIT, con la colaboración del Instituto Nacional de Investigación en Informática y Automática (INRIA) y la Universidad de Keio (Japón).

- Su objetivo es llevar el World Wide Web a su pleno potencial, desarrollando protocolos comunes que promueven su evolución y aseguran su interoperabilidad.
- Está constituido por más de 500 organizaciones en todo el mundo.
- Proporciona recomendaciones creadas por grupos de trabajo en áreas relacionadas con las interfaces de usuario:
 - Accesibilidad, Internacionalización, Adaptabilidad, buenas prácticas de programación, entre otros aspectos.
- Disponibilidad totalmente gratuita.



TEMA 2: MODELO, VISTA, CONTROLADOR (MVC)



Fuente: Elaborado por autores.

INTRODUCCIÓN A MVC

Es un patrón de arquitectura de las aplicaciones software.

- Separa la lógica de negocio de la interfaz de usuario.
- Facilita la evolución por separado de ambos aspectos.
- Incrementa reutilización y flexibilidad.

HISTORIA

- ✓ Descrito por primera vez en 1979 para Smalltalk.
- ✓ Utilizado en múltiples frameworks: Java Swing, Java Enterprise Edition (J2EE), XForms (Formato XML estándar del W3C para la especificación de un modelo de proceso de datos XML e interfaces de usuario como formularios web).
- ✓ GTK+ (escrito en C, toolkit creado por Gnome para construir aplicaciones gráficas, inicialmente para el sistema X Window).
- ✓ ASP.NET MVC Framework (Microsoft).
- ✓ Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java).
- ✓ Apache Struts (framework para aplicaciones web J2EE).
- ✓ Ruby on Rails (framework para aplicaciones web con Ruby, entre otros).

CARACTERÍSTICAS

- ✓ Un modelo (Base de datos).
- ✓ Varias vistas (Plantillas gráficas).
- ✓ Varios controladores (Programación)
- Las vistas y los controladores suelen estar muy relacionados.
- Los controladores tratan los eventos que se producen en la interfaz gráfica (vista). Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador.

FLUJO DE TRABAJO EN MVC

1. El usuario realiza una acción en la interfaz.
2. El controlador trata el evento de entrada.
3. Previamente se ha registrado, el controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta).
4. Se genera una nueva vista. La vista toma los datos del modelo.

5. El modelo no tiene conocimiento directo de la vista.
6. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.

MVC EN APLICACIONES WEB

- **Vista:** la página HTML
- **Controlador:** código que obtiene datos dinámicamente y genera el contenido HTML.
- **Modelo:** la información almacenada en una base de datos o en XML junto con las reglas de negocio que transforman esa información (teniendo en cuenta las acciones de los usuarios).



Fuente: Elaborado por autores.

Como parte práctica del MVC, les he colocado un tutorial muy bueno donde se aplican los principios básicos del MVC en PHP nativo sin frameworks, para que vean como se realiza desde cero.

Link: <https://www.codigonexo.com/wp-content/uploads/2014/06/Curso-completo-MVC.pdf>

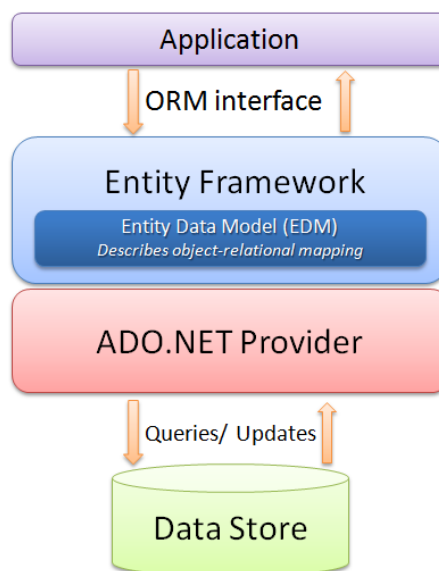
Código: <https://www.codigonexo.com/wp-content/uploads/2014/06/mvc.rar>



TEMA 3: ORM

ENTITY FRAMEWORK

Entity Framework (EF) es un ORM (object-relational mapper) que permite a los desarrolladores en .NET trabajar con datos relacionales usando objetos específicos del dominio. Elimina la necesidad de escribir la mayoría de código de acceso a datos. Es un mecanismo automatizado para acceder y consultar datos en una base de datos y trabajar con los resultados.



Fuente: Elaborado por autores.

COMPONENTES DEL ENTITY FRAMEWORK

- ✓ EDM (Entity Data Model).
- ✓ Conceptual Model.
- ✓ Storage Model.
- ✓ Mapping.
- ✓ LINQ to Entities.
- ✓ Entity SQL.
- ✓ Object Service.
- ✓ Entity Client Data Provider.
- ✓ ADO.Net Data Provider.

AHORA SÍ, QUÉ ES UN ORM

Mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional, utilizando un motor de persistencia.



Sabías que. - En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

PRINCIPALES ORM DEL MERCADO

- Java: EJB, Hibernate, Athena Framework, Java Data Objects.
- .NET: ADO.NET Entity Framework, Linq to SQL, nHibernate, DataObjects.NET.
- PHP: CakePHP, FuelPHP, Qcodo, Readbean, Zend Framework.
- Python: Django, SQLAlchemy, Storm, Tryton.
- Ruby: ActiveRecord, Datamapper, Ibatis.

ENTITY OBJECTS

```
[EdmEntityTypeAttribute(NamespaceName="SchoolDBModel", Name="Student")]
[Serializable()]
[DataContractAttribute(IsReference=true)]
public partial class Student : EntityObject
{
    #region Factory Method

    /// <summary>
    /// Create a new Student object.
    /// </summary>
    /// <param name="studentID">Initial value of the StudentID property.</param>
    /// <param name="standardId">Initial value of the StandardId property.</param>
    public static Student CreateStudent(global::System.Int32 studentID, global::System.Int32 standardId)
    {
        Student student = new Student();
        student.StudentID = studentID;
        student.StandardId = standardId;
        return student;
    }

    #endregion
    #region Primitive Properties
```

Fuente: Elaborado por autores.

Por default el Entity Data Model genera EntityObjects que son derivados de entidades (tablas/vistas).

POCO (Plain Old CLR Object):

Las clases POCO es una clase la cual no depende de ninguna clase base de un Framework específico.

```
public class Student
{
    public int StudentID { get; set; }

    public string StudentName { get; set; }

    public Standard Standard { get; set; }

    public StudentAddress StudentAddress { get; set; }

    public IList<Course> Courses { get; set; }
}
```

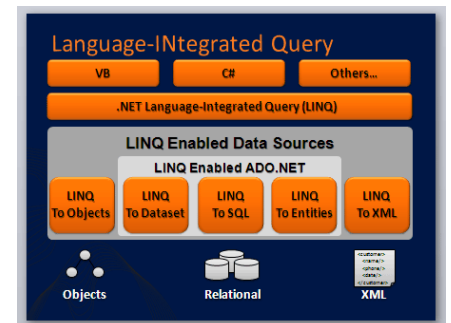
Fuente: Elaborado por autores.

LINQ (Lenguaje Integrated Query)

Linq simplemente es una expresión tipo Query en nuestro código, similar a un Query de SQL en nuestra base de datos. Es aplicable sobre colecciones:

- Listas
- Arreglos
- XML
- Tablas de base de datos

ACCESO A DATOS CON ADO.NET



Fuente: Elaborado por autores.

```
List<Customer> Customers = new List<Customer>();

string QueryString = "Data source=localhost; initial catalog=SubwayDB; integrated security=true";
SqlConnection sqlConn = new SqlConnection(QueryString);

string Query = "Select * from Customers where FirstName=@FN"; SqlCommand sqlComm = new SqlCommand(Query, sqlConn);

sqlComm.CommandType = System.Data.CommandType.Text;
sqlComm.Parameters.AddWithValue("@FN", "test").DbType = System.Data.DbType.Int16; sqlConn.Open();

SqlDataReader reader = sqlComm.ExecuteReader();

while (reader.Read())
{
    Customers.Add(new Customer()
    {
        FirstName = reader["FirstName"].ToString(),
        LastName = reader["LastName"].ToString(),
        Email = reader["Email"].ToString()
    });
}

reader.Close();
sqlConn.Close();
```

Fuente: Elaborado por autores.

ACCESO A DATOS CON LINQ

Fácil:

```
List<Customer> customers = from c in Customers
                             where c.FirstName == "Test"
                             select c;
```

Rápido de codificar y legible:

```
List<Customer> customers = dbContext.Customers.Where(c => c.FirstName ==
"Test").ToList();
```

LINQ vs NATIVE SQL

Ventajas:

- No hay cadenas mágicas como las que se pueden generar en sentencias SQL (Prevención de Inyección).
- Intellisense.
- Un desarrollo más rápido.
- Autogeneración de objetos de dominio que pueden ser proyectos reusables.
- Lazy loading (Cargado de objetos relacionados sobre demanda)
- Lambda expressions y extension methods.

Desventajas:

- Depuración (Debugging).
- Consultas complejas traen como resultado problemas de rendimiento.
- Sobrecarga al crear consultas.
- Los Índices de base de datos no son usados adecuadamente.

EJEMPLO PRÁCTICO DE ORM EN C#

Mapeo Objeto-Relacional o como se conocen comúnmente, ORM (del inglés Object Relational Mapping), permite convertir los datos de tus objetos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en nuestra aplicación, quedan vinculados a la base de datos (persistencia).

Si alguna vez has programado alguna aplicación que se conecta a una base de datos, habrás podido comprobar lo laborioso que es transformar toda la información que recibes de la base de datos, principalmente en tablas, en los objetos de tu aplicación y viceversa. A esto se le denomina mapeo. Utilizando un ORM este mapeo será automático, es más, será independiente de la base de datos que estés utilizando en ese momento pudiendo cambiar de motor de base de datos según tus necesidades. Veamos un ejemplo. Supongamos que tenemos una tabla de clientes. En nuestra aplicación queremos hacer las funciones básicas sobre base de datos CRUD (del inglés Create, Read, Update and Delete) Crear, Obtener, Actualizar y Borrar. Cada operación corresponde con una sentencia SQL.

- Crear: INSERT
- Obtener: SELECT
- Actualizar: UPDATE
- Borrar: DELETE

Si queremos insertar un cliente nuevo y no utilizamos un ORM, el código quedaría de la siguiente manera si utilizamos C#:

```
1 String query = "INSERT INTO clientes (id,nombre,email,pais) VALUES (@id, @nombre, @email, @pais)";
2
3 command.Parameters.AddWithValue("@id", "1")
4 command.Parameters.AddWithValue("@nombre", "nombre")
5 command.Parameters.AddWithValue("@email", "email")
6 command.Parameters.AddWithValue("@pais", "pais")
7
8 command.ExecuteNonQuery();
```

Fuente: Elaborado por autores.

En cambio, si utilizamos un ORM, el código se puede reducir de la siguiente manera:

```
1 var cliente = new Cliente();
2 cliente.Id = "1";
3 cliente.Nombre = "nombre";
4 cliente.Email = "email";
5 cliente.Pais = "pais";
6 session.Save(customer);
```

Fuente: Elaborado por autores.

Como han observado, se ha reducido considerablemente el código. Pero lo más importante de todo, es que, imaginen que ahora se modifica la tabla de nuestra base de datos y se añade un campo más como por ejemplo el apellido de nuestro cliente. En los dos casos, tendríamos que añadir a nuestra clase Cliente la propiedad correspondiente al apellido, pero, si no utilizas un ORM te tocará revisarte todas las sentencias INSERT, SELECT y UPDATE para introducir dicho campo en cada una de ellas. En cambio, si

utilizas un ORM, lo único que tendrás que hacer será añadir la propiedad a la clase correspondiente.

La sentencia save seguirá siendo la misma, el ORM se encargará de modificar los INSERT, SELECT y UPDATE por ti.

Además de lo que hemos visto hasta el momento, gracias a los ORM nos abstraemos del motor de base de datos que estamos utilizando, es decir si en algún momento queremos cambiar de base de datos, nos resultará sumamente sencillo. En algunos casos con solo cambiar un par de líneas de código, estaremos cambiando de motor de base de datos. Existen varios ORM en el mercado. Todo dependerá del lenguaje de programación que estemos utilizando y de nuestras necesidades.



TEMA 4: PRUEBAS UNITARIAS



Ariane 5 → Lanzado por primera vez el 4 de junio de 1996.

Fuente: Elaborado por autores.



36.7 segundos después explotó.

Fuente: Elaborado por autores.

Motivo: Fallo software. La programación no se había probado lo suficiente.

Las pruebas dentro del desarrollo de software:

- Garantizan la calidad del software.
- Garantizan la satisfacción de los requisitos.
- Ahorran tiempo y recurso en el desarrollo.

El objetivo de las pruebas es localizar las fallas, para subsanarlas, el mayor número de deficiencias lo antes posible.

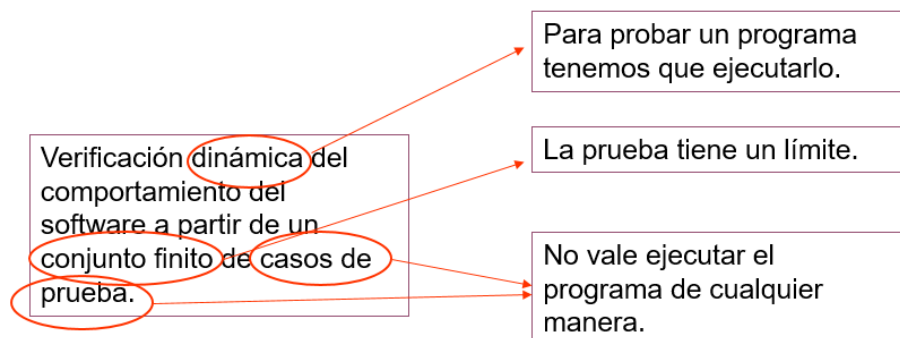
DEFINICIÓN DE PRUEBA

Verificación dinámica del comportamiento del software a partir de un conjunto finito de casos de prueba.

COMPONENTES DE LA PRUEBA

Validación: proceso de evaluar un sistema o componente durante o al final del proceso de desarrollo para determinar si satisface los requisitos especificados.

Verificación: proceso de evaluar un sistema o componente para determinar si los productos de una determinada fase satisfacen las condiciones impuestas al comienzo de la fase.



Fuente: Elaborado por autores.

ELEMENTOS DE UNA PRUEBA

Consta al menos de tres elementos:

Acciones

Configurar partida:

- El sistema muestra un formulario con todos los valores configurables.
- El jugador introduce modifica los valores.
- El sistema comprueba la validez de los valores y vuelve a la pantalla principal.

Valores de prueba

Configuración:

Intentos = 5
 Valores = 8
 Repetidos = cierto

Resultado



Fuente: Elaborado por autores.

Ejemplos de pruebas

¿Funciona el teléfono?.

Valores de prueba	Acciones	Resultado esperado
123-45-67-89	1. Descolgar auricular. 2. Marcar número de Pepote. 3. Esperar contestación.	(Pepote): "Digameeee".

Fuente: Elaborado por autores.

¿Me está bien esta camisa?

Valores de prueba	Acciones	Resultado esperado
Mi cuerpo.	1. Ponerme la camisa. 2. Abrochámela. 3. Moverme un poco. 4. Mirarme al espejo. Cuidado con la etiqueta o con arrugarla por si hay que devolverla	Elegancia y confort.

Fuente: Elaborado por autores.



```
public int suma(int a, int b)
{
    return a + b;
}
```

¿Qué casos de prueba podemos escribir?.

Valores de prueba	Acciones	Resultado esperado
???	Suma(a, b)	???



Los casos de prueba son finitos (y cuantos menos, mejor).



```
public int suma(int a, int b)
{
    return a + b;
}
```

¿Qué casos de prueba podemos escribir?.

Valores de prueba	Acciones	Resultado esperado
0, 0	Suma(a, b)	0
0, b = no 0	Suma(a, b)	b
3, 4	Suma(a, b)	7
-2, -8	Suma(a, b)	-10
-3, 6	Suma(a, b)	3
Integer.MAX_VALUE, 6	Suma(a, b)	-2147483643

Y algunas permutaciones más.

NIVELES DE PRUEBA

➤ FASES DE LA PRUEBA

- Diseño de las pruebas (¿técnicas?)
- Generación de casos de prueba (¿datos de prueba?)
- Definición del procedimiento de la prueba (¿cómo? ¿dónde?)
- Ejecución de la prueba
- Informe de la prueba (¿resultados?)

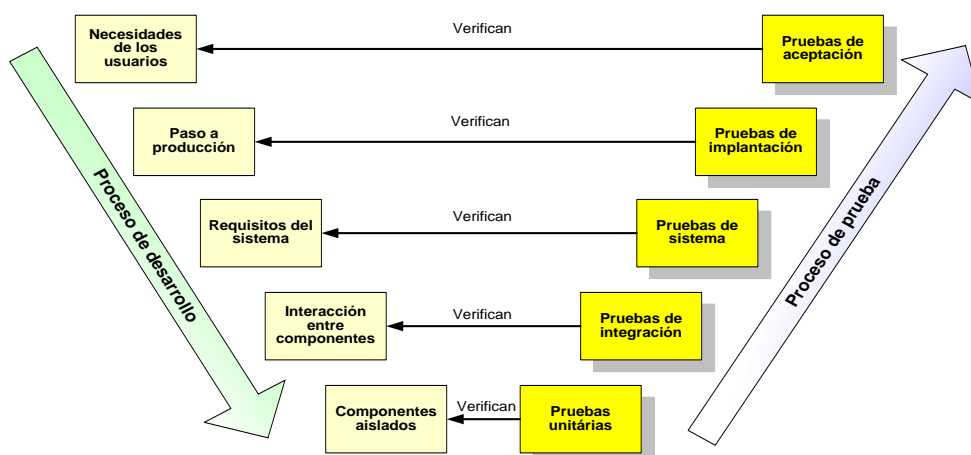
➤ TÉCNICAS DE PRUEBA

- Pruebas estructurales o de Caja Blanca.
- Pruebas funcionales o de caja negra.

➤ ESTRATEGIAS DE PRUEBA

- Pruebas unitarias.
- Pruebas de integración.
- Pruebas de sistema.
- Pruebas de aceptación.
- Pruebas de regresión.

GRAFICACIÓN DE NIVELES

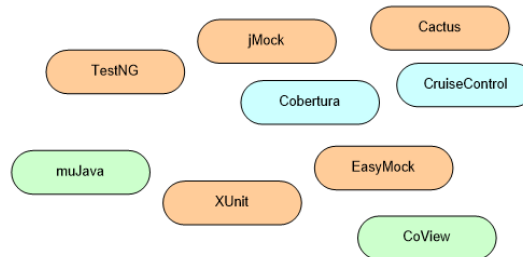


Fuente: Elaborado por autores.

PRUEBAS UNITARIAS

Cuando: Durante la construcción del sistema.
Objetivo: Prueban el diseño y el comportamiento de cada uno de los componentes una vez construidos.

Herramientas:



Fuente: Elaborado por autores.

TÉCNICAS IMPLEMENTADAS

- Análisis de caminos.
- Partición de categorías.
- Mutaciones.
- Algoritmos genéricos.

PRUEBAS DE INTEGRACIÓN

Cuando: Durante la construcción del sistema
Objetivo: Comprueban la correcta unión de los componentes entre sí a través de sus interfaces, y si cumplen con la funcionalidad establecida

Herramientas:

Las mismas, vamos sustituyendo los mocks por las clases reales y, si es necesario, escribiendo más casos de prueba.

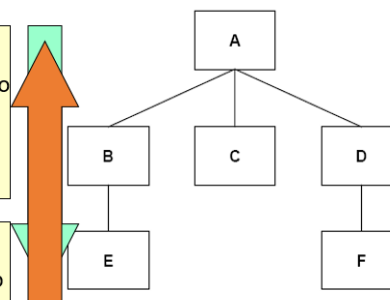
TÉCNICAS IMPLEMENTADAS

Arriba abajo:

El primer componente que se prueba es el primero de la jerarquía (A). Una de las ventajas es que las interfaces entre los distintos componentes se prueban en una fase temprana y con frecuencia.

Abajo a arriba:

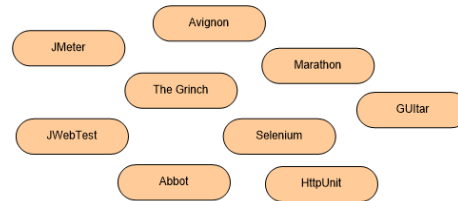
Se prueban primero los componentes de más bajo nivel (E, F). Este tipo de enfoque permite un desarrollo más en paralelo, pero presenta mayores dificultades a la hora de planificar y de gestionar.



PRUEBAS DE SISTEMA

Cuando: Durante la construcción del sistema (partes completas).
Objetivo: Prueban a fondo el sistema, comprobando su funcionalidad e integridad globalmente, en un entorno lo más parecido posible al entorno final de producción.

Herramientas:



Técnicas: probar el sistema como lo hará el usuario final.

PRUEBAS DE IMPLANTACIÓN

Cuando: Durante la implantación en el entorno de producción.
Objetivo: Comprueban el correcto funcionamiento del sistema dentro del entorno real de producción (rendimiento, copias de seguridad, etc).

Herramientas:

Las pruebas se vuelven a ejecutar en el entorno real de producción y se añaden nuevas pruebas.

PRUEBAS DE ACEPTACIÓN

Cuando: Después de la implantación en el entorno de producción.
Objetivo: Verifican que el sistema cumple con todos los requisitos indicados y permite que los usuarios del sistema den el visto bueno definitivo.

Herramientas:

Las mismas. Las pruebas se vuelven a ejecutar en el entorno real de producción y se añaden nuevas pruebas.

PRUEBAS DE REGRESIÓN

Cuando: Durante el mantenimiento del sistema.
Objetivo: Comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

Herramientas:

Las mismas.

CARACTERÍSTICAS DE UNA BUENA PRUEBA

- Ha de tener una alta probabilidad de encontrar un fallo. Cuanto más, mejor.
- No debe ser redundante. Si ya funciona, no lo probamos más.
- Debe ser la “mejor de la cosecha”. Si tenemos donde elegir, elegimos la mejor.
- No debería ser ni demasiado sencilla ni demasiado compleja. Si es muy sencilla no aporta nada, si es muy compleja a lo mejor no sabemos lo que ha fallado.

AUTOMATIZACIÓN DE LAS PRUEBAS

Algunas palabras que hemos escuchado hablando de pruebas.

“No sirven para nada”

“Pérdida de tiempo”

“Imposibles de mantener”

“Engaño”

“Descartadas al primer cambio”

“Moda”

¿Por qué?.

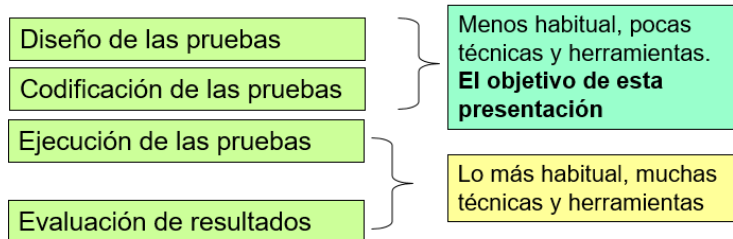
Las pruebas son polémicas:

- No son parte de la solución.
- No se las entregamos a nuestros clientes.
- Incluso pueden darles a nuestros clientes una mala impresión.
- Nuestros clientes no nos las pagan.
- Difíciles de mantener.
- Sin embargo, las pruebas son imprescindibles.
- La automatización nos permite reducir costes.

¿Qué significa automatizar?.

En este caso concreto: realizar de manera **automática** mediante **herramientas software**.

¿Qué podemos automatizar?.



VENTAJAS DE LA AUTOMATIZACIÓN

- Mayor rapidez de ejecución.
- Menos recursos.
- Evitamos pruebas obsoletas.
-



TEMA 5: SERVERLESS COMPUTING

Es un modelo o servicio de computación en la nube totalmente autogestionado en el que el proveedor de la nube ejecuta el servidor y lo administra asignando y adaptando los recursos de éste según las necesidades del usuario o de la empresa.

Las estructuras sin servidor permiten a sus usuarios concentrarse en la actividad de su empresa sin tener que preocuparse por ningún servidor. Es decir, no hay servidores que mantener, ni sistemas operativos que cuidar, ni software que administrar, ni hardware que actualizar. Esto simplemente significa que tú puedes dedicarte 100% a la actividad de tu negocio, sin preocuparte por nada más.

BENEFICIOS

- **Coste:** la ventaja principal de utilizar informática sin servidor es el hecho de que ésta te proporciona servicios solo por lo que realmente utilizas. Esto significa, que tu solo pagarás por lo que usas, ni más ni menos.

- **Mantenimiento:** como ya hemos comentado anteriormente, estos sistemas no presentan servidores o sistemas operativos que mantener. No tenemos que administrar ningún servidor ni siquiera tenemos que instalar ningún sistema operativo o software de soporte.
- **Escalamiento fácil y eficiente:** las aplicaciones sin servidor se pueden escalar automáticamente o como máximo con unos pocos clics para elegir la capacidad deseada.
- **Alta disponibilidad:** las aplicaciones sin servidor tienen disponibilidad incorporada. Por lo tanto, no necesitas tener una infraestructura especializada para que las aplicaciones estén altamente disponibles. Todo esto está disponible para tu empresa por defecto.
- **Autonomía para la empresa:** La informática sin servidor brinda a las organizaciones la libertad de concentrarse en sus ofertas comerciales principales y olvidarse de los problemas de servidores de bajo nivel. Por lo tanto, los trabajadores y la alta dirección pueden utilizar el dinero y la oportunidad para desarrollar ofertas competitivas, que pueden ayudar a la organización a superar a los competidores.

EN RESUMEN. SERVERLESS ES:



Abstracción de
los servidores



Event-driven/
escalado
instantáneo



Micro-billing

Fuente: Elaborado por autores.

BENEFICIOS



Gestión de
aplicaciones, no
servidores



Menos tareas
de DevOps



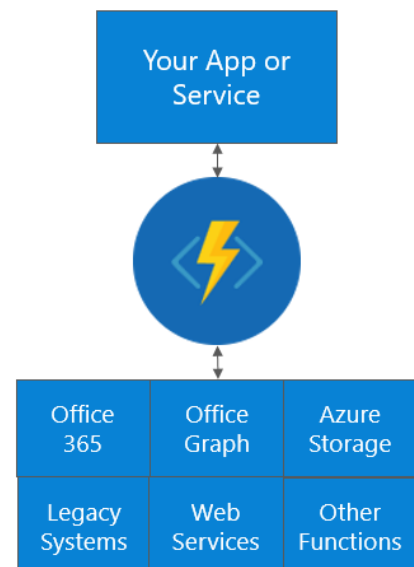
Time
to market
más rápido

Fuente: Elaborado por autores.

ESCENARIOS DE AZURE FUNCTIONS

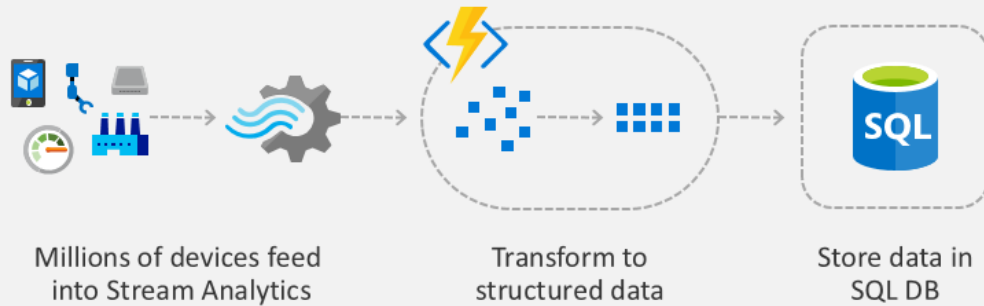
Servicio que ofrece ejecución de cómputo sobre demanda por fragmentos de código:

- Timer-based processing.
- Azure service event processing.
- SaaS event processing.
- Serverless web application architectures.
- Serverless mobile back ends.
- Real-time stream processing.
- Real-time bot messaging.



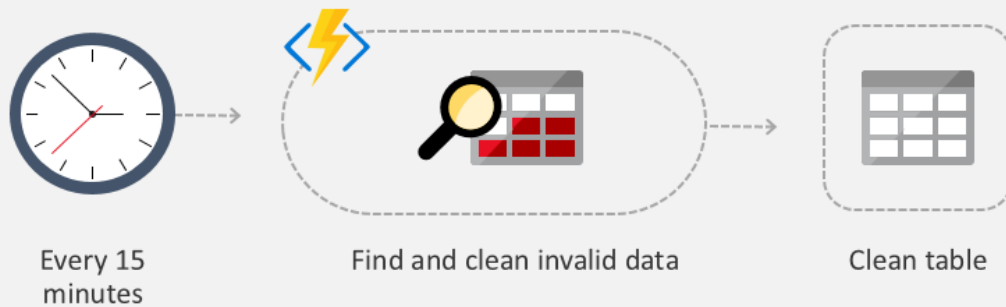
ESCENARIOS PARA SERVERLESS

Real-time stream processing



Fuente: Elaborado por autores.

Timer-based processing

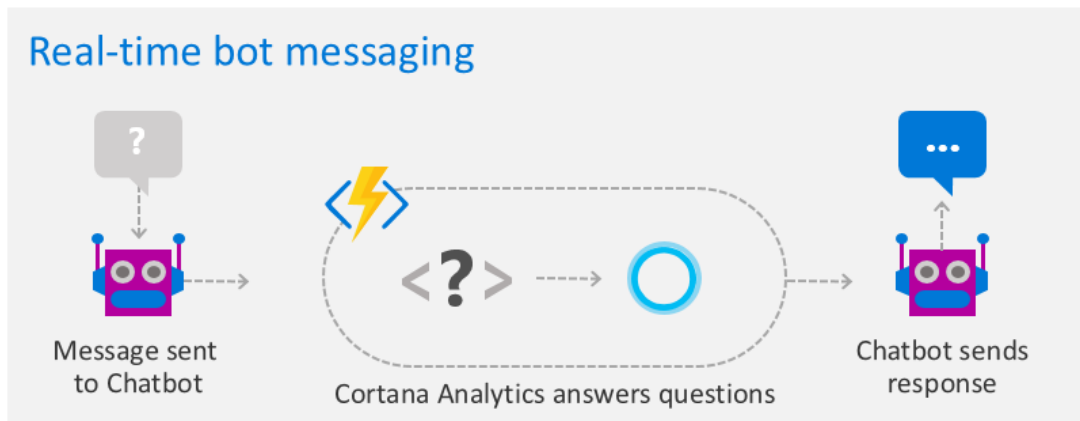


Fuente: Elaborado por autores.

Backends (Mobile/IoT/Web)



Fuente: Elaborado por autores.



Fuente: Elaborado por autores.

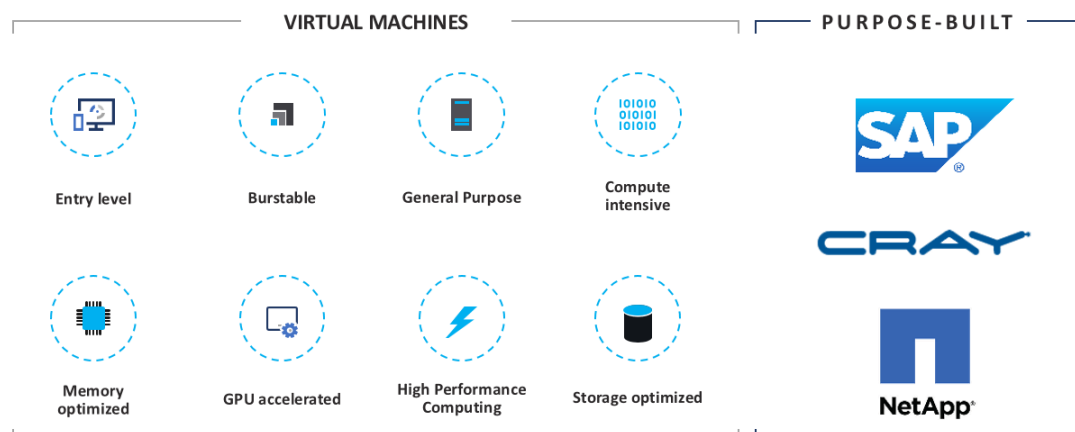
MÁQUINAS VIRTUALES EN AZURE



Fuente: Elaborado por autores.





- Máquinas Virtuales con Windows y Linux.
- Se puede escalar de 1 a 1000 instancias.
- Facturación por minuto.
- Permite templates, extensiones, etc.
- Acceso a redes virtuales + backup + monitoreo.

OPCIONES DE CÓMPUTO PARA TODA CARGA DE TRABAJO




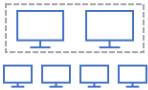
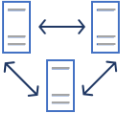
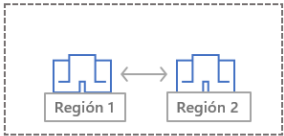
Fuente: Elaborado por autores.

ALMACENAMIENTO EN DISCO

				
	Standard HDD	Standard SSD	Premium SSD	Ultra SSD <small>Preview</small>
	Almacenamiento de bajo costo	Performance consistente	Alta performance	Latencia debajo del milisegundo
TAMAÑO	32TiB	32TiB	32TiB	64TiB
IOPS	2,000	2,000	20,000	160,000
BANDWIDTH	500 MBps	500 MBps	750 MBps	2,000 MBps

Fuente: Elaborado por autores.

ARQUITECTURA PARA ALTA DISPONIBILIDAD EN AZURE

VM SLA 99.9%	VM SLA 99.95%	VM SLA 99.99%	Regiones 54
			
Single VM	Availability sets	Availability zones	Site Recovery & Region pairs

AZ disponibles en EEUU, Europa y Asia, con más regiones en el futuro

Fuente: Elaborado por autores.

CONTENEDORES O CONTAINERS

¿Qué son?

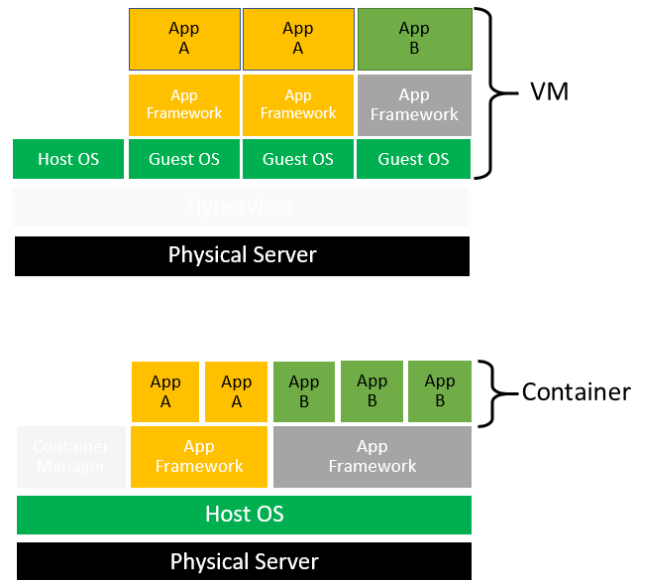
Unidad de despliegue
 Ambiente aislado
 Colección de recursos aislados

¿Beneficios?

Inicio casi instantáneo
 Ejecución repetida y segura

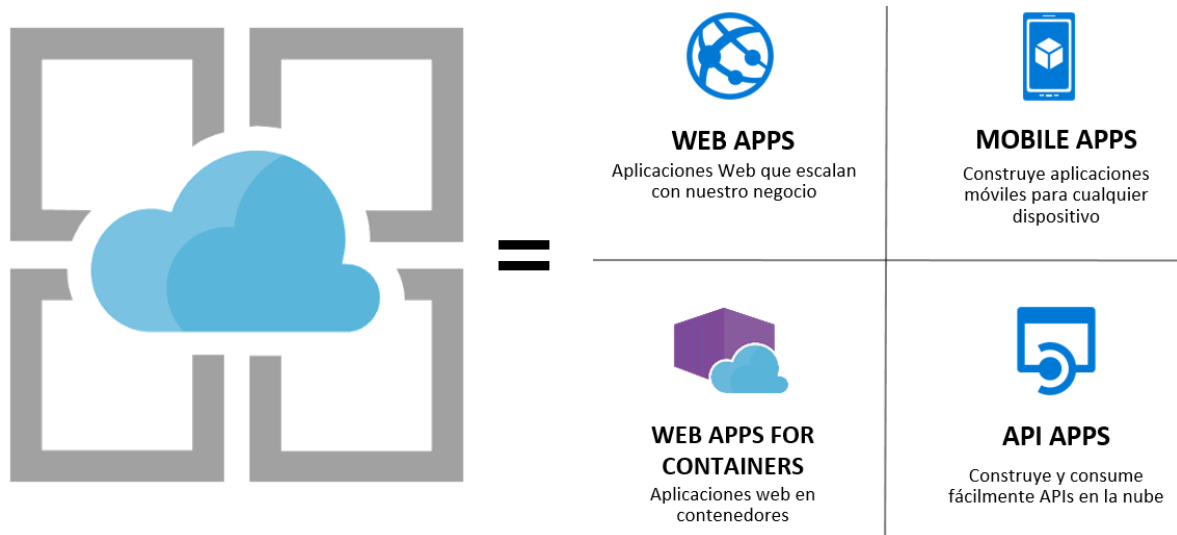
¿Escenarios?

Dev/Test
 Genial para microservicios
 Puede alojar apps monolíticas sin problemas



Fuente: Elaborado por autores.

AZURE APP SERVICES



Fuente: Elaborado por autores.

BENEFICIOS DE APP SERVICES

- Parcheo automático del SO.
- Seguridad Empresarial.
- Scale out / in automático.
- Balanceador incorporado.
- Soporta varios languages y Plataformas.
- Facilita el Despliegue / Entrega continua.

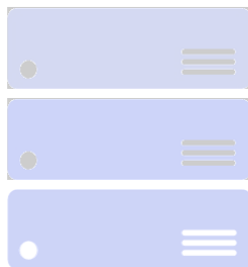
- Permite trabajos en segundo plano con WebJobs.
- Lenguajes: .NET, PHP, Node.js, Java, Ruby, Python.
- Webjobs.
- CI con GitHub, BitBucket, Azure DevOps, etc.
- SDKs para app mobile.
- Slots para ambientes.

MICROSERVICIOS

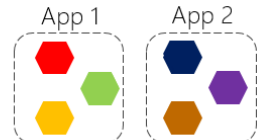
- ✓ Encapsula un escenario.
- ✓ Son desarrolladas por pequeños grupos.
- ✓ Cada Servicio puede ser desarrollado con distintos lenguajes y frameworks.
- ✓ Cada Servicio puede ser versionado y escalado por separado.
- ✓ Existe interacción entre los servicios por HTTP.
- ✓ Cada Servicio tiene una única URL independiente.
- ✓ Se pueden aislar errores, en caso de falla.

DIFERENCIA ENTRE APLICACIONES MONOLÍTICAS Y MICROSERVICIOS

- Una aplicación monolítica contiene funcionalidades específica por dominio y está normalmente dividida por capas funcionales como web, negocio y datos
- Escala clonando la app en multiples servers/VMs/Containers



- Una aplicación basada en Microservicios separa funcionalidad en pequeños servicios independientes unos de otros con su propia db.
 - Escala desplegando cada Servicio independientemente y creando instancias entre servers/VMs/containers



Fuente: Elaborado por autores.



Videos que permitirán complementar lo aprendido durante las sesiones.

MVC → https://youtu.be/JWi4_8_d-RM

ORM → <https://youtu.be/1QCom48lw0Y>

SERVERLESS → <https://youtu.be/-ci7EwXaIJg>

PRUEBAS UNITARIAS → <https://youtu.be/YuRdaR6wwWU>

Referencias Bibliográficas

Brown, C. (1988). Human-Computer Interface Design Guidelines. Ablex Publishing Corp, Norwood.

Encalada, R., Molina, D. & Ruiz, C. (2019). Tipos de Servidores. Recuperado desde: https://www.ecotec.edu.ec/material/material_2019D1_COM270_01_116736.pdf

Lynch, P. (1999). Web Style Guide: Basic Design Principles for Creating Web Sites. Yale Press.

Martínez, S. & Cueva, F. (2020). Estándares de la Interacción Humano Computador. Nuevos modelos y tendencias en la Web. Recuperado desde: <http://di002.edv.uniovi.es/~cueva/asignaturas/doctorado/2004/1FundamentosIHM>

Saz, M. (2014). Introducción al patrón MVC. Recuperado desde: <https://www.codigonexo.com/wp-content/uploads/2014/06/Curso-completo-MVC.pdf>

Valle, L. (2019). Definición, características y ejemplo práctico de ORM. Recuperado desde: <https://programarfacil.com/blog/que-es-un-orm/>