Arquitectura y Buenas Prácticas de Desarrollo Seguro

Diego Esteban Cortes Flechas Id 145749
Wilmer Fula Id 251825
Emilio Jose Olaya Soche Id 408674

Diseño de algoritmos
Docente: Juan Carlos Muñoz Vera

CORPORACIÓN UNIVERSITARIA MINUTO DE DIOS
FACULTA DE INGENIERÍA
Bogotá, D.C.
2023

Contenido

Construcción de Código

Archivo fuente

Cada archivo fuente debe contener una única clase o interfaz pública. El nombre del archivo tiene que coincidir con el nombre de la clase. Cuando existan varias clases privadas asociadas funcionalmente a una clase pública, podrán colocarse en el mismo archivo fuente que la clase pública. La clase pública debe estar situada en primer lugar dentro del archivo fuente.

En todo archivo fuente distinguimos las siguientes secciones:

1. Comentarios de inicio.
2. Sentencia de paquete.
3. Sentencias de importación.
4. Declaraciones de clases e interfaces.

Comentarios de inicio

Todo archivo fuente debe comenzar con un comentario que incluya el nombre de la clase, información sobre la versión del código, la fecha y el copyright. El copyright indica la propiedad legal del código, el ámbito de distribución, el uso para el que fue desarrollado.

```
// Transcode.cs 0.9.0 31/10/2020
// Copyright 2020, Engineers SA. Todos los derechos reservados.
// Autor: Diego, Wilmer, Emilio. (Sentencias de Importación)

/**
* Este script permite controlar los mensajes de…
*/
```

Sentencias de importación

Entre el nombre del autor del código y la descripción de este, se incluirán las sentencias de importación de los paquetes necesarios. Esta importación de paquetes obligatorios seguirá el siguiente orden:

1. Paquetes de Php, C# o Framework.

2. Paquetes de utilidades no pertenecientes a Php, C# o Framework.

3. Paquetes de la aplicación.

```
// Transcode.cs 0.9.0 31/10/2020
// Copyright 2020, Engineers. Todos los derechos reservados.
// Autor: equipo de trabajo Diseño de Algoritmos

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.SceneManagement;
using System.IO;

/**
* Este script permite controla los mensajes de…
*/
```

Declaración de clases e interfases

Elementos de declaración de una clase / interfaz	Descripción
Comentario de documentación de la clase/interfaz /** *… */	Permite describir la clase/interfaz desarrollada. Necesario para generar la documentación del código mediante Doxygen.
Comentario de implementación de la clase/interfaz, si es necesario Para una línea: // ... Para más de una línea: /* ...*/	Este comentario incluye cualquier información que no pueda incluirse en el comentario de documentación de la clase/ interfaz.
Variables de clase (estáticas)	En primer lugar las variables de clase públicas (public), después las protegidas (protected), posteriormente las de nivel de paquete (sin modificador), y por último las privadas (private).
Variables de instancia	Primero las públicas (public), después las protegidas (protected), luego las de nivel de paquete (sin modificador), y finalmente las privadas (private).
Métodos	Deben agruparse por funcionalidad en lugar de agruparse por ámbito o accesibilidad. Por ejemplo, un método privado puede estar situado entre dos métodos públicos. El objetivo es desarrollar código fácil de leer y comprender.

Sangría

Como norma general se establecen 4 espacios como unidad de sangría, se puede usar plugin para el IDE o editor para dar formato.

Longitud de línea

La longitud de línea no debe superar los 80 caracteres por motivos de visualización e impresión.

División de línea

Cuando una expresión ocupe más de una línea, esta se podrá romper o dividir en función de los siguientes criterios:

1. Tras una coma.
2. Antes de un operador.
3. Se recomienda las rupturas de nivel superior a las de nivel inferior.
4. Alinear la nueva línea con el inicio de la expresión al mismo nivel que la línea anterior.
5. Si las reglas anteriores generan código poco comprensible, entonces estableceremos tabulaciones de 8 espacios.

Comentarios

Distinguimos dos tipos de comentarios: los comentarios de implementación y los de documentación.

Comentarios de implementación

Se utilizan para describir el código ("el cómo"), y en ellos se incluye información relacionada con la implementación, tales como descripción de la función de variables locales, fases lógicas de ejecución de un método, captura de excepciones, etc.

Pueden ser en bloque o en línea, en bloque permiten la descripción de ficheros, clases, bloques, estructuras de datos y algoritmos, en línea son comentarios cortos localizados en una sola línea y tabulados al mismo nivel que el código que describen. Si ocupa más de una línea se utilizará un comentario de bloque. Deben estar precedidos por una línea en blanco.

Ejemplo de comentario en bloque:

```
/*
Esta función se llama solo una vez al arrancar el app.
Se bajan los diálogos y se salvan en un documento
para leerlo en caso de no haber internet.
*/
```

Ejemplo de comentario de una sola línea:

```
$contador = 4 + 10; // Inicialización del contador.
```

Comentarios de documentación

Se utilizan para describir la especificación del código, desde un punto de vista independiente de la implementación, de forma que pueda ser consultada por desarrolladores que probablemente no tengan acceso al código fuente.

Declaraciones

Las declaraciones deben ser por línea.

```
$idCuenta; // Identificador de la cuenta de usuario
```

Inicialización

Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente.

```
$idCuenta = 1;
$tipoCuentas = [ "Checking", "Savings", "Credit" ];
```

Localización

Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen, y nunca en el momento de su uso y se debe evitar el uso de declaraciones que oculten a otras declaraciones de ámbito superior.

```
$contador = 0; // Inicio del método
public function unMetodo() {
   if (condition) {
      $contador = 2; // ¡¡ EVITAR !! ...
   }
}
```

Declaración de clases / interfaces

1. No incluir ningún espacio entre el nombre del método y el paréntesis inicial del listado de parámetros.

2. El carácter inicio de bloque ("{") debe aparecer al final de la línea que contiene la sentencia de declaración.

3. El carácter fin de bloque ("}") se sitúa en una nueva línea tabulada al mismo nivel que su correspondiente sentencia de inicio de bloque, excepto cuando la sentencia sea nula, en tal caso se situará detrás de "{".

4. Los métodos se separarán entre sí mediante una línea en blanco.

```
class ClaseEjemplo extends Object {

   $variable1;
   $variable2;

   public  function__construct(){
      $variable1 = 0;
      $variable2 = 1;
   }
   ...
}
```

Sentencias

Las sentencias pertenecientes a un bloque de código estarán tabuladas un nivel más a la derecha con respecto a la sentencia que las contiene.

El carácter inicio de bloque "{" debe situarse al final de la línea que inicia el bloque. Y el carácter final de bloque "}" debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer carácter de dicho bloque.

Todas la sentencias de un bloque deben encerrarse entre llaves “{...}", aunque el bloque conste de una única sentencia. Esta práctica permite añadir código sin cometer errores accidentalmente al olvidar añadir las llaves.

```
if (condicion) {
   $variable++;
}
```

Espacios en blanco

1. Se utilizarán espacios en blanco entre una palabra clave y un paréntesis. Esto permite que se distingan las llamadas a métodos de las palabras clave.

```
while (true) {
   ...
}
```

2. Después de cada coma en un listado de argumentos.

```
$objeto->UnMetodo(a, b, c);
```

3. Para separar un operador binario de sus operandos. Nunca se utilizarán espacios entre los operadores unitarios (p.e., "++" o "--") y sus operandos.

```
$a += $b + $c;
$a = ($a + $b) / ($c + $d);

$contador++;
```

4. Para separar las expresiones incluidas en la sentencia "for".

```
for (expresion1; expresion2; expresion3)
```

Nomenclatura de identificadores

Es necesario usar nombres descriptivos y claros. Deben ser legibles y poder identificar si es una constante, una variable, una clase o un paquete y evitar codificaciones complejas.

Clases e interfases

Los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúsculas. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúsculas.

Los nombres serán simples y descriptivos. Debe evitarse el uso de acrónimos o abreviaturas, salvo en aquellos casos en los que dicha abreviatura sea más utilizada que la palabra que representa (URL, HTTP, etc.).

Las interfaces se nombrarán siguiendo los mismos criterios que los indicados para las clases. Como norma general toda interfaz se nombrará con el prefijo "I" para diferenciarla de la clase que la implementa (que tendrá el mismo nombre sin el prefijo “I").

```
class Account
class AccountService
interface IAccountService
```

Métodos

Los métodos deben ser verbos escritos en minúsculas. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúsculas.

Los métodos deben ser cortos, hacer una única funcionalidad y mantenerse dentro del mismo nivel de abstracción. Se deberá reducir al mínimo el número de argumentos y se deberá eliminar toda la duplicidad.

```
public function AddUser(User $user);
public function DeleteUser(User $user);
public function UpdateUser(User $user);
```

Variables

Las variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas.

Los nombres de variables deben ser cortos y sus significados tienen que describir con claridad la función que desempeñan en el código.

Debe evitarse el uso de nombres de variables con un sólo carácter, excepto para variables temporales, deben ser legibles y evitar codificaciones complejas

```
$idUser;
$account;
$limit;
```

Constantes

Todos los nombres de constantes tendrán que escribirse en mayúsculas. Cuando los nombres de constantes sean compuestos las palabras se separarán entre sí mediante el carácter de subrayado “_”.

```
LONGITUD_MAXIMA;
LONGITUD_MINIMA;
```

Visibilidad de atributos de instancia y de clase

Los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas, en tales casos serán declarados como protegidos.

El acceso a los atributos de una clase se realizará por medio de los métodos "get" y "set" correspondientes (propiedades), incluso cuando el acceso a dichos atributos se realice en los métodos miembros de la clase.

```
class Account {
   private $id;
   private $balance;
   …
  public function updateBalance(Transaction $transaction) {
        this.setId(transaction.getIdUser());
        ….
   }
    ...
}
```

Referencias a miembros de una clase

Evitar el uso de objetos para acceder a los miembros de una clase (atributos y métodos estáticos). Utilizaremos en su lugar el nombre de la clase.

```
MetodoUtilidad(); // Acceso desde la propia clase estática
ClaseUtilidad.MetodoUtilidad(); // Acceso común desde cualquier clase
```

Asignación sobre variables

Se deben evitar las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia, ya que dichas sentencias suelen ser difíciles de leer.

```
$a = $b = $c = 2; // Evitar
```

No utilizar el operador de asignación en aquellos lugares donde sea susceptible de confusión con el operador de igualdad.

Ejemplo incorrecto

```
if ((c = d++) == 0) { }
```

Ejemplo correcto

```
$c = d++;
```

```
if ($c == 0) { }
```

No utilizar asignaciones embebidas o anidadas.

Ejemplo incorrecto

```
$c=($c=3)+4+$d; //Evitar
```

Ejemplo correcto

```
$c = 3;
$c = $c + 4 + $d;
```

Paréntesis

Utiliza paréntesis en expresiones que incluyan distintos tipos de operadores para evitar problemas de precedencia de operadores.

Ejemplo incorrecto

```
if (w == x && y == z) // INCORRECTO
```

Ejemplo correcto

```
if ((w == x) && (y == z)) // CORRECTO
```

Valores de retorno

Los valores de retorno tendrán que ser simples y comprensibles, de acuerdo con el propósito y comportamiento del objeto en el que se utilicen.

Ejemplo incorrecto

```
public function esProgramador(Empleado emp) {
  if (emp.getRol().equals(ROL_PROGRAMADOR)) {
      return true;
  } else {
     return false;
   }
}
```

Ejemplo correcto

```
public boolean esProgramador(Empleado emp) {
  boolean esUnProgramador = false;
  if (emp.getRol().equals(ROL_PROGRAMADOR)) {
     esUnProgramador = true;
  }
  return esUnProgramador;
}
```

Toda expresión compuesta, por uno o más operadores binarios, situada en la parte condicional del operador ternario deberá ir entre paréntesis.

```
(x >= y) ? x : y;
```

Desarrollo Seguro

Para asegurar que los equipos de desarrollo cumplen con los estándares de desarrollo seguro requeridos se puede establecer lineamientos de prácticas a cumplir como las siguientes:

1. Leer, entender y aplicar las prácticas propuestas por las metodologías: OWASP, NIST, SANS, SAMM, BSIMM y MICROSOFT.

2. Conocimiento en seguridad, y capacitación en Desarrollo Seguro.

Exigir que todas las personas encargadas de programación en los equipos de desarrollo tengan mínimo un curso en Desarrollo de Software Seguro y se certifiquen, ejemplo en Udemy.


udemy

CERTIFICATE OF COMPLETION

Desarrollo de Software Seguro

Instructors Yury Zavaleta

Andres Martinez

Date April 26, 2022

Length 6.5 total hours


udemy

CERTIFICADO DE FINALIZACIÓN

Desarrollo de Software Seguro

Instructores Yury Zavaleta

Rafael González Delanoy

Fecha 27 de Abril de 2022

Duración 6.5 horas en total


udemy

CERTIFICADO DE FINALIZACIÓN

Desarrollo de Software Seguro

Instructores Yury Zavaleta

Oscar Quintero

Fecha 26 de Abril de 2022

Duración 6.5 horas en total


udemy

CERTIFICADO DE FINALIZACIÓN

Desarrollo de Software Seguro

Instructores Yury Zavaleta

Walberto Marrugo Ortega

Fecha 25 de Abril de 2022

Duración 6.5 horas en total

3. Análisis de amenazas. Revisando cumplimiento de OWASP y CWE/SANS y ejecutando un análisis STRIDE (Spoofing - Tampering Repudiation - Information disclosure - Denial of service - Elevation of privilege).

4. Verificar requerimientos de seguridad en autenticación y contraseñas, autorización, generación de logs, manejo de sesiones, validación de datos, canonización o codificación y cifrado.

5. Elaborar matriz RACI (Responsible - Accountable - Consulted - Informed) para el ambiente de desarrollo.

6. Documentar proceso de control de cambios y alinearlo con el proceso de control de cambios interno del cliente.

Análisis de Código Estático

7. Aparte de estas prácticas, una vez generado el código fuente, tanto el de los micro-servicios como el de la aplicación móvil, hay dos formas de asegurarnos de la seguridad del mismo por medio de dos tipos de análisis. El análisis de código estático que se hace directamente sobre el código fuente, y el escaneo de vulnerabilidades que se hace sobre el servicio o aplicación en estado funcional.

8. Para el análisis de código estático se puede usar herramientas como Roslyn, que usa los analizadores de errores .NET tambien esta la herramienta llamada SonarLint, permite tener los errores y advertencias en tiempo real en el editor de texto con información relevante sobre el posible problema y la forma de arreglarlo.

Análisis de Vulnerabilidades

9. Adicionalmente se puede usar herramientas como SonarCloud para detectar vulnerabilidades. Las vulnerabilidades se clasifican en rojas son consideradas críticas o de alto nivel, las vulnerabilidades anaranjadas son consideradas de nivel mediano y las azules de nivel bajo o informativas.

10. Para aplicaciones móviles existen herramientas como MobSF que detecta vulnerabilidades sobre los archivos ejecutables de Android e iOS archivo como por ejemplo .apk o .ipa

Diagrama E.R.

En este diagrama se muestra la relación que hay entre las tablas de usuarios, productos, rol, compras e inventario en donde cada tabla tiene diferentes campos los cuales son los necesarios para realizar el desarrollo del software, realizando las apis tipo get, post, put, delete dependiendo la necesitad de la funcionalidad que tenga el front, este diagrama muestra una relación de uno a muchos entre la tabla de usuarios y compras, una relación de 1 a 1 entre usuarios y rol ya que cada usuario es único, de 1 a muchos entre compras e inventario y de 1 a muchos entre inventario y productos

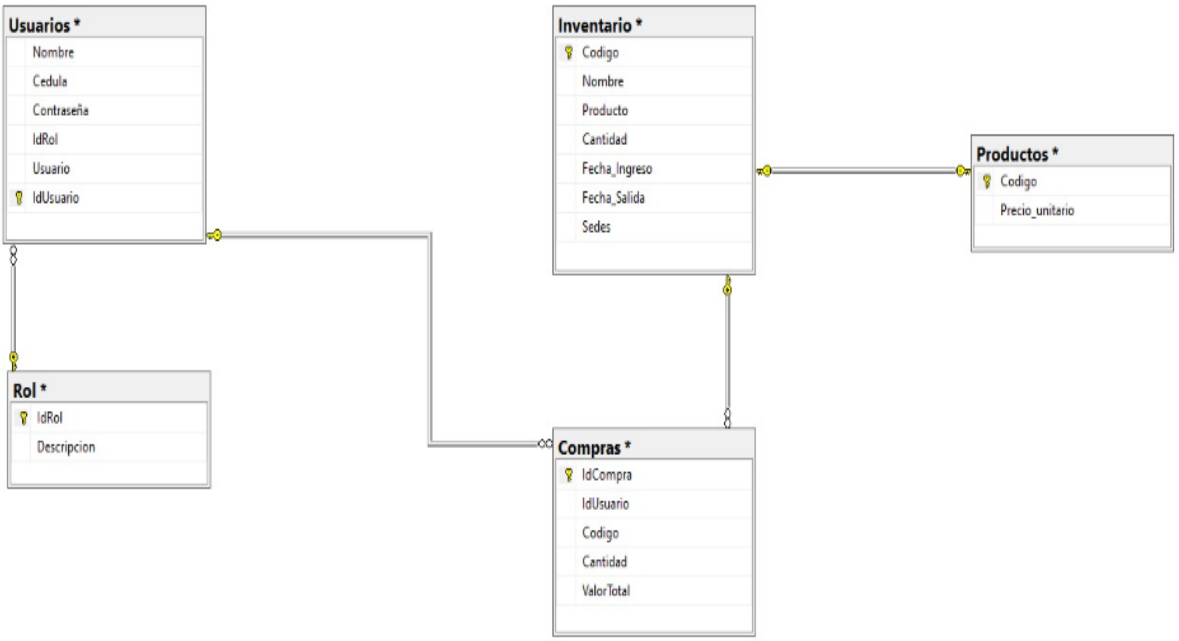


Diagrama de secuencia

En este diagrama se muestra el flujo de la toma de un pedido, consulta de productos, listado y resultado de las consultas, para poder realizar el pedido y registrarlo.

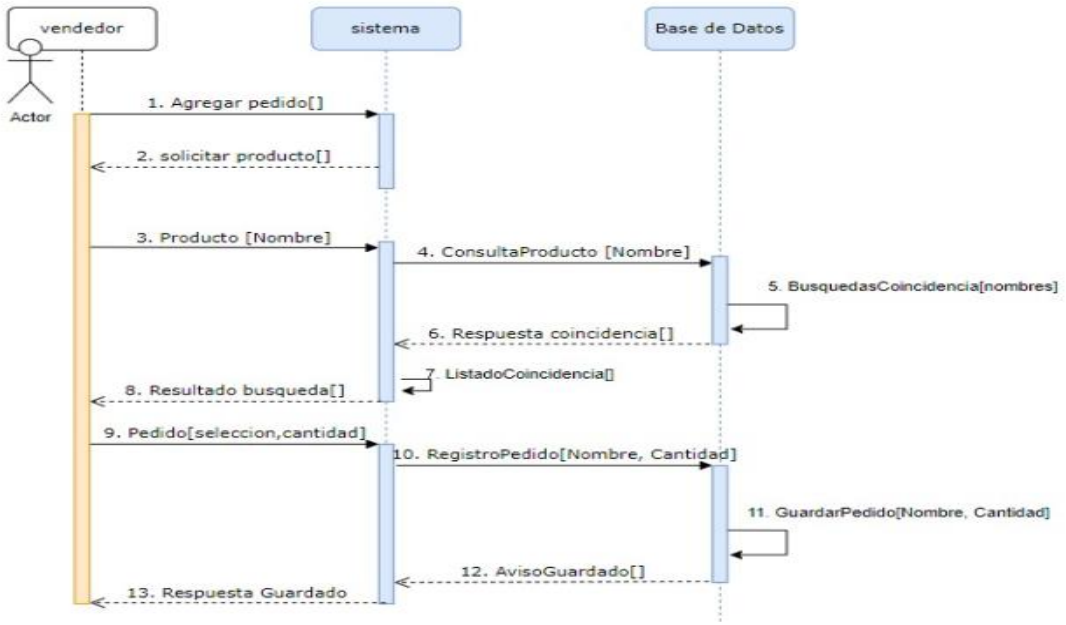


Diagrama de despliegue

Este diagrama muestra cómo los diferentes componentes del sistema interactúan entre sí y cómo se distribuyen en la red. La aplicación web y las APIs se alojan en el servidor y se comunican con las bases de datos de inventario a través de protocolos HTTP/HTTPS. Las Sedes se conectan al servidor a través de una red LAN y pueden acceder a la aplicación web y las APIs para ingresar datos de inventario.

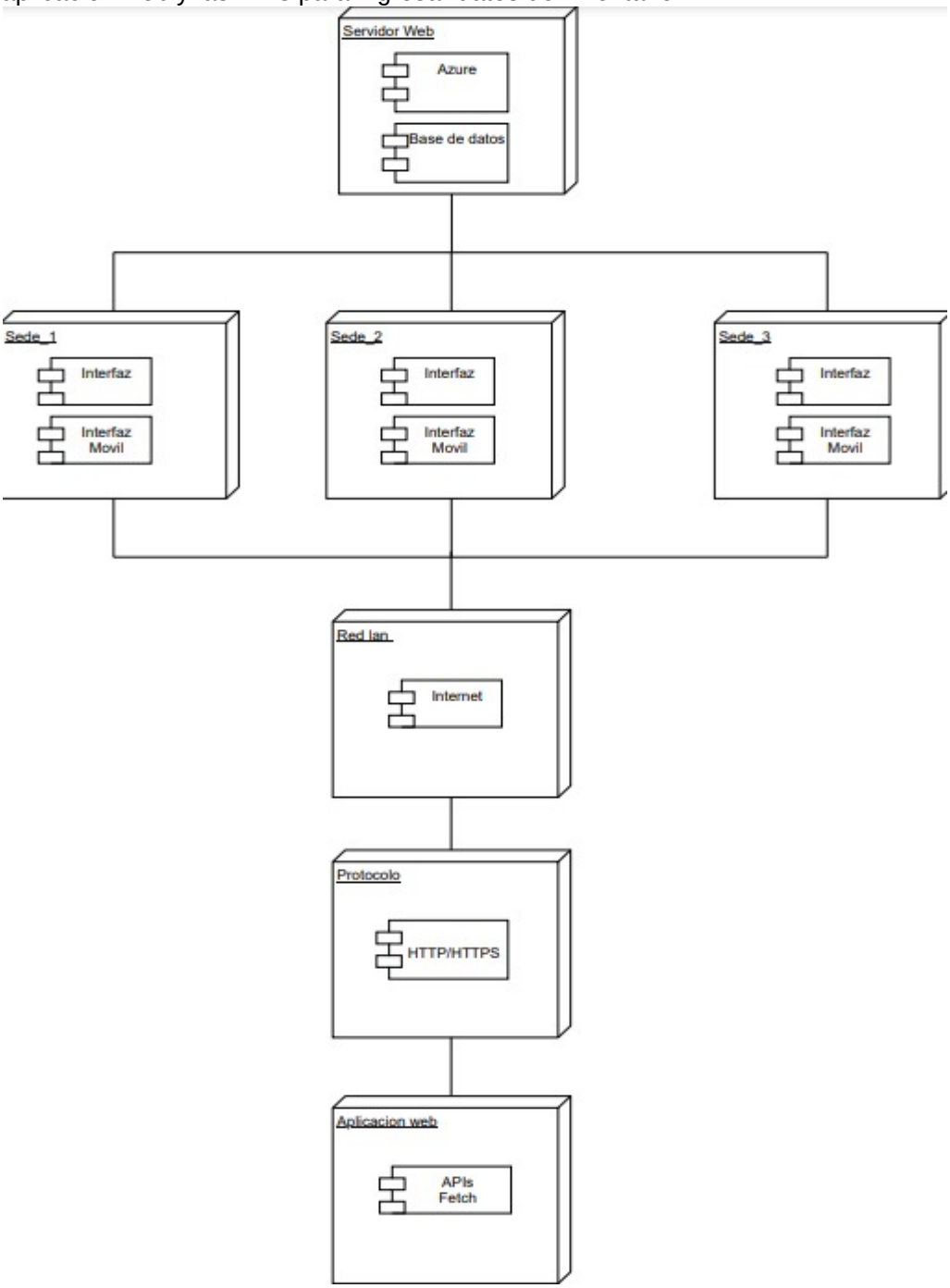
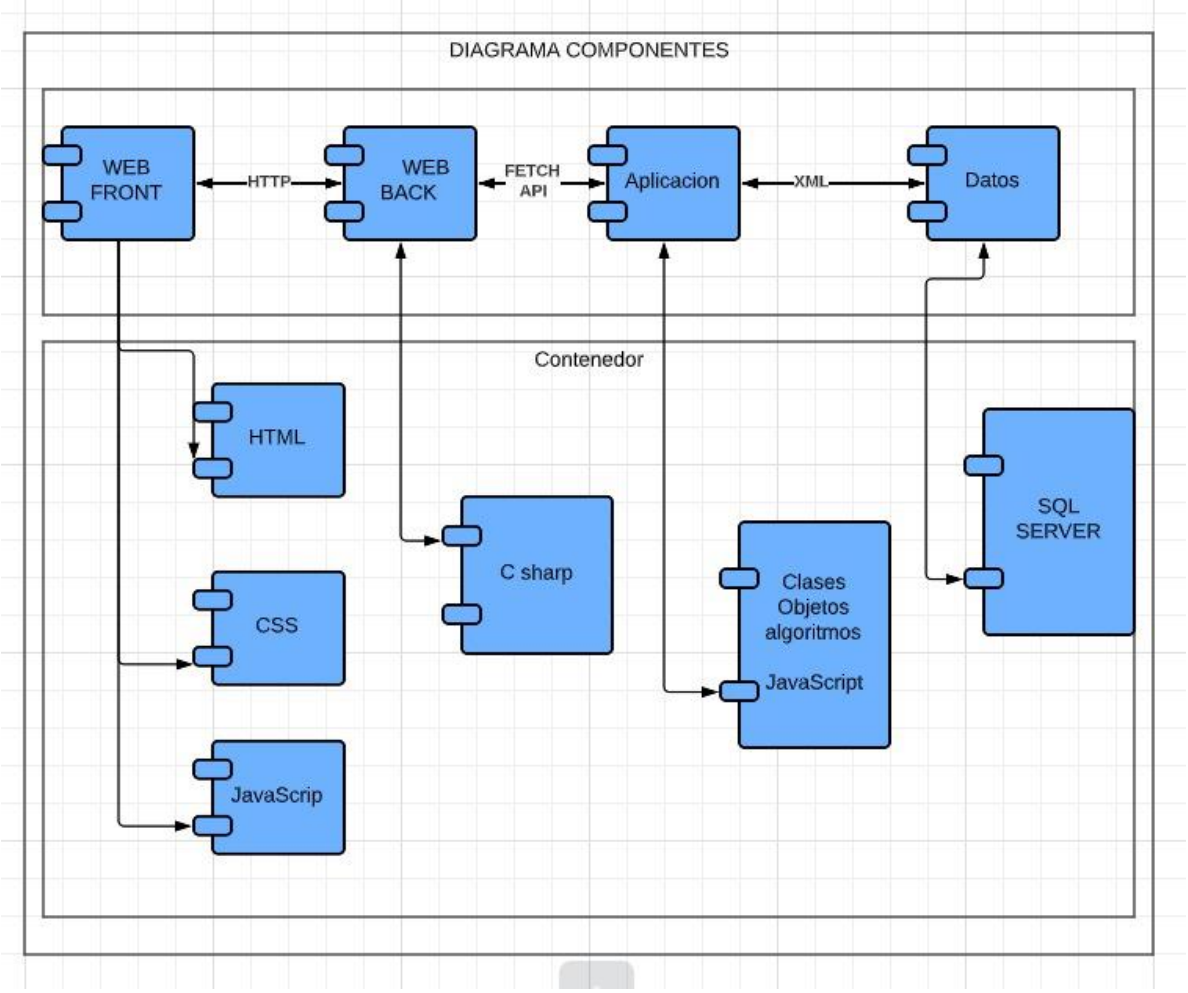


Diagrama de componentes

Diagrama de componente formado por cuatro módulos Frontend, Backend, lógico y datos. Cada uno contiene las aplicaciones que lo conforman y como se conectan cada módulo. se observa la composición del frontend por aplicaciones HTML, CSS, JavaScript. el siguiente modulo del backend comprendido por lenguaje C Sharp que se conecta al frontend por medio de conexión http. seguido va el módulo de aplicación el cual está formado por los diferentes algoritmos, clases y objetos todos desarrollados en javascript y se conectan con el módulo de backend por medio de api fetch, por último, se encuentra el módulo de datos en el cual se almacenara toda la información esto con la herramienta SQL server y con conexión XML al módulo de aplicación.



Referencias

Barragan, L. A. A. (2016). Lenguaje de modelamiento unificado (UML) para modelamiento de embotelladora. *Scientia et technica*, *21*(1), 6.

Elmasri, R., Navathe, S. B., Castillo, V. C., Pérez, G. Z., & Espiga, B. G. (2007). *Fundamentos de sistemas de bases de datos* (No. QA76. 9D3 E553 2007.). Pearson educación.