Proyecto ASP.NET Angular

Tabla de contenidos

- Proyecto ASP.NET Angular
- Tabla de contenidos
  - Introducción
  - Instalación
    -
  - Decidir que Sistema Operativo usaremos
    - Instalar MySQL
    - Instalación .NET
  - Intstalar Angular
    - Instalar Visual Studio Code
  - MYSQL
  - Angular y TypeScript (Frontend)
  - .NET
    - Sección

Introducción

Tabla de contenidos

Este proyecto es una aplicación web desarrollada utilizando ASP.NET y Angular. Combina la potencia del backend de ASP.NET con la flexibilidad y la capacidad de respuesta del framework de frontend Angular.

1. El objetivo de este proyecto:

- Crear una aplicación web donde realizar un CRUD normal y general permitiendo leer, eliminar, añadir, y actualizar directamente en la base de datos
- Ha sido montado en SLQ SERVER.

2. Las caracteristicas de usar estas tecnologías son:

- Integración de ASP.NET y Angular para aprovechar las fortalezas de ambos frameworks.
- Diseño modular y componentizado para facilitar el desarrollo y el mantenimiento.
- Uso de servicios RESTful para la comunicación entre el frontend y el backend.
- Pruebas unitarias y de integración para asegurar la calidad del código.

3. Por ultimo, pero no menos importante, para describir estas tecnologías:

- ASP.NET: Framework de desarrollo web de Microsoft.
- Angular: Framework de desarrollo de aplicaciones web de Google.
- C#: Lenguaje de programación utilizado en el backend.
- TypeScript: Lenguaje de programación utilizado en el frontend.
- HTML/CSS: Lenguajes de marcado utilizados para la estructura y el estilo de la interfaz de usuario.

- # URL
    -

Instalación

Tabla de contenidos

```
# ...
```

Decidir que Sistema Operativo usaremos

Esto es una decisión muy importante porque como se comprobará en este MarkDown utilizar Linux facilita muchisimo las cosas.

- Utilizar la consola de comandos para todo es extremadamente sencillo.

Instalar MySQL

Tabla de contenidos

```
sudo apt update
sudo apt install mysql-server mysql-client
```

- Ahora toca acceder, pero en primer lugar debemos definir el acceso sin sudo:

```
sudo mysql -u root -p
```

```
use mysql;
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root';
FLUSH PRIVILEGES;
exit

# NOTA: En caso de ser MariaDB usar:
# ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';
```

- Solo queda reinicirar MySQL y entrar con root/root

```
sudo service mysql restart
mysql -u root -p
# Y ahora ponemos la conraseña: root
```

Instalación .NET

Tabla de contenidos

- Recurso:
  - https://learn.microsoft.com/es-es/dotnet/core/install/linux-ubuntu-2204
- Componentes:
  - 1. SDK (Entorno Desarrollo + Ejecución)
  - 2. CLR (Entorno Ejecución) -> ASP.NET Core

1. Instalamos el SDK

```
sudo apt-get update
sudo apt-get install -y dotnet-sdk-7.0

# Visualizar las versiones instaladas
dotnet --list-sdks
dotnet --list-runtimes

# Para ver la ayuda del comando:
dotnet --help
```

2. Si luego queremos instalar paquetes adicionales de .NET, lo haremos por consola.

- Por ejemplo Newtonsoft.Json, que es una biblioteca muy popular para trabajar con JSON en .NET

```
dotnet add package Newtonsoft.Json
```

3. Instalamos Visual Studio Code como IDE

- Dentro, instalamos la extensión C# (el que viene con ese nombre de Microsoft, no el que pone C# Dev Kit)

Intstalar Angular

Tabla de contenidos

- Podemos instalar esta capa de Front-end tanto en Linux como en Windows.
- Me he tomado la libertad de tomar este magnifico apartado de nuestro querido profesor ya que la explicación en Linux son estos pasos tal cuales se explican a continuación.

> NOTA: para instalar Angular en Windows se hace a partir del instalador de NodeJS El repositorio para todo del 18 es este: https://nodejs.org/download/release/latest-v18.x/ Windows: https://nodejs.org/download/release/latest-v18.x/node-v18.18.2-x64.msi En windows NO hace falta yarn. Una vez instalado NODE, saltar a la sección 3. Instalar cliente Angular

1. Instalar Yarn

```
curl -sL https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list
sudo apt update && sudo apt install yarn
yarn --version
```

2. Instalar NodeJS y NPM

- Recursos:
  - https://github.com/nodesource/distributions#installation-instructions

```
sudo apt-get update
sudo apt-get install -y ca-certificates curl gnupg
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --
dearmor -o /etc/apt/keyrings/nodesource.gpg

NODE_MAJOR=18
echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg]
https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | sudo tee
/etc/apt/sources.list.d/nodesource.list

sudo apt-get update
sudo dpkg --remove --force-remove-reinstreq libnode72:amd64
sudo apt-get install nodejs -y
```

> IMPORTANTE: Si hemos instalado una versión incorrecta o queremos otra mas superior, procedemos a desinstalar con lo siguiente:

```
sudo apt-get purge nodejs
sudo rm -r /etc/apt/sources.list.d/nodesource.list
sudo rm -r /etc/apt/keyrings/nodesource.gpg
```

> IMPORTANTE: EN windows una vez acabada la instalación debemos hacer lo siguiente:
>
> 1. Reiniciar Powershell en modo administrador
> 2. Ejecutar este comando: Set-ExecutionPolicy Unrestricted
> 3. Decir [S] Si
> 4. Volver a reiniciar el powershell

3. Instalar el Cliente de Angular

```
sudo npm install -g @angular/cli
# Para ver la versión de NPM (la actual es la 7.5.4)
npm -v
# Instalamos el Cliente...
sudo yarn -npm install -g @angular/cli
# Vemos la versión actual
ng version

# Saldrá algo similar a esto...
Global setting: enabled
Local setting: No local workspace configuration file.
Effective status: enabled


     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 16.2.0
Node: 18.17.1
Package Manager: npm 9.8.1
OS: linux x64
```

Instalar Visual Studio Code

Tabla de contenidos

Recursos:

- https://ubunlog.com/visual-studio-code-editor-codigo-abierto-ubuntu-20-04/
- Instalación por consola:

```
sudo apt update
sudo apt update
sudo apt install software-properties-common apt-transport-https wget
wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64]
https://packages.microsoft.com/repos/vscode stable main"
sudo apt install code
```

- Extensiones:
  - Sería bueno que tuviesemos la opción de Formatear al guardar para mejorar la visualización del código.

1. Le damos a CTRL + MAY + X (Extensiones)
2. Escribimos Spanish Lenguage Pack y le damos a [install]. Pulsamos abajo a la derecha en [Restart].
3. Escribimos PHP Intelephense y le damos a [Instalar]
4. Escribimos Markdown All in One y le damos a [Instalar]
5. Abrimos la configuración de Visual Studio Code: [CTRL + ","]
6. Buscamos: settings.json. Le damos a Editar en JSON. Y ponemos esto: (para los archivo php y json formateamos al guardar)

```
{
  "[php]": {
    "editor.formatOnSave": true
  },
  "[json]": {
    "editor.quickSuggestions": {
      "strings": true
    },
    "editor.suggest.insertMode": "replace",
    "editor.formatOnSave": true
  }
}
```

MYSQL

Tabla de contenidos

- Para este proyecto he montado la base de datos de Mascotas en MYSQL SERVER.
- Instalación:
    1. Primero tendremos que descargarnos el sistema de administración de datos donde los anidaremos:
    - https://www.microsoft.com/es-es/download/details.aspx?id=101064
    2. Segundo tendremos que instalar el SQL Management Studio donde gestionaremos nuestra base de datos: -https://learn.microsoft.com/es-es/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16
- He creado una base de datos muy sencilla para hacer nuestro CRUD sobre Mascotas:

```
# SELECT TOP (1000) [id_mascota]
      ,[nombre]
      ,[edad]
      ,[descripcion]
  FROM [DB_MASCOTAS].[dbo].[Mascota]
```

En esta parte del codigo podemos traernos las 1000 primeras filas y observar los datos de prueba que hemos creado. Podemos observar las columnas que hemos creado.

- También para facilitar el manejo de datos en nuestro CRUD, MYSQL nos facilita una herramienta de crear procedimientos almacenados. Hemos aprovechado esta función:
    1. Añadir:

```
# USE [DB_MASCOTAS]
GO
/****** Object:  StoredProcedure [dbo].[Mascota_Add]    Script Date:
29/01/2024 19:00:48 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[Mascota_Add]
@nombre varchar(50),
@edad int,
@desc varchar(MAX)
AS
INSERT INTO Mascota (nombre, edad, descripcion)
VALUES (@nombre, @edad, @desc)
```

2. Eliminar:

```
USE [DB_MASCOTAS]
GO
/****** Object:  StoredProcedure [dbo].[Mascota_All]    Script Date:
29/01/2024 19:05:52 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[Mascota_All]
AS
SELECT id_mascota, nombre, edad, descripcion FROM Mascota
ORDER BY id_mascota ASC
```

3. Todos:

```
USE [DB_MASCOTAS]
GO
/****** Object:  StoredProcedure [dbo].[Mascota_All]    Script Date:
29/01/2024 19:06:27 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[Mascota_All]
AS
SELECT id_mascota, nombre, edad, descripcion FROM Mascota
ORDER BY id_mascota ASC
```

4. Actualizar:

```
# USE [DB_MASCOTAS]
GO
/****** Object:  StoredProcedure [dbo].[Mascota_All]    Script Date:
29/01/2024 19:06:27 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[Mascota_All]
AS
SELECT id_mascota, nombre, edad, descripcion FROM Mascota
ORDER BY id_mascota ASC
```

- Con esto terminamos con la parte de MySQL

Angular y TypeScript (Frontend)

Tabla de contenidos

- Pasamos a la parte del Frontend de nuestra WEB API. En esta parte hemos utilizado Angular, un fantastico framework que junto con Material UI tiene una gran cantidad de componentes esteticos prediseñados que nos facilitaran un mpontón las cosas.
- Hemos hecho 4 cosas principalmente:
    1. Crear el modelo que se recivirá desde el back.
    2. Crear la carpeta de servicios donde añadiremos las funciones de nuestro CRUD.
    3. Crear la vista de nuestra app web que iría en app.component.html
    4. Añadir a app.component.ts el código donde iría la lógica de nuestra app.
- El diseño de nuestra web se encuentra en el componente html de Angular.
- Este sería: app.component.html
- Hemos hecho una interfaz muy sencilla propia de un CRUD:

```
<div class="container">
  <div class="row text-center mt-4">
    <div class="col">
      <img
        src="https://p4.wallpaperbetter.com/wallpaper/901/505/133/cats-dogs-pets-
wallpaper-preview.jpg"
        alt=""
        class="img-fluid"
        width="200px"
      />
    </div>
  </div>
  <div class="row text-center mt-5">
    <div class="col">
      <h2>CRUD Mascotas</h2>
    </div>
  </div>
  <div class="row mt-5">
    <div class="col-md-2">
      <label for="txtId" class="from-label">ID</label>
      <input
        type="text"
        name="id"
        id="txtId"
        class="form-control"
        [(ngModel)]="mascota.id"
        disabled="true"
      />
    </div>
    <div class="col-md-4">
```

```
        <label for="txtNombre" class="from-label">Nombre</label>
        <input
          type="text"
          name="id"
          id="txtNombre"
          class="form-control"
          [(ngModel)]="mascota.nombre"
        />
      </div>
      <div class="col-md-2">
        <label for="txtEdad" class="from-label">Edad</label>
        <input
          type="text"
          name="id"
          id="txtEdad"
          class="form-control"
          [(ngModel)]="mascota.edad"
        />
      </div>
      <div class="col-md-4">
        <label for="txtDesc" class="from-label">Descripcion</label>
        <input
          type="text"
          name="id"
          id="txtDesc"
          class="form-control"
          [(ngModel)]="mascota.descripcion"
        />
      </div>
    </div>
    <div class="row mt-5">
      <div class="col">
        <button
          class="btn btn-success"
          type="button"
          (click)="onAddMascota(mascota)"
        >
          Agregar
        </button>
        <button
          class="btn btn-warning m-2"
          type="button"
          (click)="onUpdateMascota(mascota)"
        >
          Modificar
        </button>
        <button
          class="btn btn-danger"
          type="button"
          (click)="onDeleteMascota(mascota.id)"
        >
          Eliminar
        </button>
        <button class="btn btn-primary m-2" type="button" (click)="clear()">
```

```
          Limpiar
        </button>
      </div>
    </div>
    <div class="row mt-5">
      <div class="col">
        <table id="dtMascota" class="table table-striped" style="width: 100%;">
          <thead>
            <tr>
              <th>Seleccionar</th>
              <th>Id</th>
              <th>Nombre</th>
              <th>Edad</th>
              <th>Descripcion</th>
            </tr>
          </thead>
          <tbody>
            <tr *ngFor="let item of datatable">
              <td>
                <input
                  type="button"
                  value="Seleccionar"
                  (click)="onSetData(item)"
                />
              </td>
              <td>{{ item.idMascota }}</td>
              <td>{{ item.nombre }}</td>
              <td>{{ item.edad }}</td>
              <td>{{ item.descripcion }}</td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </div>
```

- El modelo muy sencillo, la estructura de datos es la siguiente:

```
export class Mascota {
  id:number = 0;
  nombre:string = "";
  edad:number = 0;
  descripcion:string = "";
}
```

- La carpeta de servicios:
    - Esta es una parte muy importante ya que es la parte que se conectaría a la API. Una vez más añadimos las funciones especificas de nuestro CRUD.

- Por ultimo tendriamos app.component.ts donde iría la parte lógica de nuestra app. (Añadiré un ejemplo para no poner todo el codigo). Esta parte del codigo controlará la lógica tal como lo podría hacer PHP.

```
onAddMascota(mascota:Mascota):void{
  this.mascotaService.addMascota(mascota).subscribe(res => {
    if(res){
      alert(`La mascota ${mascota.nombre} se ha registrado con exito!`);
      this.clear();
      this.onDataTable();
    } else {
      alert('Error! :(')
    }
  });
}
```

.NET

Tabla de contenidos

- Para la parte del back hemos utilizado ASP.NET. Para facilitarnos mucho las cosas la hemos implementado en Visual Studio Community.
- Esto es debido a que trae numerosas plantilla que nos facilitan y agilizan mucho el trabajo.
- Es muy sencillo, simplemente hemos "Creado un proyecto" -> "Utilizado una plantilla" -> "Aplicación WEB ASP.NET(.NET Framework)" con .NET Framework 4.7.2
- Muy muy sencillo, nos crea el proyecto completo con todas sus carpetas solo tocaremos dos carpetas y una clase.

MUY IMPORTANTE! - - Cabe destacar que nos basamos en el MVC para realizar este proyecto.

1. Primero deberemos configurar la coadena de conexion para que apunte a nuestra BD aparte de a nuestro servidor donde hemos dicho anteriormente que lo hemos hecho con SQLEXPRESS.

> NOTA: SQL Studio no funcionará sin SQLEXPRESS.

- Nos vamos a Web.config

```
<connectionStrings>
  <add name="BDLocal" connectionString="Database=DB_MASCOTAS;Server=JS-
PORTATIL\SQLEXPRESS;User=sa;Password=123654on;Integrated Security=;"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

2. Posteriormente procedemos al paso de creación del modelo.
    - Comenzamos con las propiedades en Mascota.cs además de sus constructores.
    - > NOTA: ASP.NET utilza C#.

```
  using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace back_mascota.Models
{
    public class Mascota
    {
        public int idMascota { get; set; }
        public string nombre { get; set; }
        public int edad { get; set; }
        public string descripcion { get; set; }
        public int idTipo { get; set; }


        public Mascota() { }
```

```
        public Mascota(int id, string Nombre, int Edad, string desc, int tipo)
        {
            idMascota = id;
            nombre = Nombre;
            edad = Edad;
            descripcion = desc;
            idTipo = tipo;
        }

        public Mascota(string Nombre, int Edad, string desc, int tipo)
        {
            nombre = Nombre;
            edad = Edad;
            descripcion = desc;
            idTipo = tipo;
        }
    }
}
```

- - IMPORTANTE! - En la carpeta models encontramos tambien las clases modelo para Raza.cs y TipoMascota.cs.

1. Lo siguiente es crear la clase que gestionará nuestra BD en este caso he creado GestorMascota.cs. Se encarga de la logica que manejará los datos y creará la conexión con la misma.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Web;

namespace back_mascota.Models
{
    public class GestorMascota
    {

        public List<Mascota> getMascota()
        {
            List<Mascota> lista = new List<Mascota>();
            string strConn =
ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();

                SqlCommand cmd = conn.CreateCommand();
                cmd.CommandText = "Mascota_All";
```

```
                cmd.CommandType = CommandType.StoredProcedure;

                SqlDataReader dr = cmd.ExecuteReader();

                while(dr.Read())
                {
                    int id = dr.GetInt32(0);
                    string nombre = dr.GetString(1).Trim();
                    int edad = dr.GetInt32(2);
                    string desc = dr.GetString(3).Trim();

                    Mascota mascota = new Mascota(id, nombre, edad, desc);

                    lista.Add(mascota);
                }

                dr.Close();
                conn.Close();
            }

            return lista;
        }

        public bool addMascota(Mascota mascota)
        {
            bool res = false;

            string strConn =
ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

            using (SqlConnection conn = new SqlConnection(strConn))
            {
                SqlCommand cmd = conn.CreateCommand();
                SqlDataAdapter adapter = new SqlDataAdapter(cmd);
                cmd.CommandText = "Mascota_Add";
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@nombre", mascota.nombre);
                cmd.Parameters.AddWithValue("@edad", mascota.edad);
                cmd.Parameters.AddWithValue("@desc", mascota.descripcion);

                try
                {
                    conn.Open();
                    cmd.ExecuteNonQuery();
                    res = true;
                }
                catch(Exception ex)
                {
                    Console.WriteLine(ex.Message);
                    res = false;
                    throw;
                }
                finally
```

```
                {
                    cmd.Parameters.Clear();
                    conn.Close();
                }

                return res;
            }
        }

        public bool updateMascota(int id, Mascota mascota)
        {
            bool res = false;

            string strConn =
ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

            using (SqlConnection conn = new SqlConnection(strConn))
            {
                SqlCommand cmd = conn.CreateCommand();
                SqlDataAdapter adapter = new SqlDataAdapter(cmd);
                cmd.CommandText = "Mascota_Update";
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@id", id);
                cmd.Parameters.AddWithValue("@nombre", mascota.nombre);
                cmd.Parameters.AddWithValue("@edad", mascota.edad);
                cmd.Parameters.AddWithValue("@desc", mascota.descripcion);

                try
                {
                    conn.Open();
                    cmd.ExecuteNonQuery();
                    res = true;
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                    res = false;
                    throw;
                }
                finally
                {
                    cmd.Parameters.Clear();
                    conn.Close();
                }

                return res;
            }
        }

        public bool deleteMascota(int id)
        {
            bool res = false;
```

```
            string strConn =
ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

            using (SqlConnection conn = new SqlConnection(strConn))
            {
                SqlCommand cmd = conn.CreateCommand();
                SqlDataAdapter adapter = new SqlDataAdapter(cmd);
                cmd.CommandText = "Mascota_Delete";
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@id", id);

                try
                {
                    conn.Open();
                    cmd.ExecuteNonQuery();
                    res = true;
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                    res = false;
                    throw;
                }
                finally
                {
                    cmd.Parameters.Clear();
                    conn.Close();
                }

                return res;
            }
        }
    }
}
```

4. Posteriormente tenemos que añadir nuestro GestorMascota.cs al controlador para ello hemos creado MascotaController.cs
    - Gracias a que VS entiende cuando nombramos una clase como NombreDeNuestroControladorController.cs. Nos crea practicamente al clase controlador, solo tendriamos que rellenar las funciones GET, POST, PUT, DELETE.
5. Para finalizar en esta parte de ASP.NET deberemos configurar App_Start, concretamente la clase WebApiConfig.cs para que en vez de trabajar con XML trabajaremos con JSON ya que es lo mas común. Lo hariamos a traves de estas líneas de codigo:

```
GlobalConfiguration.Configuration.Formatters.JsonFormatter.MediaTypeMappings
                .Add(new System.Net.Http.Formatting.RequestHeaderMapping("Accept",
                            "text/html",
                            StringComparison.InvariantCultureIgnoreCase,
                            true,
```

```
                        "application/json")
                );
```

- Con esto acabariamos la parte de ASP.NET

Sección

Tabla de contenidos

```
# ...
```