Contenido

[Modelo de BD 2](#_Toc179966523)

[Construcción paso a paso de la aplicación 6](#_Toc179966524)

## Modelo de BD

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

DDD  
Diccionario de Datos

|  |
| --- |
|  |
| | **Tabla** | **Campo** | **Tipo de Dato** | **Descripción** | | --- | --- | --- | --- | | Nivel | id\_nivel | INT | Identificador único del nivel. | |  | nivel\_nombre | VARCHAR(50) | Nombre del nivel (ej. "Primero Medio"). | | Asignatura | id\_asignatura | INT | Identificador único de la asignatura. | |  | asignatura\_codigo | VARCHAR(20) | Código de la asignatura. | |  | nivel\_id | INT | Identificador del nivel asociado a la asignatura. | | Bibliografia | id\_bibliografia | INT | Identificador único de la bibliografía. | |  | isbn | VARCHAR(13) | Número ISBN del libro. | |  | titulo | VARCHAR(100) | Título del libro. | |  | editorial | VARCHAR(50) | Editorial del libro. | |  | anio | INT | Año de publicación. | |  | autor | VARCHAR(50) | Autor del libro. | | Bibliografia\_asignatura | id\_biblioasignatura | INT | Identificador único de la relación bibliografía-asignatura. | |  | bibliografia\_id | INT | Identificador de la bibliografía asociada. | |  | asignatura\_id | INT | Identificador de la asignatura asociada. | | Unidad | id\_unidad | INT | Identificador único de la unidad. | |  | nombre\_unidad | VARCHAR(50) | Nombre de la unidad. | |  | horas\_pedagogicas | INT | Horas pedagógicas asignadas a la unidad. | |  | asignatura\_id | INT | Identificador de la asignatura asociada. | | Conocimiento | id\_conocimiento | INT | Identificador único del conocimiento. | |  | nombre\_conocimiento | VARCHAR(50) | Nombre del conocimiento. | |  | unidad\_id | INT | Identificador de la unidad asociada. | | Objetivos\_de\_aprendizaje | id\_objetivos\_aprendizaje | INT | Identificador único del objetivo de aprendizaje. | |  | codigo | VARCHAR(20) | Código del objetivo de aprendizaje. | |  | descripcion | VARCHAR(200) | Descripción del objetivo de aprendizaje. | |  | conocimiento\_id | INT | Identificador del conocimiento asociado. | | Indicadores\_de\_evaluacion | id\_indicadores\_evaluacion | INT | Identificador único del indicador de evaluación. | |  | indicador\_nombre | TEXT | Nombre del indicador de evaluación. | |  | objetivos\_aprendizaje\_id | INT | Identificador del objetivo de aprendizaje asociado. | |

## Construcción paso a paso de la aplicación

Para crear una aplicación web ASP.NET Core que se conecte a una base de datos SQL usando la cadena de conexión que proporcionaste y que integre la API de ChatGPT para mejorar las respuestas a las consultas, sigue estos pasos detallados:

1. Crear un Nuevo Proyecto

1. Abre Visual Studio.

2. Selecciona Crear un nuevo proyecto.

3. Busca y selecciona **Aplicación web ASP.NET Core** y haz clic en Siguiente.

4. Asigna un nombre a tu proyecto y elige una ubicación. Haz clic en Crear.

5. En la siguiente ventana, selecciona **Aplicación web (Modelo-Vista-Controlador) (MVC**) y asegúrate de que el marco de trabajo sea **.NET 6.0 o superior**. Haz clic en Crear.

Texto

Descripción generada automáticamente

2. Configurar la Conexión a la Base de Datos

Paso 1: Agregar Entity Framework Core

1. Haz clic derecho en tu proyecto en el Explorador de soluciones y selecciona Administrar paquetes NuGet. Captura de pantalla de computadora

   Descripción generada automáticamente

2. Busca e instala los siguientes paquetes:

- `Microsoft.EntityFrameworkCore.SqlServer`

Captura de pantalla de computadora

Descripción generada automáticamente

- `Microsoft.EntityFrameworkCore.Tools`

Captura de pantalla de computadora

Descripción generada automáticamente

Comprobación

Texto

Descripción generada automáticamente

Paso 2: Configurar la Cadena de Conexión

1. Abre el archivo `appsettings.json`.

2. Agrega la cadena de conexión que proporcionaste:

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

```json

{

"ConnectionStrings": {

"MiConexion": "Server=localhost;Database=**AsistenteInstruccional**;Integrated Security=True;Encrypt=True;"

}

}

```

Comprobación

Texto

Descripción generada automáticamente

3. Crear el Modelo y el Contexto

Paso 1: Crear una Entidad

1. Crea una nueva carpeta llamada `Models` en el proyecto.

2. Agrega una clase llamada `Consulta.cs` en la carpeta `Models`:

Captura de pantalla de un celular

Descripción generada automáticamente

comprobación

Captura de pantalla de un celular

Descripción generada automáticamente

```csharp

public class Consulta

{

public int Id { get; set; }

public string Pregunta { get; set; }

public string Respuesta { get; set; }

}

```

Texto

Descripción generada automáticamente con confianza media

Paso 2: Crear el Contexto de la Base de Datos

1. Agrega una clase llamada `MiDbContext.cs` en la carpeta `Models`: (**Como en paso anterior)**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

```csharp

using Microsoft.EntityFrameworkCore;

public class MiDbContext : DbContext

{

public MiDbContext(DbContextOptions<MiDbContext> options) : base(options) { }

public DbSet<Consulta> Consultas { get; set; }

}

Una captura de pantalla de una computadora

Descripción generada automáticamente

4. Configurar el Contexto en `Startup.cs`

1. Abre el archivo `Programs.cs`.

2. En el método `ConfigureServices`, agrega el contexto de la base de datos:

Captura de pantalla de computadora

Descripción generada automáticamente

```csharp

using Microsoft.EntityFrameworkCore;

using ProyAI\_MVC.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<MiDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("MiConexion")));

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

5. Crear un Controlador

1. Crea una nueva carpeta llamada `Controllers`.

2. Agrega una clase llamada `ConsultaController.cs` en la carpeta `Controllers`:

```csharp

using Microsoft.AspNetCore.Mvc;

using System.Linq;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

public class ConsultaController : Controller

{

private readonly MiDbContext \_context;

private readonly HttpClient \_httpClient;

public ConsultaController(MiDbContext context)

{

\_context = context;

\_httpClient = new HttpClient();

}

public IActionResult Index()

{

var consultas = \_context.Consultas.ToList();

return View(consultas);

}

[HttpPost]

public async Task<IActionResult> Crear(string pregunta)

{

if (!string.IsNullOrEmpty(pregunta))

{

// Llamar a la API de ChatGPT

var respuesta = await ObtenerRespuestaChatGPT(pregunta);

// Guardar en la base de datos

var nuevaConsulta = new Consulta { Pregunta = pregunta, Respuesta = respuesta };

\_context.Consultas.Add(nuevaConsulta);

\_context.SaveChanges();

}

return RedirectToAction(nameof(Index));

}

private async Task<string> ObtenerRespuestaChatGPT(string pregunta)

{

var apiKey = "TU\_API\_KEY"; // Reemplaza con tu clave de API de OpenAI

var requestBody = new

{

model = "gpt-3.5-turbo",

messages = new[]

{

new { role = "user", content = pregunta }

}

};

\_httpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", apiKey);

var json = JsonConvert.SerializeObject(requestBody);

var content = new StringContent(json, Encoding.UTF8, "application/json");

var response = await **\_httpClient.PostAsync("https://api.openai.com/v1/chat/completions", content);**

**response.EnsureSuccessStatusCode();**

var responseBody = await response.Content.ReadAsStringAsync();

dynamic jsonResponse = JsonConvert.DeserializeObject(responseBody);

return jsonResponse.choices[0].message.content.ToString();

}

}

```

6. Crear Vistas

1. Crea una nueva carpeta llamada `Views/Consulta`.

2. Agrega una vista llamada `Index.cshtml`:

```html

@model IEnumerable<Consulta>

<h1>Consultas</h1>

<table>

<thead>

<tr>

<th>Pregunta</th>

<th>Respuesta</th>

</tr>

</thead>

<tbody>

@foreach (var consulta in Model)

{

<tr>

<td>@consulta.Pregunta</td>

<td>@consulta.Respuesta</td>

</tr>

}

</tbody>

</table>

<h2>Realizar Nueva Consulta</h2>

<form asp-action="Crear" method="post">

<input type="text" name="pregunta" required />

<button type="submit">Enviar</button>

</form>

```

7. Ejecutar Migraciones

1. Abre la Consola del Administrador de Paquetes en Visual Studio.

2. Ejecuta los siguientes comandos para crear y aplicar migraciones:

```bash

Add-Migration InitialCreate

Update-Database

```

8. Ejecutar la Aplicación

1. Presiona `F5` para ejecutar la aplicación.

2. Navega a `/Consulta` para ver la lista de consultas y realizar nuevas.

Resumen de Integración de ChatGPT

- Integración de API: En el controlador `ConsultaController`, se utiliza `HttpClient` para realizar una llamada a la API de ChatGPT. La función `ObtenerRespuestaChatGPT` envía la pregunta a la API y recibe la respuesta.

- Almacenamiento: Cada pregunta y respuesta se almacenan en la base de datos a través del contexto `MiDbContext`.

Notas Adicionales

- API Key: Asegúrate de reemplazar `TU\_API\_KEY` con tu clave de API de OpenAI.

- Manejo de Errores: Considera agregar manejo de errores para la llamada a la API y para la interacción con la base de datos.