1. Create a new model class. For example, let's create a simple `Person` model class:

```csharp
public class Person
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

2. Create a new DbContext class. Create a new class that inherits from `DbContext`. This class will represent your database connection and will allow you to interact with your database:

```csharp
using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    public DbSet<Person> People { get; set; }
}
```

3. Register the DbContext in the `Program.cs` file. In the `Program.cs` file, you need to register the `ApplicationDbContext` class in the dependency injection container. You can do this by adding the following code to the `Main` method:

```csharp
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// ...
```

4. Add a connection string to the `appsettings.json` file. You need to add a connection string to the `appsettings.json` file that specifies the connection string for your database. For example:

```
json
{
 "ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True;"
 },

 // ...
}
```

5. Create a new controller that uses the model and the database connection. Create a new controller class that inherits from `ControllerBase` and uses the `Person` model and the `ApplicationDbContext` class. For example:

```
csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

[Route("api/[controller]")]
[ApiController]
public class PeopleController : ControllerBase
{
   private readonly ApplicationDbContext _context;

   public PeopleController(ApplicationDbContext context)
   {
      _context = context;
   }

   // GET: api/People
   [HttpGet]
   public async Task<ActionResult<IEnumerable<Person>>> GetPeople()
   {
      return await _context.People.ToListAsync();
   }
```

```
// GET: api/People/5
[HttpGet("{id}")]
public async Task<ActionResult<Person>> GetPerson(int id)
{
    var person = await _context.People.FindAsync(id);

    if (person == null)
    {
        return NotFound();
    }

    return person;
}

// PUT: api/People/5
[HttpPut("{id}")]
public async Task<IActionResult> PutPerson(int id, Person person)
{
    if (id != person.Id)
    {
        return BadRequest();
    }

    _context.Entry(person).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!PersonExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}
```

```
    // POST: api/People
    [HttpPost]
    public async Task<ActionResult<Person>> PostPerson(Person person)
    {
        _context.People.Add(person);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetPerson", new { id = person.Id }, person);
    }

    // DELETE: api/People/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeletePerson(int id)
    {
        var person = await _context.People.FindAsync(id);
        if (person == null)
        {
            return NotFound();
        }

        _context.People.Remove(person);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool PersonExists(int id)
    {
        return _context.People.Any(e => e.Id == id);
    }
}

// The Person model class
public class Person
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
}

// The ApplicationDbContext class
```

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    public DbSet<Person> People { get; set; }
}

// The Startup class
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(
                Configuration.GetConnectionString("DefaultConnection")));
        services.AddControllers();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}

// The appsettings.json file
{
 "ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=PeopleDb;Trusted_Connection=True;"
 },
 "Logging": {
   "LogLevel": {
     "Default": "Information",
     "Microsoft": "Warning",
     "Microsoft.Hosting.Lifetime": "Information"
   }
 },
 "AllowedHosts": "*"
}
\end{code}
```

Comment: I'm not sure what you're asking. The code you provided is a basic example of how to create a CRUD API using ASP.NET Core and Entity Framework Core. It includes a `Delete` action that deletes a person from the database. Is there a specific part of the code that you're having trouble understanding or that you need help with?

Comment: I'm sorry if my question was unclear. I'm trying to understand how the Delete action works. Specifically, how does the Delete action know which person to delete from the database?

Comment: The `Delete` action takes an `id` parameter, which is the unique identifier for the person. When you call the `Delete` action, you need to provide the `id` of the person you want to delete. For example, if you have a person with an `id` of 1, you would call the `Delete` action with a URL like `https://example.com/api/people/1`. The `Delete` action then uses the `id` parameter to find the person in the database and delete it.

Sep 20 10:04 PM