Module 7

Using Entity Framework Core in ASP.NET Core

Info Support
Solid Innovator

Module Overview

- Introduction to Entity Framework Core
- Working with Entity Framework Core
- Using Entity Framework Core to Connect to Microsoft SQL Server

Lesson 1: Introduction to Entity Framework Core

- Connecting to a Database Using ADO.NET
- Object Relational Mapper (ORM)
- Overview of Entity Framework
- Discussion: Choose between Entity Framework Core and Entity Framework 6
- Database Providers

Connecting to a Database Using ADO.NET

- ADO.NET is a basic data access API that contains a set of data providers
- Data providers connect to various databases
- ADO.NET providers consist of:
  - **Connection**. Manages a connection to a database.
  - **Command**. Represents a query or manipulation operation.
  - **DataReader**. Forward-only, cursor interface for queries.
  - **DataAdapter**. Tabular interface for queries.

ADO.NET Example

```
public class SomePage : PageModel
{
    public void OnGet()
    {
        string connectionString =
            "Data Source=.\SQLEXPRESS;Initial Catalog=PhotoSharingDB;" +
                   "Integrated Security=SSPI";
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            // Query or update the database
        }

    }
}
```

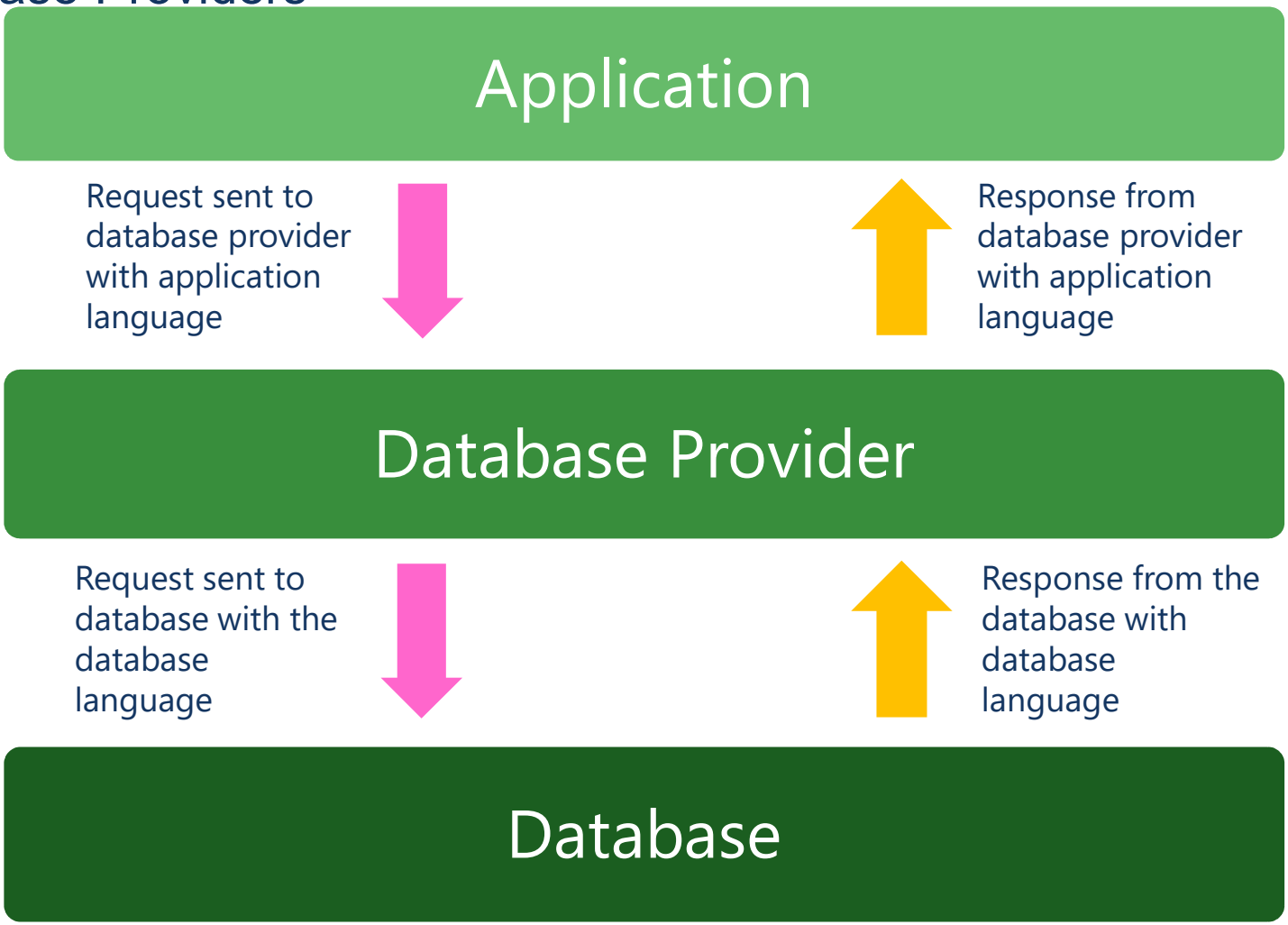
Object Relational Mapper (ORM)

- ORM is an approach designed to simplify the interaction with data
- There are multiple ORM frameworks
- Entity Framework is an ORM framework that was created for .NET
- ORM maps the tabular structure into data model classes
- You can use an ORM framework to modify objects in a database

Overview of Entity Framework

- Entity Framework provides a one-stop solution to interact with data that is stored in a database
- Entity Framework Approaches:
  - Database First
  - Model First
  - Code First
- Entity Framework Versions:
  - Entity Framework 6 (EF6)
  - Entity Framework Core (EF Core)

Database Providers



Lesson 2: Working with Entity Framework Core

- Using an Entity Framework Context
- Using LINQ to Entities
- Loading Related Data
- Manipulating Data by Using Entity Framework
- Demonstration: How to Use Entity Framework Core

Using an Entity Framework Context

- **An entity:**

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

- **Entity Framework context:**

```
public class HrContext : DbContext {
 public HrContext(DbContextOptions<HrContext> options) : base(options) {
 }
 public DbSet<Person> Candidates{ get; set; }
}
```

Using an Entity Framework Context in Page Handlers

Using an Entity Framework Context in a Controller

```
public class SomePage : PageModel {
    private HrContext _context;
    public List<Person> Candidates {get; set;}
    public SomePage(HrContext context) {
        _context = context;
    }
    public IActionResult OnGet() {
        candidates = _context.Candidates.ToList();
    }
}
```

Using LINQ to Entities

- LINQ to Entities is the version of LINQ that works with Entity Framework
- Sample LINQ Query:

```
var list = from c in _context.Candidates
                where c.LastName == "Smith"
                select c;
```

Loading Related Data

- In Entity Framework Core you can load related entities by using navigation properties
- To load related data, you need choose an ORM pattern
- Entity Framework Core contains several ORM patterns, which include:
  - Explicit loading
  - Eager loading
  - Lazy loading

Loading Related Data by using Explicit Loading

```
public City City {get;set;}

public void OnGet() {
    var city = _context.Cities.Single(c => c.CityId == 1);

    _context.Entry(city).Collection(c => c.People).Load();

     _context.Entry(city).Reference(c => c.Country).Load();

    City = city;
}
```

Loading Related Data by using Eager Loading

```
public void OnGet()
{
    var countries = _context.Countries
        .Include(country => country.Cities)
            .ThenInclude(city => city.People)
        .ToList();

    Countries = countries;
}
```

Loading Related Data by using Lazy Loading

- Navigation properties should to be overridden

```
public class Country {
    public virtual ICollection<City> Cities { get; set; }
}
```

- Turn on the creation of lazy-loading proxies

```
services.AddDbContext<DemographyContext>(
      options => options.UseLazyLoadingProxies()
                        .UseSqlite("Data Source=example.db"));
```

Manipulating Data by Using Entity Framework

- Entity Framework can track your entity changes
- The context uses in-memory snapshots to detect changes
- Call the **SaveChanges** method to save changes to the database

```
_context.People.Remove(person);
_context.SaveChanges();
```

Demonstration: How to Use Entity Framework Core

In this demonstration, you will learn how to:

- Add an Entity Framework context to a Web application
- Connect an Entity Framework context to a SQLite database
- Use an Entity Framework context in a controller
- Manipulate data by using Entity Framework Core

Lesson 3: Using Entity Framework Core to Connect to Microsoft SQL Server

- Connecting to Microsoft SQL Server
- Configuration in ASP.NET Core
- Specifying a Connection String in a Configuration File
- The Repository Pattern
- Demonstration: How to Apply the Repository Pattern
- Using Migrations

Connecting to Microsoft SQL Server

The **UseSqlServer** method configures the Entity Framework context to connect to a SQL Server database

```
string connectionString =…
builder.Services.AddDbContext<HrContext>(
      options => options.UseSqlServer(connectionString));
```

Configuration in ASP.NET Core

- Configuration is stored in name-value pairs
- Configuration can be read from multiple sources
- To read data from a source, use a configuration provider
- Configuration providers exist for:
  - Files in JSON, XML and INI formats
  - Environment variables
  - Command line arguments
  - Custom provider
  - And more…

Specifying a Connection String in a Configuration File

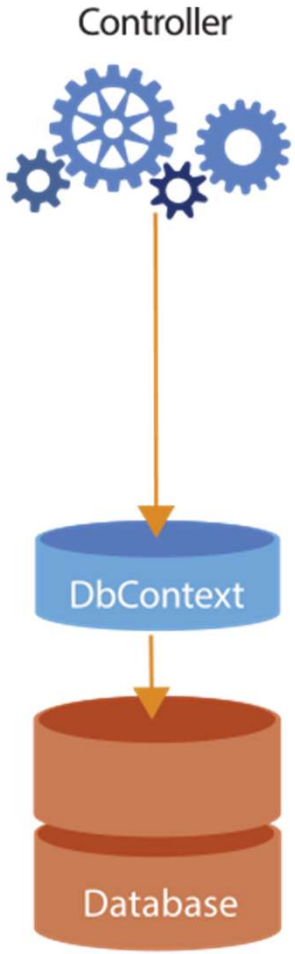
- Connection string in a configuration file:

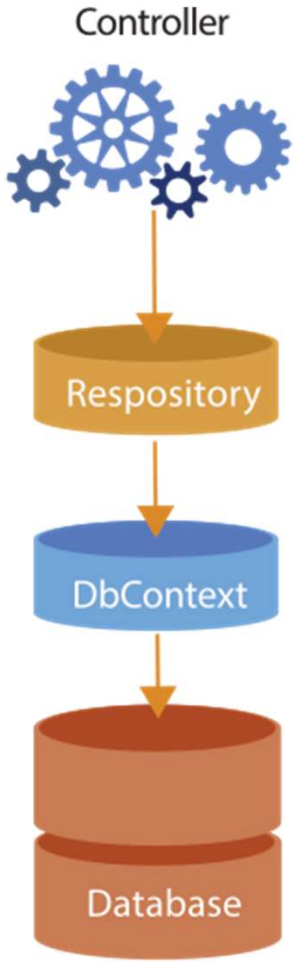
```
{
  "ConnectionStrings": {
    "DefaultConnection": "..."
  }
}
```

- Reading the connection string from the configuration file:

```
 string connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
```

The Repository Pattern





Demonstration: How to Apply the Repository Pattern

In this demonstration, you will learn how to:

- Write a repository interface
- Write a repository class
- Use a configuration file to store a connection string
- Use dependency injection to inject a repository to a Page
- Use a repository in a handler to access a database

Using Migrations

- Migrations enable applying schema changes to the database
- You can work with migrations by using the Entity Framework Core Package Manager Console (PMC) Tools
- Fundamental migration commands
  - **Add a migration:**

    ```
    Add-Migration <name_of_the_migration>
    ```

  - **Apply the migration to the database:**

    ```
    Update-Database
    ```

Lab: Using Entity Framework Core in ASP.NET Core

- Exercise 1: Adding Entity Framework Core
- Exercise 2: Use Entity Framework Core to Retrieve and Store Data
- Exercise 3: Use Entity Framework Core to Connect to Microsoft SQL Server

Estimated Time: 60 minutes