# Validacija Entiteta u C# koristeći Data Annotations

U ovom poglavlju predstavljena je implementacija validacije u C# koristeći Data Annotations. Validacija entiteta omogućava definisanje pravila direktno na nivou klasa i automatski ih integriše sa ASP.NET Core-om i drugim alatima. Ovaj pristup pojednostavljuje održavanje koda i obezbeđuje da svi podaci budu validni pre nego što se dalje procesiraju.

## Primer Entiteta sa Data Annotations

Sledeći kod prikazuje klasu 'User' sa atributima za validaciju kao što su [Required], [StringLength], i [EmailAddress]. Ovi atributi definišu pravila koja će se primeniti na podatke.

|  |
| --- |
| using System.ComponentModel.DataAnnotations;  public class User {  [Key]  public int IdUser { get; set; }   [Required(ErrorMessage = "Ime je obavezno.")]  [StringLength(50, ErrorMessage = "Ime ne sme biti duze od 50 karaktera.")]  public string? FrstName { get; set; }   [Required(ErrorMessage = "Prezime je obavezno.")]  [StringLength(50, ErrorMessage = "Prezime ne sme biti duže od 50 karaktera.")]  public string? LastName { get; set; }   [Required(ErrorMessage = "Email je obavezan.")]  [EmailAddress(ErrorMessage = "Email adresa nije validna.")]  public string? Email { get; set; }   [Required(ErrorMessage = "Lozinka je obavezna.")]  [MinLength(6, ErrorMessage = "Lozinka mora imati najmanje 6 karaktera.")]  public string? PasswordHash { get; set; }   public bool Satus { get; set; }   [Required]  public DateTime CreatedDate { get; set; }   [Required(ErrorMessage = "Uloga korisnika je obavezna.")]  public int IdRole { get; set; } } |

## Validacija Korišćenjem Validator.TryValidateObject

Sledeći kod prikazuje kako validirati entitet koristeći klasu Validator iz .NET-a. Metoda TryValidateObject omogućava ručnu validaciju objekta prema pravilima definisanim u klasi.

|  |
| --- |
| using System.ComponentModel.DataAnnotations;  public ResultWrapper Add(User user) {  var validationResults = new List<ValidationResult>();  var validationContext = new ValidationContext(user, null, null);   if (!Validator.TryValidateObject(user, validationContext, validationResults, true))  {  return new ResultWrapper  {  Message = string.Join(", ", validationResults.Select(vr => vr.ErrorMessage)),  Success = false  };  }   user.PasswordHash = HashingHelper.CreateHash(user.PasswordHash!);   if (userRepository.Add(user))  {  return new ResultWrapper  {  Message = "Uspešno dodat.",  Success = true  };  }  else  {  return new ResultWrapper  {  Message = "Greška prilikom dodavanja korisnika.",  Success = false  };  } } |

Korišćenje Data Annotations atributa za validaciju u C# olakšava implementaciju pravila validacije i održavanje koda. Metoda Validator.TryValidateObject pruža fleksibilnost za ručnu validaciju, dok se validacija u ASP.NET Core-u automatski integriše sa modelima. Ovaj pristup je idealan za većinu aplikacija koje zahtevaju jednostavna pravila validacije.

**2. Kreiranje DTO klase i spajanje tabela za izveštaj**

U ovom poglavlju, prikazaćemo proces kreiranja DTO (Data Transfer Object) klase i implementacije spajanja podataka iz više tabela pomoću SQL JOIN naredbi. Naš cilj je generisanje izveštaja koji sadrži informacije o korisnicima i njihovim porudžbinama, uključujući ime i prezime korisnika, naziv proizvoda, cenu i druge relevantne podatke.

**Kreiranje DTO klase**

DTO klasa se koristi za prenos podataka između slojeva aplikacije. Kreiraćemo klasu koja odgovara strukturi našeg izveštaja.

**Kod**

|  |
| --- |
| public class UserOrderDTO  {  public string FirstName { get; set; }  public string LastName { get; set; }  public string ProductName { get; set; }  public decimal Price { get; set; }  public DateTime OrderDate { get; set; }  …..  } |

**SQL JOIN za spajanje tabela**

**Radimo sa sledećim tabelama:**

* **Users** - sadrži informacije o korisnicima (kolone: UserId, FirstName, LastName….).
* **Orders** - sadrži informacije o porudžbinama (kolone: OrderId, UserId, ProductId, OrderDate…..).
* **Products** - sadrži informacije o proizvodima (kolone: ProductId, ProductName, Price…).

Cilj je spojiti tabele tako da generišemo izveštaj koji sadrži: Ime, Prezime, Naziv proizvoda, Cena, Datum porudžbine.

Prvo se dodaje potpis metode u interfejs **OrderRepository**

|  |
| --- |
| public interface IOrderRepository :IRepository<Order>  {  int GetOrderByUserId(int userId);  List<UserOrderDTO> GetAllUserOrdersDetails();  } |

Zatim se vrši implementacija GetAllUserOrdersDetails(), u klasi OrderRepository

|  |
| --- |
| public List<UserOrderDTO> GetAllUserOrdersDetails()  {  List<UserOrderDTO> list = new List<UserOrderDTO>();  using (SqlConnection sqlConnection = new SqlConnection())  {  sqlConnection.ConnectionString = ConnectionBase.ConnectionString;  sqlConnection.Open();  SqlCommand sqlCommand = sqlConnection.CreateCommand();  sqlCommand.CommandText = "select u.\*, p.\*, o.\* " +  "\r\nfrom Users u inner join Orders o\r\n" +  "on u.IdUser = o.IdUser inner join\r\nProducts p " +  "on p.IdProduct = o.IdProduct";  SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();  while (sqlDataReader.Read())  {  UserOrderDTO userOrder = new UserOrderDTO();  userOrder.IdUser = sqlDataReader.GetInt32(0);  userOrder.FrstName = sqlDataReader.GetString(1);  userOrder.LastName = sqlDataReader.GetString(2);  userOrder.Email = sqlDataReader.GetString(3);  userOrder.PasswordHash = sqlDataReader.GetString(4);  userOrder.Satus = sqlDataReader.GetBoolean(5);  userOrder.CreatedDate = sqlDataReader.GetDateTime(6);  userOrder.IdRole = sqlDataReader.GetInt32(7);  userOrder.IdProduct = sqlDataReader.GetInt32(8);  userOrder.Name = sqlDataReader.GetString(9);  userOrder.Description = sqlDataReader.GetString(10);  userOrder.Price = sqlDataReader.GetDecimal(11);  userOrder.CountProduct = sqlDataReader.GetInt32(12);  userOrder.IdCategory = sqlDataReader.GetInt32(13);  userOrder.IdOrder = sqlDataReader.GetInt32(14);  userOrder.TotalPrice = sqlDataReader.GetDecimal(15);  userOrder.OrderDate = sqlDataReader.GetDateTime(16);  list.Add(userOrder);  }  }  return list;  } |