ConfiguredPokemonTests.cs

```
using ErrorOr;
using FluentAssertions;
using Xunit;

namespace Domain.UnitTests;

public class ConfiguredPokemonTests
{
    #region CreateFromRequest Tests

    [Fact]
    public void CreateFromRequest_WithValidData_ShouldCreateConfiguredPokemon()
    {
        // Arrange
        var request = DomainTestUtil.CreateValidPokemonRequest();

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeFalse();
        var pokemon = result.Value;

        pokemon.Name.Should().Be(request.Name);
        pokemon.HP.Should().Be(request.HP);
        pokemon.Attack.Should().Be(request.Attack);
        pokemon.Defense.Should().Be(request.Defense);
        pokemon.SpecialAttack.Should().Be(request.SpecialAttack);
        pokemon.SpecialDefense.Should().Be(request.SpecialDefense);
        pokemon.Speed.Should().Be(request.Speed);
        pokemon.AbilityId.Should().Be(request.AbilityId);

        // Verify IVs
        pokemon.HpIv.Should().Be(request.HpIv);
        pokemon.AttackIv.Should().Be(request.AttackIv);
        pokemon.DefenseIv.Should().Be(request.DefenseIv);
        pokemon.SpecialAttackIv.Should().Be(request.SpecialAttackIv);
        pokemon.SpecialDefenseIv.Should().Be(request.SpecialDefenseIv);
        pokemon.SpeedIv.Should().Be(request.SpeedIv);

        // Verify EVs
        pokemon.HpEv.Should().Be(request.HpEv);
        pokemon.AttackEv.Should().Be(request.AttackEv);
        pokemon.DefenseEv.Should().Be(request.DefenseEv);
        pokemon.SpecialAttackEv.Should().Be(request.SpecialAttackEv);
        pokemon.SpecialDefenseEv.Should().Be(request.SpecialDefenseEv);
        pokemon.SpeedEv.Should().Be(request.SpeedEv);
    }

    #region Name Validation Tests
```

```
    [Theory]
    [InlineData(null)]
    [InlineData("")]
    [InlineData(" ")]
    [InlineData("\t")]
    [InlineData("\n")]
    public void
CreateFromRequest_WithInvalidName_ShouldReturnValidationError(string invalidName)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithName(invalidName);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("Pokemon name cannot be empty");
    }

    #endregion

    #region Stats Validation Tests

    [Theory]
    [InlineData(0)]
    [InlineData(-1)]
    [InlineData(-100)]
    public void CreateFromRequest_WithInvalidHP_ShouldReturnValidationError(int
invalidHp)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithHp(invalidHp);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("Pokemon stats must be non-
negative (HP must be positive)");
    }

    [Theory]
    [InlineData(-1)]
    [InlineData(-50)]
    public void
CreateFromRequest_WithNegativeAttack_ShouldReturnValidationError(int
invalidAttack)
    {
        // Arrange
```

```
        var request =
DomainTestUtil.CreatePokemonRequestWithAttack(invalidAttack);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("Pokemon stats must be non-
negative (HP must be positive)");
    }

    [Theory]
    [InlineData(-1)]
    [InlineData(-25)]
    public void
CreateFromRequest_WithNegativeDefense_ShouldReturnValidationError(int
invalidDefense)
    {
        // Arrange
        var request =
DomainTestUtil.CreatePokemonRequestWithDefense(invalidDefense);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Description.Should().Be("Pokemon stats must be non-
negative (HP must be positive)");
    }

    [Theory]
    [InlineData(-1)]
    [InlineData(-75)]
    public void
CreateFromRequest_WithNegativeSpecialAttack_ShouldReturnValidationError(int
invalidSpecialAttack)
    {
        // Arrange
        var request =
DomainTestUtil.CreatePokemonRequestWithSpecialAttack(invalidSpecialAttack);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Description.Should().Be("Pokemon stats must be non-
negative (HP must be positive)");
    }

    [Theory]
```

```
    [InlineData(-1)]
    [InlineData(-30)]
    public void
CreateFromRequest_WithNegativeSpecialDefense_ShouldReturnValidationError(int
invalidSpecialDefense)
    {
        // Arrange
        var request =
DomainTestUtil.CreatePokemonRequestWithSpecialDefense(invalidSpecialDefense);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Description.Should().Be("Pokemon stats must be non-
negative (HP must be positive)");
    }

    [Theory]
    [InlineData(-1)]
    [InlineData(-40)]
    public void
CreateFromRequest_WithNegativeSpeed_ShouldReturnValidationError(int invalidSpeed)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithSpeed(invalidSpeed);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Description.Should().Be("Pokemon stats must be non-
negative (HP must be positive)");
    }

    #endregion

    #region IV Validation Tests

    [Theory]
    [InlineData(0)] // Boundary: minimum valid
    [InlineData(15)] // Typical valid
    [InlineData(31)] // Boundary: maximum valid
    public void CreateFromRequest_WithValidIVs_ShouldCreatePokemon(int validIV)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithIVs(validIV, validIV,
validIV, validIV, validIV, validIV);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);
```

```
        // Assert
        result.IsError.Should().BeFalse();
    }

    [Theory]
    [InlineData(-1)] // Negative
    [InlineData(-10)]
    [InlineData(32)] // Too high
    [InlineData(50)]
    [InlineData(100)]
    public void CreateFromRequest_WithInvalidHpIV_ShouldReturnValidationError(int
invalidIV)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithIVs(invalidIV, 31,
31, 31, 31, 31);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("IVs must be between 0 and 31");
    }

    [Theory]
    [InlineData(-1)] // Negative
    [InlineData(32)] // Too high
    public void
CreateFromRequest_WithInvalidAttackIV_ShouldReturnValidationError(int invalidIV)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithIVs(31, invalidIV,
31, 31, 31, 31);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Description.Should().Be("IVs must be between 0 and 31");
    }

    #endregion

    #region EV Validation Tests

    [Theory]
    [InlineData(0)] // Boundary: minimum valid
    [InlineData(128)] // Typical valid
    [InlineData(255)] // Boundary: maximum valid
    public void CreateFromRequest_WithValidEVs_ShouldCreatePokemon(int validEV)
    {
```

```
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithEVs(validEV, 0, 0, 0,
0, 0);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeFalse();
    }

    [Theory]
    [InlineData(-1)] // Negative
    [InlineData(-50)]
    [InlineData(256)] // Too high
    [InlineData(300)]
    public void
CreateFromRequest_WithInvalidEVRange_ShouldReturnValidationError(int invalidEV)
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithEVs(invalidEV, 0, 0,
0, 0, 0);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("EVs must be between 0 and
255");
    }

    [Fact]
    public void CreateFromRequest_WithEVTotalExactly510_ShouldCreatePokemon()
    {
        // Arrange
        var request = DomainTestUtil.CreatePokemonRequestWithEVs(255, 255, 0, 0,
0, 0);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeFalse();
    }

    [Theory]
    [InlineData(511)] // Too high
    [InlineData(600)]
    [InlineData(1000)]
    public void
CreateFromRequest_WithEVTotalExceeding510_ShouldReturnValidationError(int totalEV)
    {
```

```
        // Arrange
        var evPerStat = totalEV / 6;
        var remainder = totalEV % 6;
        var request = DomainTestUtil.CreatePokemonRequestWithEVs(
            evPerStat + remainder, evPerStat, evPerStat, evPerStat, evPerStat,
evPerStat);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("Total EVs cannot exceed 510");
    }

    #endregion

    #region AbilityId Validation Tests

    [Theory]
    [InlineData(0)] // TODO: Check if 0 is valid
    [InlineData(-1)]
    [InlineData(-100)]
    public void
CreateFromRequest_WithInvalidAbilityId_ShouldReturnValidationError(int
invalidAbilityId)
    {
        // Arrange
        var request =
DomainTestUtil.CreatePokemonRequestWithAbilityId(invalidAbilityId);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Validation);
        result.FirstError.Description.Should().Be("AbilityId must be valid");
    }

    [Theory]
    [InlineData(1)]
    [InlineData(50)]
    [InlineData(999)]
    public void CreateFromRequest_WithValidAbilityId_ShouldCreatePokemon(int
validAbilityId)
    {
        // Arrange
        var request =
DomainTestUtil.CreatePokemonRequestWithAbilityId(validAbilityId);

        // Act
        var result = ConfiguredPokemon.CreateFromRequest(request);
```

```
        // Assert
        result.IsError.Should().BeFalse();
        result.Value.AbilityId.Should().Be(validAbilityId);
    }

    #endregion

    #endregion

    #region AddMove Tests

    [Fact]
    public void AddMove_WithValidMove_ShouldAddMoveSuccessfully()
    {
        // Arrange
        var pokemon = DomainTestUtil.CreateValidConfiguredPokemon();
        var move = DomainTestUtil.CreateValidConfiguredMove(1);

        // Act
        var result = pokemon.AddMove(move);

        // Assert
        result.IsError.Should().BeFalse();
        pokemon.Moves.Should().HaveCount(1);
        pokemon.Moves.Should().Contain(move);
    }

    [Fact]
    public void AddMove_WithFourMoves_ShouldAddAllMovesSuccessfully()
    {
        // Arrange
        var pokemon = DomainTestUtil.CreateValidConfiguredPokemon();
        var moves = new[]
        {
            DomainTestUtil.CreateValidConfiguredMove(1),
            DomainTestUtil.CreateValidConfiguredMove(2),
            DomainTestUtil.CreateValidConfiguredMove(3),
            DomainTestUtil.CreateValidConfiguredMove(4)
        };

        // Act & Assert
        foreach (var move in moves)
        {
            var result = pokemon.AddMove(move);
            result.IsError.Should().BeFalse();
        }

        pokemon.Moves.Should().HaveCount(4);
        pokemon.Moves.Should().BeEquivalentTo(moves);
    }

    [Fact]
    public void AddMove_WhenAlreadyHasFourMoves_ShouldReturnConflictError()
```

```
    {
        // Arrange
        var pokemon = DomainTestUtil.CreateValidConfiguredPokemon();

        // Add 4 moves first
        for (var i = 1; i <= 4; i++)
pokemon.AddMove(DomainTestUtil.CreateValidConfiguredMove(i));

        var fifthMove = DomainTestUtil.CreateValidConfiguredMove(5);

        // Act
        var result = pokemon.AddMove(fifthMove);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Conflict);
        result.FirstError.Description.Should().Be("A Pokémon cannot have more than
4 moves");
        pokemon.Moves.Should().HaveCount(4);
    }

    [Fact]
    public void AddMove_WithDuplicateMoveId_ShouldReturnConflictError()
    {
        // Arrange
        var pokemon = DomainTestUtil.CreateValidConfiguredPokemon();
        var move1 = DomainTestUtil.CreateValidConfiguredMove(1);
        var move2 = DomainTestUtil.CreateValidConfiguredMove(1); // Same MoveId

        pokemon.AddMove(move1);

        // Act
        var result = pokemon.AddMove(move2);

        // Assert
        result.IsError.Should().BeTrue();
        result.FirstError.Type.Should().Be(ErrorType.Conflict);
        result.FirstError.Description.Should().Be("This move is already added to
this Pokémon");
        pokemon.Moves.Should().HaveCount(1);
    }

    [Theory]
    [InlineData(1)]
    [InlineData(2)]
    [InlineData(3)]
    public void AddMove_WithVariousNumberOfMoves_ShouldMaintainCorrectCount(int
numberOfMoves)
    {
        // Arrange
        var pokemon = DomainTestUtil.CreateValidConfiguredPokemon();

        // Act
        for (var i = 1; i <= numberOfMoves; i++)
```

```
        {
            var result =
pokemon.AddMove(DomainTestUtil.CreateValidConfiguredMove(i));
            result.IsError.Should().BeFalse();
        }

        // Assert
        pokemon.Moves.Should().HaveCount(numberOfMoves);
    }

    #endregion
}
```

Path: `./Backend/Domain.UnitTests/ConfiguredPokemonTests.cs`