**DIRS21 Mapping System Documentation**

**System Architecture**

The mapping system uses a plugin-based architecture with these core components:

* **MapHandler**: Main entry point that orchestrates mapping operations
* **MapperRegistry**: Stores and retrieves mappers
* **ValidationService**: Handles pre and post-mapping validation
* **MapperFactory**: Creates mapper instances
* **Auto-discovery**: Automatically finds and registers mappers/validators at startup

**Key Classes and Methods**

**MapHandler**

public object Map(object data, string sourceType, string targetType)

Orchestrates validation and transformation. This is the main method as per requirements.

**IMapper Interface**

string SourceType { get; }

string TargetType { get; }

object Map(object source);

Contract for all mapper implementations.

**MapperBase<TSource, TTarget>**

protected abstract TTarget MapInternal(TSource source);

Base class providing type-safe mapping.

**IValidatorType**

string TypeName { get; }

ValidationResult Validate(object data);

Self-registering validators.

**ServiceCollectionExtensions**

services.AddDIRS21Mapping();

services.AddMappersFromAssembly();

services.AddValidatorsFromAssembly();

Clean DI registration.

**Extending the System**

To add a new partner (e.g., Booking.com):

**1. Create Model**

public class BookingReservation

{

public string BookingReference { get; set; }

public string GuestFullName { get; set; }

public string ArrivalDate { get; set; }

}

**2. Create Mapper**

public class ReservationToBookingMapper : MapperBase<Reservation, BookingReservation>

{

public override string SourceType => "Model.Reservation";

public override string TargetType => "Booking.Reservation";

protected override BookingReservation MapInternal(Reservation source)

{

return new BookingReservation

{

BookingReference = $"BK-{source.Id}",

GuestFullName = source.GuestName,

ArrivalDate = source.CheckIn.ToString("yyyy/MM/dd")

};

}

}

**3. Create Validator (Optional)**

public class BookingReservationValidator : ITypedValidator

{

public string TypeName => "Booking.Reservation";

public ValidationResult Validate(object data)

{

// Validation logic

}

}

That's it! The mapper and validator are automatically discovered - no registration needed.

**Assumptions**

1. **Type Naming**: All types follow pattern {Domain}.{Entity} (e.g., "Model.Reservation", "Google.Room")
2. **Models**: Models have public properties with getters and setters
3. **Mappers:** Stateless and thread-safe
4. **Auto-discovery**: All assemblies are scanned
5. **Validation**: Optional but recommended

**Limitations**

1. **Type Safety**: Map returns object, requires casting
2. **No Caching**: Mappers created on each request
3. **No Async Support**: Async method just wraps sync call
4. **No Configuration**: All mappings must be coded
5. **Validation Scope**: Object-level only, no field-level validation

**Design Patterns Used**

* **Strategy Pattern**: Each mapper is a strategy
* **Factory Pattern**: MapperFactory creates mappers
* **Registry Pattern**: Central mapper storage
* **Template Method**: MapperBase defines algorithm
* **Dependency Injection**: All dependencies injected