Extending the Mapping System

* In this project, we assume that internal DIRS21 data models are already available within our codebase, allowing direct access to these models. The system is designed to automatically generate only the partner-specific data models based on configuration. Therefore, we focus on dynamically creating external models while leveraging the pre-existing internal DIRS21 data models directly in our code.

1. Adding New Mappings

Adding new mappings involves updating the JSON configuration file (mapping-config.json) with the new source and target mappings. The system dynamically uses this configuration to determine how fields should be mapped between source and target objects.

Steps to Add New Mappings

1. **Open the JSON Configuration File (mapping-config.json)**: Locate the configuration file in your project directory. This file defines the mapping rules for the system.
2. **Define the Target Type in the Types Section**: If the target type is new, define it in the Types section with its name and properties.

```
"Types": [
  {
    "Name": "NewTargetType",
    "Properties": {
      "TargetField1": "string",
      "TargetField2": "DateTime"
    }
  }
]
```

3. **Add a New Mapping in the Mappings Section**: Create a new mapping rule between the source and target types. Specify the SourceType, TargetType, and a list of fields, each defining a source field, target field, and any conversion rules.

```
"Mappings": [
  {
    "SourceType": "DIRS21.NewSourceType",
    "TargetType": "NewTargetType",
    "GenerateClassFlag": true,
    "Fields": {
      "SourceField1": { "TargetField": "TargetField1", "ConversionType": "StringToInt" },
      "SourceField2": { "TargetField": "TargetField2", "ConversionType": "DateTimeToString",
"Format": "yyyy-MM-dd" }
    }
  }
]
```

4. **Specify Conversions (Optional)**: If the data types between source and target fields differ, specify the ConversionType and optionally, the Format for date/time fields. Supported conversions may include:
    - "DateTimeToString" with a date format (e.g., "yyyy-MM-dd").

    - "StringToDateTime" with an expected string format.
    - "IntToString" or "StringToInt" for numeric conversions.
5. **Save and Reload**: After updating the JSON configuration, save it. The system will use the updated mappings on the next execution without needing code changes.

Example: Adding a New Mapping

Suppose you want to map DIRS21.Room to an external ExternalRoom format.

1. **Define ExternalRoom in Types**:

```
{
  "Name": "ExternalRoom",
  "Properties": {
    "ExternalRoomId": "string",
    "RoomName": "string",
    "Capacity": "int32"
  }
}
```

2. **Add the Mapping in Mappings**:

```
{
  "SourceType": "DIRS21.Room",
  "TargetType": "ExternalRoom",
  "GenerateClassFlag": true,
  "Fields": {
    "Id": { "TargetField": "ExternalRoomId", "ConversionType": "IntToString" },
    "Name": { "TargetField": "RoomName" },
    "MaxOccupancy": { "TargetField": "Capacity" }
  }
}
```

---

2. Adding New Data Sources (Repositories)

To add support for new configuration sources (e.g., loading from XML or a database), implement IJsonRepo or create a similar interface that the Application layer can call.

Steps to Add a New Repository

1. **Define a New Interface if Needed**: If the data source format requires a different repository, define a new interface (e.g., IXmlRepo).
2. **Implement the Repository in Infrastructure Layer**: Create a new class in the Infrastructure layer that implements IJsonRepo (or your new interface) and provides the loading logic for the new data source.

```
public class XmlRepo : IJsonRepo
{
  public async Task<MappingConfiguration> LoadJsonFileAsync(string filePath)
  {
    // Implement XML or database loading logic here
  }
```

```
}
```

3. **Inject the New Repository in Program.cs**: Register the new repository in Program.cs, allowing the Application layer to use it.

```
services.AddTransient<IJsonRepo, XmlRepo>();
```

---

3. Adding a New Mapping Algorithm

To support alternative mapping strategies, implement the IMapAlgorithm interface with a custom mapping class.

Steps to Add a New Mapping Algorithm

1. **Create a New Class That Implements IMapAlgorithm**: Define a custom class that implements the Execute method from IMapAlgorithm.

```
public class CustomMapper : IMapAlgorithm
{
    public object Execute(object source, string sourceType, string targetType)
    {
        // Implement custom mapping logic here
    }
}
```

2. **Register the New Mapper in Program.cs**: Inject the new mapping algorithm into MapHandler or directly into the Program.cs file.

```
services.AddTransient<IMapAlgorithm, CustomMapper>();
```

3. **Configure Usage in MapHandler**: In MapHandler, specify when to use the custom algorithm, or conditionally select it based on mapping needs.

Summary

These instructions guide you through extending the Mapping System by adding new mappings, conversions, data sources, and mapping algorithms. Each extension is designed to minimize code changes and maximize flexibility, following Clean Architecture principles:

- **New Mappings**: Update the mapping-config.json file.
- **Custom Conversions**: Extend the ApplyConversion method in DynamicMapper.
- **New Data Sources**: Implement IJsonRepo or a similar repository interface.
- **New Mapping Algorithms**: Implement IMapAlgorithm for alternative mapping strategies.

By following these steps, you can extend the system seamlessly, ensuring it remains flexible and adaptable to new mapping requirements.