**Service layer**

**What is a Service layer?**

The service layer provides a bridge between the presentation layer and the model layer of domain logic and resource-specific data. This is implemented using service contracts, which are defined using PHP interfaces.

In general, the service layer:

1. Resides below the presentation layer and above the domain layer.
2. Contains service contracts, which define how the implementation will behave.
3. Provides an easy way to access the REST/SOAP API framework code (which also resides above the service contracts). You can bind service contracts to web service APIs in configuration files — no coding required.
4. Provides a stable API for other modules to call into.

**Who accesses the service layer?**

All calls from web service interfaces, or users working with your storefront (that is, controller-initiated requests), are typically routed through the service layer. We strongly encourage the use of service contracts 74o call business logic.

External applications can make requests for business logic with simple SOAP and REST calls. With some simple XML or JSON, you can expose the service layer’s PHP API and make it accessible to REST or SOAP web services. Once implemented, a web service can make a single API call and return an information-rich data structure.

Service contract clients include:

1. Controllers (initiated by actions of users of the storefront)
2. Web services (SOAP and REST API calls)
3. Other Magento modules through service contracts

**Service contract anatomy**

The service contract of a module is defined by the set of interfaces in the module’s /Api directory.

This directory contains:

1. Service interfaces in the /Api namespace of the module (Catalog API).
2. Data (or entity) interfaces in the Api/Data directory (Catalog API/Data). Data entities\* are data structures passed to and returned from service interfaces.

Files in the data directory contain get() and set() methods for entries in the entity table and extension attributes.

Typically, service contracts provide three distinct types of interfaces:

1. Repository interfaces
2. Management interfaces
3. Metadata interfaces

However, there is no requirement that service contracts conform to all three patterns.

**Advantages of service contracts**

Service contracts allow you to add a new customer extension that adds or changes business logic-level resource models without breaking the system.

This is done using the <preference> element in a custom module’s dependency injection configuration file (di.xml) file.

The di.xml file specifies which PHP class to use for the interface Magento\Customer\Api\CustomerRepositoryInterface.

Another module can change this interface file by specifying a different class name. However, if the client code uses the interface definition only, no class change is necessary.