jamk.fi

Magento 2 – Dynamics 365 Business Central Connector

Dominik Rohaľ

Bachelor's thesis
May 2020
School of Technology
Information and Communications Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Roháľ, Dominik	Bachelor's thesis	May 2020
		Language of publication: English
	Number of pages: 52	Permission for web publication: yes

Title of publication

Magento 2 – Dynamics 365 Business Central Connector

Degree program

Information and Communications Technology

Supervisor(s)

Salmikangas, Esa

Assigned by

Solteq Oyj

Abstract

There was a need for a universal solution allowing integrating Microsoft Dynamics 365 Business Central enterprise resource planning system into Magento 2 online stores, on the market. The currently used solutions required development work on the Microsoft Dynamics 365 Business Central's side. However, these two systems were often developed and maintained by different vendors. Connecting these systems required close cooperation of multiple development teams.

A new solution, not requiring a development work on the enterprise resource planning system's side, needed to be developed. The development of the solution required understanding both systems and its implementation required programming in PHP programming language.

The implementation consists of multiple modules that are highly reusable and extendable. The modules can be easily installed in any Magento 2 project and allow merchants to transfer products and inventories from their enterprise resource planning system to their Magento 2 online store, and also transfer orders created in the online store to the enterprise resource planning system.

The implementation proved that the integration of these two systems can be done without development work on the enterprise resource planning system's side. All the expectations were met, and the implemented solution can be offered to the potential customers.

Keywords/tags (subjects)

PHP, integration, Magento 2, Microsoft Dynamics 365 Business Central, e-commerce, enterprise resource planning system

Miscellaneous

Contents

Figures

Tables

Acronyms

ACL	Access Control List
API	Application Program Interface
CLI	Command-line Interface
CMS	Content Management System
EAV	Entity Attribute Value
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MSI	Multi-Source Inventory
OData	Open Data Protocol
SEO	Search Engine Optimization
SKU	Stock-keeping Unit

1 Introduction

Over the past few decades, technologies have improved drastically and many new technologies have been invented. The technological progress has a significant impact on the society. The way how consumer goods are produced and delivered to the consumers is very different from what it was in the past. New options and possibilities allow a better cooperation between different producers. Easier communication allows smaller producers to focus on smaller parts of final products, put them together and sell them through merchants to end consumers or other companies for further process or use. An electronic system providing communication between different companies or their units in the chain of production and delivery of products or services to end consumers, other companies or government is called an e-commerce system, also known as e-commerce. For a regular person, e-commerce is an online store that can be used to order goods or services regardless of time or place.

The number of online shoppers increases every year. This has multiple causes: young people more familiar with modern technologies are growing up and gaining responsibilities, including the responsibility of buying goods on their own. Also, older more traditional people often find online shopping more convenient than a traditional visit to a physical store or market. People prefer the ability to compare various products offered by multiple sellers and choose the best option without physical visit to stores during opening hours. This way, people can also avoid overcrowded shopping malls that are often tens of kilometers away from their homes, especially in less densely populated countries, such as Finland.

The growing popularity of online shopping is not without a response. Many new companies have been founded with their focus on online selling. Also, retail sellers known for their physical stores have entered the online market. Having an online system allowing cooperation with business partners or reaching end consumers is almost a necessity for many businesses.

In many cases, creating an e-commerce system is a very repetitive task. Most of the systems can be grouped where each system of a group needs to have the same or

very similar main functionalities. However, every system has its specifics. Thus, developing every e-commerce system from scratch would not be efficient and creating a new system by copying an existing system would not solve the problem. The solution is pre-developed software that can be reused, modified and extended by adding new functionalities to meet the needs of a specific business. This software is called an e-commerce platform.

There are many e-commerce platforms available on the market. Several e-commerce platforms are used by the hosting company Solteq Oyj which develops and delivers e-commerce solutions to its customers. One of the e-commerce platforms that are used by the hosting company is Magento 2. Besides e-commerce systems, the hosting company offers solutions to its customers including enterprise resource planning (ERP) systems. Sometimes, these two systems need to be connected. The ERP system may also be connected to other systems, for example, systems used in warehouses. In these cases, the e-commerce system must send the created orders to the ERP system and it also must update product information and product availability. The piece of software which allows communication between two different systems is called a connector.

One of the ERP systems used by the hosting company in its solutions is Microsoft Dynamics 365 Business Central which is the successor of Microsoft Dynamics NAV. In some cases, the hosting company provides an e-commerce solution to its customer; however, the customer's ERP system is in the hands of other vendors and the cooperation with the other third-party companies may be complicated. The aim of this bachelor's thesis is to investigate the possibility of connecting the ERP system with the Magento 2 e-commerce system by using ERP system's built-in endpoints; thus, not requiring customizations on the ERP system's side, and implementing a universal solution installable to multiple Magento 2 installations.

The current and potential future customers of Solteq Oyj interested in implemented solutions did not agree to be mentioned in any public document, therefore the names of the customers are left out of this thesis.

2 Magento 2 overview

2.1 Magento 2 in general

Magento 2 is an e-commerce platform built on open source technology that provides online merchants with a flexible shopping cart system, as well as control over the look, content, and functionality of their online store. Magento offers powerful marketing, search engine optimization, and catalog-management tools. (Kristen 2019.)

Practically, almost every online store requires the same functionality. There are not many larger online stores on the market without a shopping cart, categories, user accounts or product pages. Except for not many exceptions, every bigger online store must have this core functionality. However, each retail seller is a specific case and requires for their online store additional functionality specific for their business. Magento 2 is available in two editions, each of them fitting the needs of different types of online sellers.

2.2 Magento 2 editions

2.2.1 Magento 2 Open Source

Magento 2 Open Source is an open-source e-commerce platform which provides a free comprehensive solution for online retailers. Developers can override core files and extend functionality, which makes the platform highly customizable and flexible to meet the merchant's needs. There is an extensive marketplace for modules to add extra functionality. (Shaun n.d.)

The Open source edition of the platform is suitable mainly for smaller businesses with limited resources that do not need or cannot afford the advanced features contained in paid platforms.

2.2.2 Magento 2 Commerce

Magento 2 Commerce comes in two versions, on-premise and cloud. Unlike the Open Source, it is not free; yet, it has more features and functionality. It is designed for

large businesses that require technical support. The pricing for Commerce is tiered depending on an e-commerce business's revenue. (Shaun n.d.)

As already mentioned above, the Magento 2 Commerce edition is not for free, and it is suitable for larger businesses that can benefit from the advanced features and functionality. Besides that, development work and extensions developed by third-party vendors are usually much more expensive in comparison to Magento 2 Opens Source edition.

2.3 Admin panel

2.3.1 Structure

Magento 2 is a robust platform providing many features. The admin panel allows admin users with applicable access rights to manage an online store. In there is a screenshot of an admin panel in a Magento 2 demo store that does not have any activated extension that would customize its appearance.

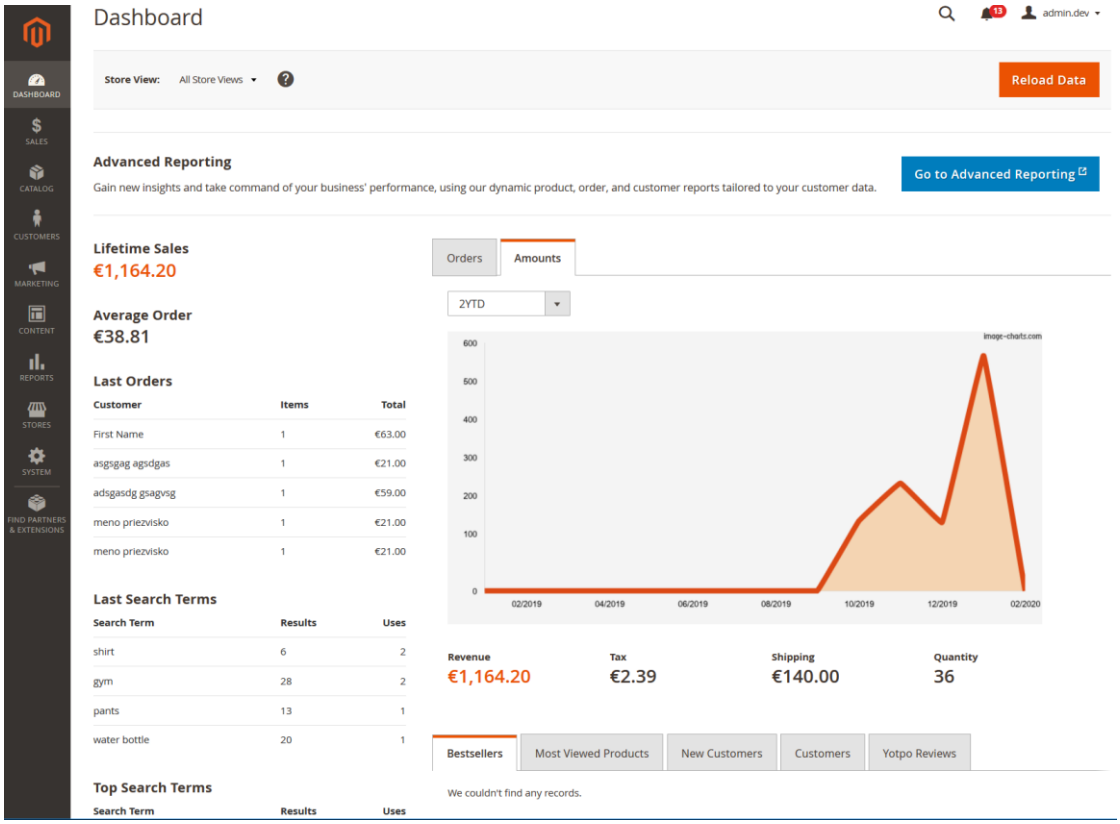

Figure 1. Admin panel

Since Magento 2 is a powerful platform that contains many functionalities that are accessible from the admin panel, they are logically sorted into multiple sections. As it can be seen in Figure 1, the main element of the admin panel is the left navigation. The left navigation is visible from every admin panel's page and provides the admin users quick access to the admin panel's sections.

The left navigation contains at least the following sections:

- Dashboard
- Sales
- Catalog
- Customers
- Marketing
- Content
- Reports
- Stores
- System

Dashboard

The dashboard is the default startup place for admin users. shows a dashboard of a Magento 2 demo store where the admin users are provided an overview of a store's critical information, such as revenue, shipping, tax, last orders, best-selling products, or top searched terms.

Sales

Under the sales section, orders, invoices, shipments, credit memos, billing agreements, and transactions can be observed and managed.

Catalog

Products and categories can be found under this tab. Admin users can add new or edit existing categories or products.

Customers

Tools that allow admin users to manage customers are sorted in the customer section. For example, admin users can list currently logged in customers, find any customer and edit their information, admin users can also manage customer groups.

Marketing

The marketing section groups many useful marketing tools. The tools contain promotions, communication with customers (e.g. newsletter), reviews, SEO (Search Engine Optimization) and search settings.

Content

Store content and design can be managed under this section. The admin users can edit CMS (Content Management System) pages, add new elements (static blocks or widgets) to a selected page. Under this section, also a theme can be configured or changed.

Reports

Reports on the store can be found in this section. There are different kinds of reports available in the core Magento 2. For example, there are reports about orders, taxes, refunds, searched terms, and wish lists accessible in this section.

Stores

The stores section provides access to store configuration. Also, it provides access to e. g. tax configuration, product attribute and product attribute set configuration, and extension management.

System

Under this section, besides other functionality, the admin users can manage caches and indexes, create user roles and grant permissions.

Additional sections can be added by custom modules. However, according to Admin Panel Placement and Design (2019), it is not considered a good practice.

2.3.2 Admin ACL

Magento 2 Admin ACL is a robust authentication system that allows the store owners to create an access control list (ACL) which allows the store owners to define user roles and specify their access rights by selecting resources from a tree-structured list. Each admin user can be assigned to a specific user role and is thus able to access only the parts of an admin panel which are allowed for his/her user role.

A custom resource can be added to the list by creating *acl.xml* file in the module's *etc* directory. The content of the file adding a resource, with title "Solteq" and id "Solteq_Base::admin", looks like the following:

```
| <?xml version="1.0"?>
| <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:frame-
work:Acl/etc/acl.xsd">
|   <acl>
|     <resources>
|       <resource id="Magento_Backend::admin">
|         <resource id="Solteq_Base::admin" title="Solteq" sortOr-
der="1"/>
|       </resource>
|     </resources>
|   </acl>
| </config>
```

After adding nested resources to the resource with the title "Solteq", the roles resources list looks like the list illustrated in Figure 2.

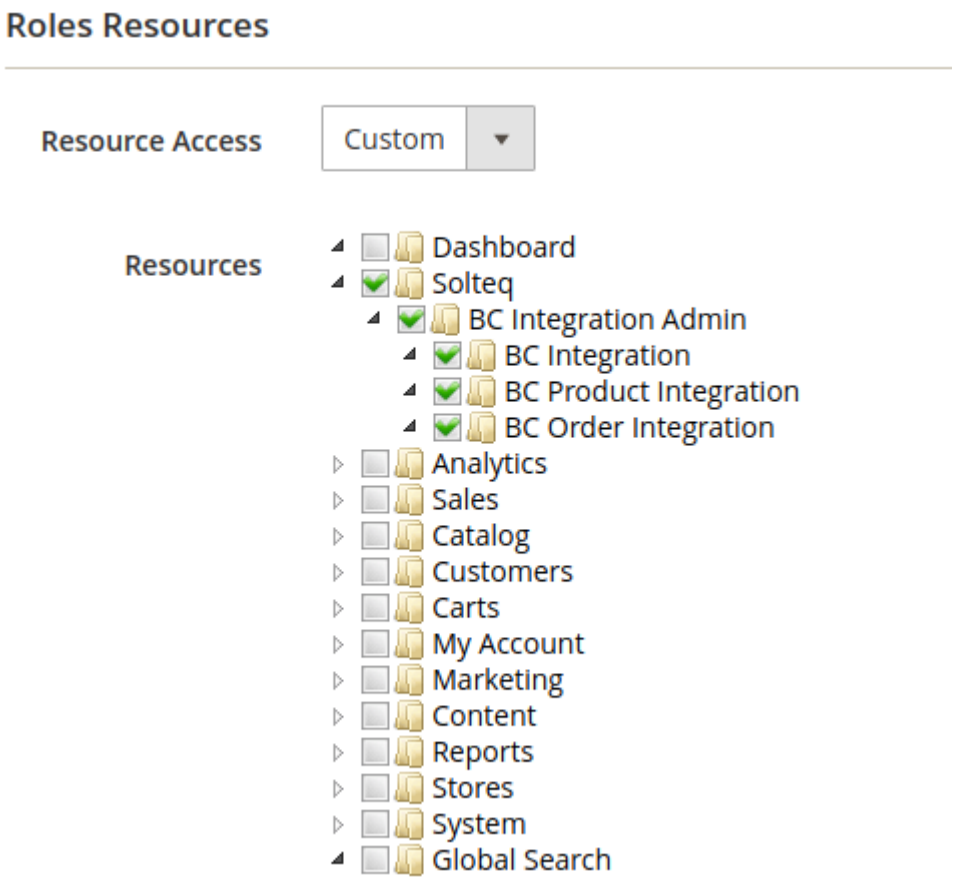

Figure 2. Roles resources list

In Figure 2, there can be seen an ACL configuration. The role resources list, or ACL list, contains resource, also called ACL rule, with title “Solteq” which also has child ACL rules. Next to each ACL rule, there is a checkbox element. Click on a checkbox grant or denies resource to a specific group of admin users. If a parent resource is not granted to an admin user, a child resource is not granted as well. For example, child resource with title “BC Integration Admin” cannot be granted without granting its parent resource with title “Solteq”.

2.4 Websites, Stores, and Views

2.4.1 Magento 2 Scope

Every Magento installation has a hierarchy of website(s), store(s), and store view(s). The term scope determines where in the hierarchy a database entity such as a prod-

uct, attribute, or category content element, or configuration setting applies. Websites, stores, and store views have one-to-many parent/child relationships. A single installation can have multiple websites, and each website can have multiple stores and store views. (Websites, Stores, and Views 2019)

Basically, scope or context determines a specific part of a Magento 2 installation.

A fresh Magento installation has a default hierarchy consisting of the main website, default store and store view.

Scope in Magento 2 has the following 4 levels:

- Global
- Website
- Store
- Store view

Global scope

According to Scope (2019), the global scope level provides system-wide settings and resources that are available throughout the Magento installation.

Website scope

A single Magento instance can have multiple websites running on different domains. A website is a top-level container that either has one store or groups multiple stores that share e.g. cart, payment gateways, and user sessions.

Store scope

Stores that do not share cart, payment gateways or user sessions must be contained by a different website. Each store can have a different appearance, categories, and product selection. All stores are managed from the same admin. A store has at least one store view.

Store view scope

Besides other reasons to have multiple store views, it allows having localization support. Support of one language can be represented as one store view. The customers can switch between different store views contained by one store using a language switcher.

2.4.2 Single store mode

A single store mode consists of one website, one store and one store view as seen in Figure 3. The single store mode can be set from the store configuration in the admin panel. The mode prevents the admin users from making different configurations at the store view or website levels.

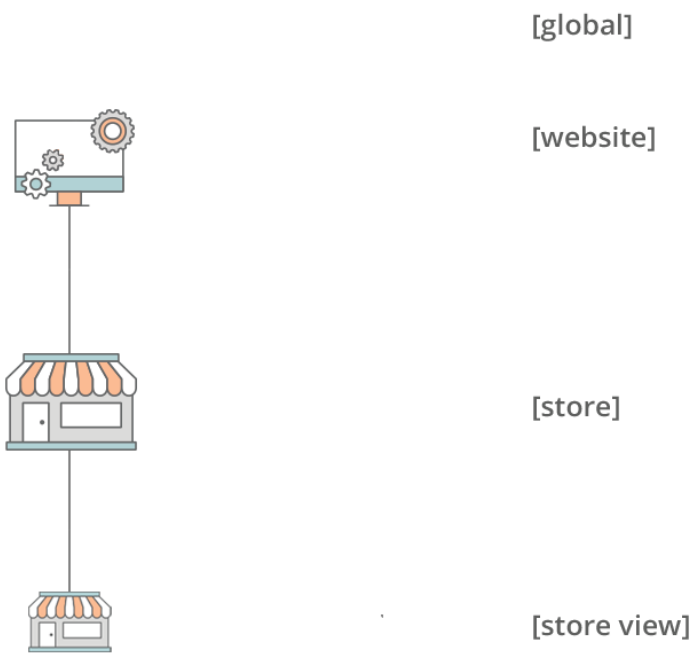

Figure 3. Single Website, Store, and View (Single Store Mode 2019)

2.4.3 Multiple stores mode

Magento 2 allows creating multiple stores in a single Magento 2 installation. The multiple stores can be created in order to run separate stores that can focus on different customer groups, sell different sorts of products and be visually different from each other. However, the stores are managed from the same admin panel and share functionalities. A hierarchy of a Magento 2 installation with multiple stores running on different websites is illustrated in Figure 4.

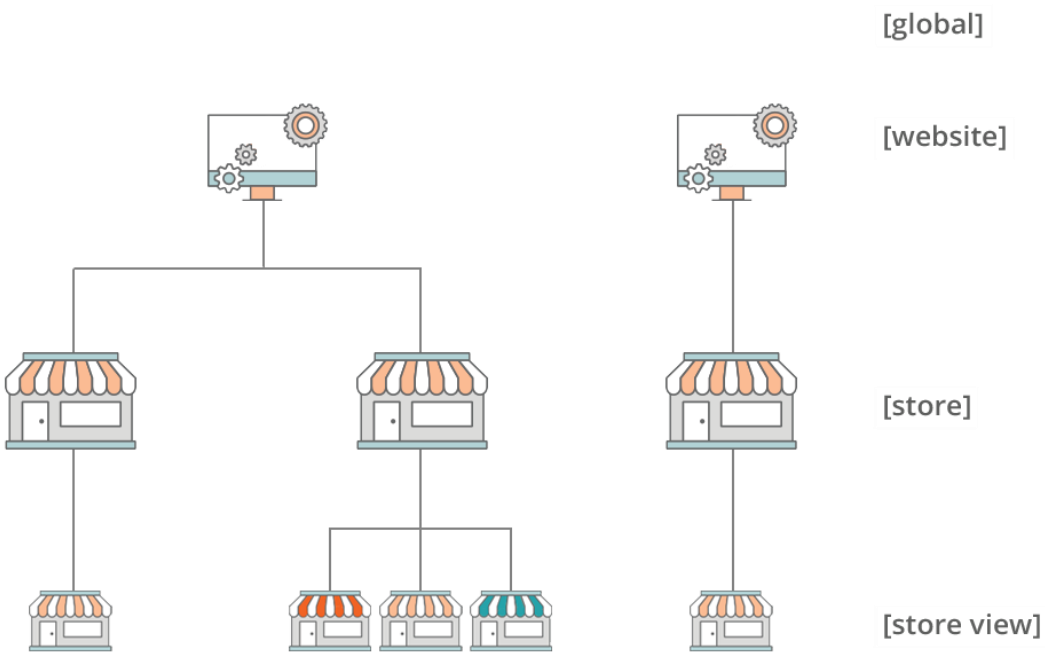

Figure 4. Hierarchy of Websites, Stores, and Store Views (Scope 2019)

2.5 Magento 2 Module overview

A module is a logical group, i.e. a directory containing blocks, controllers, helpers, models related to a specific business feature. In keeping with Magento's commitment to optimal modularity, a module encapsulates one feature and has minimal dependencies on other modules. (Module overview 2019)

The modules in Magento 2, besides providing new functionality, in many cases disable or extend an existing functionality provided either by the platform itself, as core functionality, or by any other third-party vendor, which makes Magento 2 a highly customizable platform.

One of the disadvantages of Magento 2 is that it is a robust platform not ready to be used in production immediately after installation. A larger amount of development work must be carried out when compared to smaller and simpler e-commerce platforms, which also means that it is not a suitable platform for the smallest businesses. On the other hand, Magento 2 is very powerful and provides many features that support businesses in their growth.

Fortunately, there are many Magento 2 modules or extensions developed by third-party vendors available on the market. They can be installed based on the needs of a specific business. Themes, payment gateways, modules providing shipment methods and integrations with other systems can be considered the most important modules or extensions.

2.5.1 Module installation

In Magento 2, there are two ways how to install the modules. One of the ways how they can be installed is using Composer, a package management tool. This option is typically used to install general modules provided by third-party vendors. The package management tool makes module updates easier and checks module dependencies. The second way how the modules can be installed in Magento 2 is placing module files inside a vendor folder in *app/code* directory in Magento 2 installation. This way of module installation is usually used to install custom modules developed for specific Magento 2 installation.

In many cases, the needs of a merchant's business cannot be satisfied by installing third-party modules. In these cases, custom modules fulfilling specific and often unique requirements of concrete merchants must be implemented.

2.5.2 Module configuration

Often, modules need to be configured in order to adjust provided functionality to the needs of a specific merchant. This is achieved by setting up configuration values. Each configuration value can be identified by its configuration path. A configuration path consists of section id, group id and field id, separated by slashes.

The general configuration page is accessible from the admin panel of the "Configuration" item in the "STORES" tab. The organization of the general admin page can be seen in Figure 5.

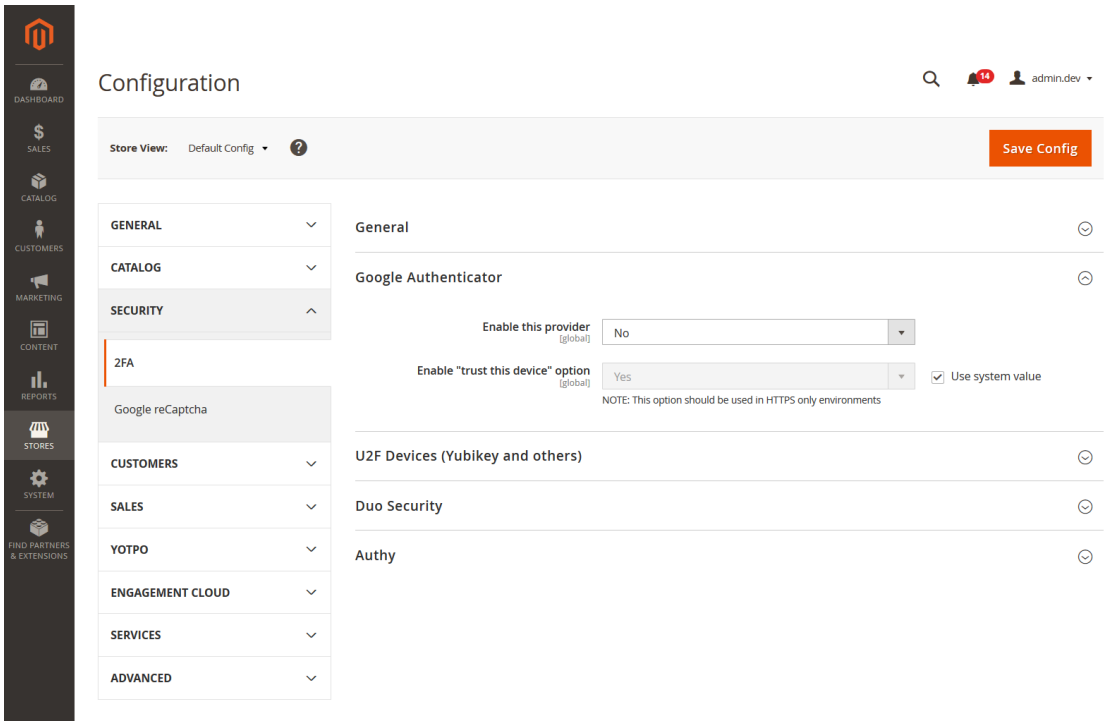

Figure 5. Store configuration page

As can be seen in Figure 5, the configuration page consists of four main elements. In the left column, tabs containing sections are placed. The content of the selected section is displayed in the right column (the two-column layout is standard for the configuration page). The content consists of configuration fields where each field represents a configuration value. Grouped fields are presented as dropdowns.

If a configuration value is saved, it is stored in database table *core_config_data*. Different scopes can have different values. It has been defined for each value at which scope level it can be configured. The default configuration values can be defined in *config.xml* file located under *etc* folder in the module's root directory. The value from the file is used only if there is no record with the path in the *core_config_data* database table. The content of the *config.xml* file defining the default configuration value with path "section/group/field" and the default value "value" has the following structure:

```
| <default>
|   <section>
|       <group>
|         <field>value</field>
|       </group>
```

```
|    </section>
| </default>
```

New tabs, sections, groups and fields can be added to general configuration by creating a *system.xml* file in *adminhtml* folder placed under module's *etc* directory. The file contains the defined elements with their sort orders, resources (used for ACL) and field scope levels.

2.5.3 Cron jobs and commands

In Magento 2 installation, there are many processes triggered at some point, then it takes some amount of time till the execution is done and during this time they perform some actions, e.g. a newsletter. The newsletter is sent to the customers periodically. The process of sending the newsletter to the customers is started at some point, it takes time till the email with the newsletter is sent to all customers and once all the emails have been sent, the functionality is not active until it is triggered again. This also applies to integrations.

Magento 2 provides a mechanism which triggers processes running in Magento 2 installations. The mechanism or program is called cron. The cron allows to run commands or scripts automatically at a specified time. Each module in Magento 2 can have its own *crontab.xml* file which is in module's *etc* directory. In the *crontab.xml* file a list of jobs is defined. Each job has its name, path to a class containing the executed method and the method name executed by the cron job. Not less importantly, a schedule is also defined there. The schedule uses cron syntax known from Linux operating system. Cron jobs are sorted into cron groups where only one job of a group can be run at the same time. A *crontab.xml* file can look like the following:

```
| <?xml version="1.0" ?>
| <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Ma-
gento_Cron:etc/crontab.xsd">
|     <group id="default">
|         <job instance="Solteq\BcIntegration\Cron\Test" method="exe-
cute" name="solteq_bcintegration_cron_test">
|             <schedule>0 4 * * *</schedule>
|         </job>
|     </group>
| </config>
```

The code above defines a cron job in group “default” which executes the method execute implemented in class Test, the file *Test.php* of which is in directory Cron of BcIntegration module in Solteq vendor folder. The method is executed every day at 4 a.m.

In some cases, scheduled cron jobs are not enough. For example, initial product import cannot wait till it is scheduled. In this and other similar cases, there is a need for a way which allows to trigger processes in Magento 2 at any time. One of the ways is defining a custom command, which triggers the process by executing the command in command line interface. The module’s commands are defined in *di.xml* file which is in module’s *etc* folder. The content of the *di.xml* file defining a command which calls the method called *execute* is implemented in Test class, the *Test.php* file of which is in Command folder of BcIntegration module in Solteq vendor; this is illustrated in the following code snippet:

```
| <?xml version="1.0"?>
| <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectMan-
ager/etc/config.xsd">
|   <type name="Magento\Framework\Console\CommandList">
|     <arguments>
|       <argument name="commands" xsi:type="array">
|         <item name="solteq_bcintegration_command_test"
xsi:type="object">Solteq\BcIntegration\Console\Test</item>
|       </argument>
|     </arguments>
|   </type>
| </config>
```

2.6 Sources and Stocks

A source is a physical location of products from where products are shipped to customers. The source can represent a warehouse, drop shipper, distribution center, or a physical store selling products face to face. Each source has specific attributes, such as location, and defines total and salable quantity for products. In Magento 2, every online store must have at least one (default) source. (Managing Sources 2019)

Each source can be assigned to stock and each stock must have at least one assigned source. A stock represents a virtual, aggregated inventory of products for sources of

sales channels. The sales channels can be different websites running under a single Magento 2 instance. Depending on an online store's configuration, a single stock can be assigned to multiple sales channels. (Managing Stock 2019)

An example of sources, stocks and sales channels mapping for a specific product is illustrated in Figure 6.

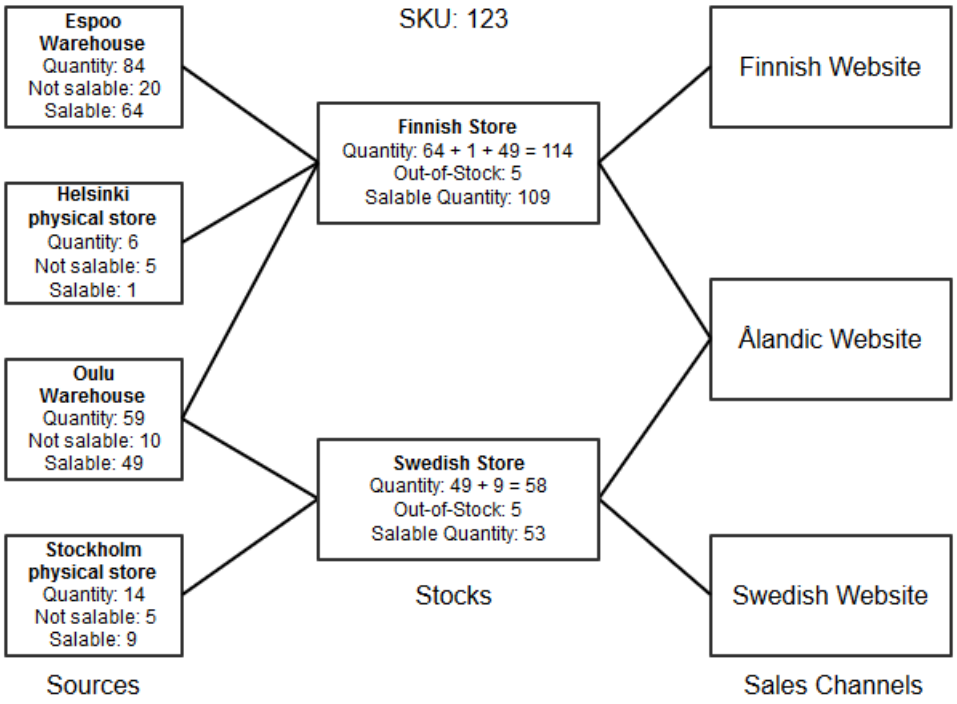

Figure 6. Sources, stocks and sales channels mapping

3 Microsoft Dynamics 365 Business Central overview

Many companies use a centralized system that collects various data, gives a global real-time view of data, automates business operations, improves the productivity of employees, and breaks barriers between different business units. Systems providing these functionalities are called Enterprise Resource Planning Systems, or ERPs. One of the ERP systems available on the market is Microsoft Dynamics 365 Business Central.

Microsoft Dynamics 365 Business Central is a business management solution for small and mid-sized organizations that automates and streamlines business processes and helps to manage businesses. Highly adaptable and rich with features, Business Central enables companies to manage their business, including for example, finance, manufacturing, sales, shipping, project management, services. Companies can easily add functionality relevant to the region of operation, and that is customized to support even highly specialized industries. (Welcome to Dynamics 365 Business Central 2020)

Microsoft Dynamics 365 BC provides various ways allowing its integration into other systems, such as Magento 2 online store. The ERP system provides endpoints that allow other systems to request or send data. Besides simple requesting data, the ERP system has implemented mechanisms that allow to filter and partially process requested data on the ERP system's side what enhances efficiency. An example of a total architecture of the system that contains the Microsoft Dynamics 365 Business Central ERP system and its integration in a Magento 2 online store is illustrated in Figure 7.

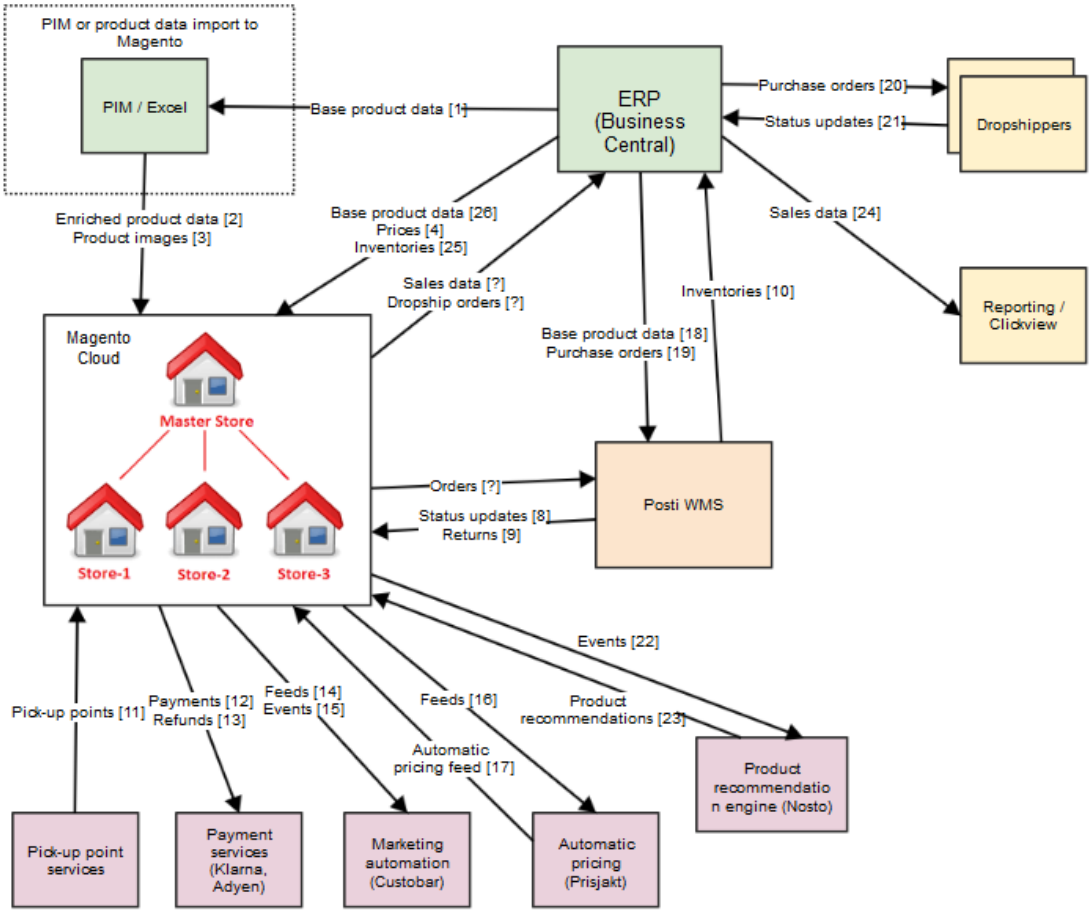

Figure 7. Total platform architecture including the integration of ERP system in Magento 2 online store

4 Integration implementation in general

4.1 Architecture

In order to avoid code redundancy and achieve higher flexibility and scalability, the logic of the Microsoft Dynamics 365 Business Central integration in Magento 2 is split into multiple logical groups called modules. In the future, the functionalities of these modules can be easily extended by adding new modules that can reuse the functionality implemented in these modules. The dependencies between the implemented modules and possible new modules are illustrated in Figure 8.

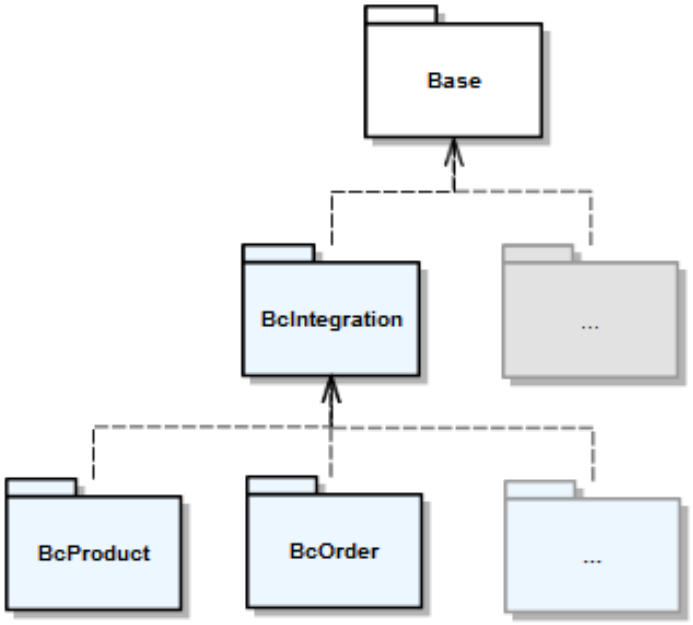

Figure 8. Dependencies between modules in the Solteq vendor folder

In a Magento installation the modules are in the vendor folder *Solteq* which is in *app/code* directory together with other custom modules or modules provided by third-party vendors not installed using Composer. The module placement can be seen in Figure 9.

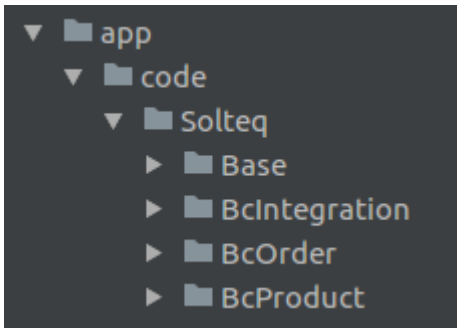

Figure 9. Module placement

As can be seen in Figure 8 and Figure 9, the integration is implemented in the following four modules:

- Base
- BcIntegration
- BcProduct

- BcOrder

Base module

In the Base module, the most used functionalities needed in almost every module are implemented. All other modules in the *Solteq* vendor folder are dependent on this module by reusing its functionalities. The functionalities contain error logging, accessing and modifying configuration values in Magento's database, base structure of configuration page, and predefined ACL rules.

BcIntegration module

The BcIntegration module has the same function as the Base module but on the level of MS Dynamics 365 BC integration in Magento 2. The module extends the functionalities implemented in the Base module by functionalities specific for the integration. It contains communication with the external system, common configuration values, configuration page substructure, and integration's general ACL rules.

BcProduct module

Integrations related to products are implemented in the BcProduct module. The module processes product data retrieved from the external system using the functionalities implemented in the BcIntegration module.

BcOrder module

Order integration is implemented in the BcOrder module. The same as in the BcProduct module, the BcOrder module reuses the functionalities implemented in the BcIntegration module. The module is used to send orders created in Magento 2 online store to an external ERP system.

Other modules

The modularity makes Magento 2 a very scalable platform; therefore, the implemented functionalities can be easily extended and supplemented by implementing other modules that can reuse the already implemented functionalities.

The architecture allows installing the BcProduct and the BcOrder modules independently on each other following the needs of a specific merchant. However, neither the BcProduct nor the BcOrder can be installed without the Base and the BcIntegration modules.

4.2 Base configuration

Since each Magento 2 installation is unique, most of the modules need to be configured. The module configuration of the implemented modules is done from the admin panel. The Base module adds a new tab with the title “SOLTEQ” to the general store configuration. The tab is used to keep all configuration sections added by modules in the sorted Solteq vendor and accessible for store administrators from one place. The Solteq tab is pictured in Figure 10.

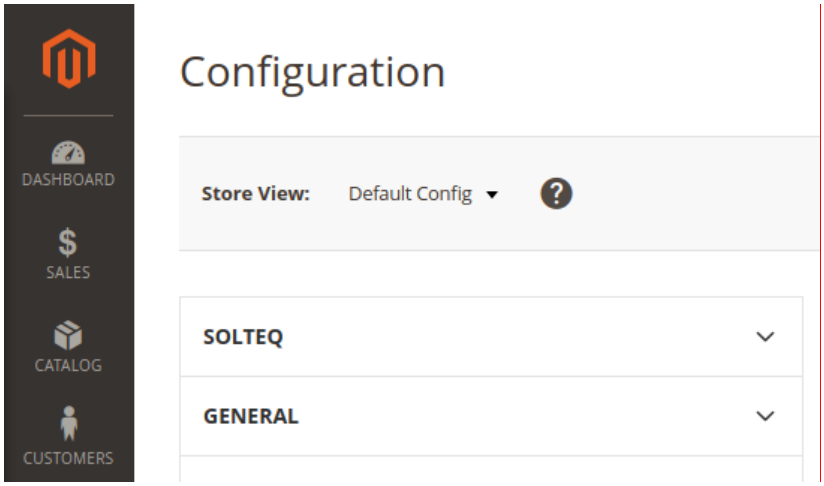

Figure 10. Solteq tab on the configuration page

Usually, online stores have multiple admin users with different responsibilities. Not every store administrator should have full access to the whole administration panel; thus, the implemented modules define their own ACL rules. In order to keep the ACL rules sorted, the Base module defines an ACL rule Solteq, which is used as a parent ACL rule for all rules defined by modules in Solteq vendor. The parent ACL rule allows to grant or to deny access to the configuration of all modules in Solteq vendor. As a child ACL rule of the main rule with title “Solteq”, there is a rule titled “BC Integration

Admin" used as a parent rule of all Integration modules. The structure of the ACL rules is pictured in Figure 11.

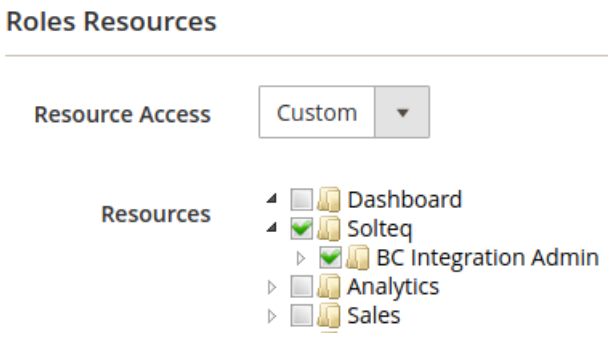

Figure 11. ACL rule structure

5 Communication with an external system

5.1 Implementation overview

Communication with an external ERP system, Microsoft Dynamics 365 Business Central, is implemented in a custom module named BcIntegration. The communication is achieved by calling an external ERP system's built-in application programming interface (API), so the integration does not require any development work outside Magento 2 modules, which is the strongest side of the implemented solution.

Setting up communication requires information about the ERP system. In Table 1, there is an overview of the module's configuration values. All the values are configured for the global scope; therefore, the same value is used for the whole Magento 2 installation regardless from store.

Table 1. BcIntegration module configuration values

Name	Path
Business Central ID	solteq_bc_integration/connection/bc_id
URL Base Part	solteq_bc_integration/connection/url_base_part
Username	solteq_bc_integration/connection/username
Password	solteq_bc_integration/connection/password

Company ID	solteq_bc_integration/connection/company_id

The information about the external system can be filled by an admin user with applicable access rights from the admin panel. The module adds a configuration section with the title “BC Integration” under “SOLTEQ” tab, which is added by The Base module. As can be seen in Figure 12, the section has one group named “Connection”.

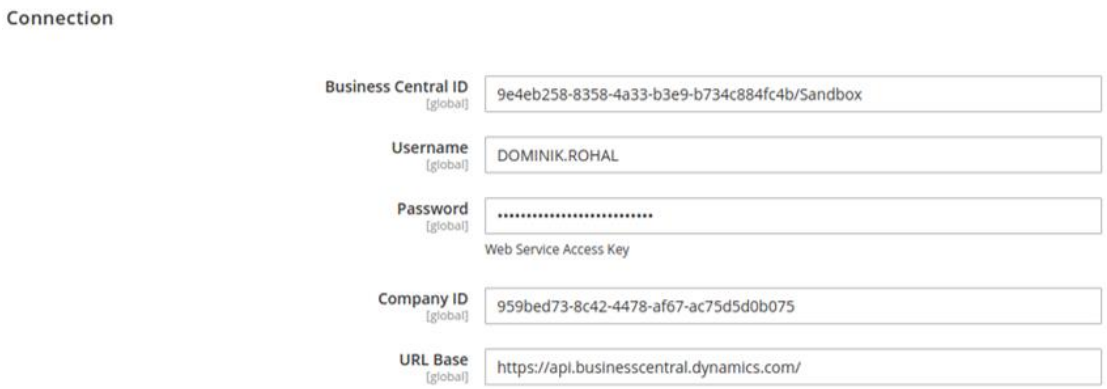

Figure 12. BC Integration section on the configuration page

Username, Password, and Company ID values can be found on the merchant’s Microsoft Dynamics 365 Business Central administration page. URL Base and Business Central ID, also known as a tenant ID, are parts of the administration website URL, where URL Base is followed by Business Central ID.

The structure of the BcIntegration module is illustrated in Figure 13. Classes Connection and Config are implemented in the BcIntegration module. The SolteqConfig class is implemented in the Base module and class Curl is provided by Magento 2 platform.

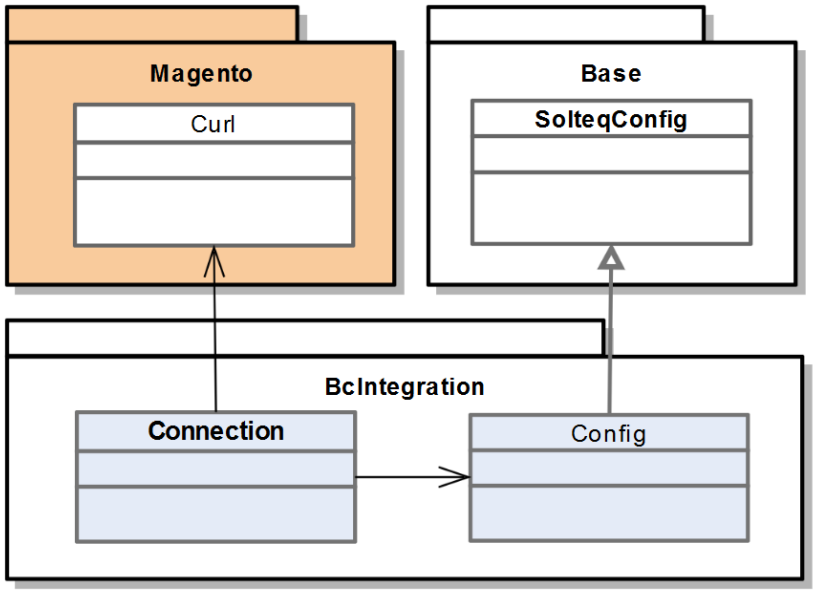

Figure 13. Relations between BcIntegration module's classes

The SolteqConfig class is used to access configuration values by configuration value path and scope is extended by the Config class. The Config class simplifies access to the configuration values found in Table 1. Each value has its own get method (getter) implemented in the Config class, which handles corresponding configuration value path and scope.

The communication with the ERP system is implemented in the Connection class. The Configuration class uses the Config class to access values required for setting up a connection with the external system. The methods implemented in the class combine the configuration values with the values parsed as method parameters and create a hypertext transfer protocol (HTTP) request which returns JavaScript Object Notation (JSON) as a response. The request to the external system is called using the Curl class provided by the Magento 2 platform. The calling of a request requires authentication and URL with parameters. The authentication is handled by the module and consists of a username and password taken from the module's configuration.

5.2 Calling standard API

One of the ways how to access the data stored in the ERP system MS Dynamics 365 Business Central from other systems is by calling a standard API. The API is called by sending a GET HTTP request to URL with the following pattern:

```
| <URL Base>/v1.0/<Business Central ID>/api/beta/companies(<Company ID>)<path>
```

The *<URL Base>*, *<Business Central ID>* and *<Company ID>* are values taken from the module's configuration. The *<path>* is a value that comes as a method's parameter. The method implementing the logic of calling the API returns the response as a text (string).

The same path is used to send a POST HTTP request to the standard API. Sending a POST HTTP request allows both-way communication with the external system. The data from Magento 2 are sent in the body of a POST HTTP request, and the API returns a text (string) as the response.

5.3 Calling Open Data Protocol web services

The data stored in MS Dynamics 365 Business Central can also be retrieved by calling Open Data Protocol (OData) web services. The data can be retrieved by calling a GET request to URL with the following format:

```
| <URL Base>/v1.0/<Business Central ID>/ODataV4/Company("<Company Name>")<path>
```

The *<URL Base>* and *<Business Central ID>* are values taken from the module's configuration. The *<Company Name>* value is retrieved by calling the standard API GET HTTP request with "/?$select=name" as the path which is appended to the request's URL. The response of the standard API call is a JSON with the *<Company Name>* value stored under the key "name". The *<path>* is a value coming as a method's parameter, and it is appended to the request's URL. The method implementing the logic of calling the OData web services returns the response as a text (string).

6 Product integration

6.1 Product integration overview

The product integration is implemented in a module named BcProduct which depends on the BcIntegration module. The BcProduct module uses the BcIntegration module to request product data from the external system and access configuration values.

All configuration fields specific for the product integration are in the “Product BC Integration” section under the “SOLTEQ” tab on the store configuration page. The section placement is pictured in Figure 14.

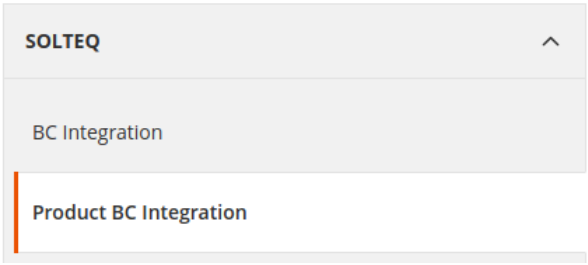

Figure 14. Product integration configuration section

There is a section with the module’s general configuration under the “Product BC Integration called “General” containing only one configuration field listed in Table 2 together with its configuration path.

Table 2. General product integration configuration fields

Name	Path
Enabled	solteq_bc_product/general/enabled

In Figure 15, there is the “General” group of the “Product BC Integration” section.

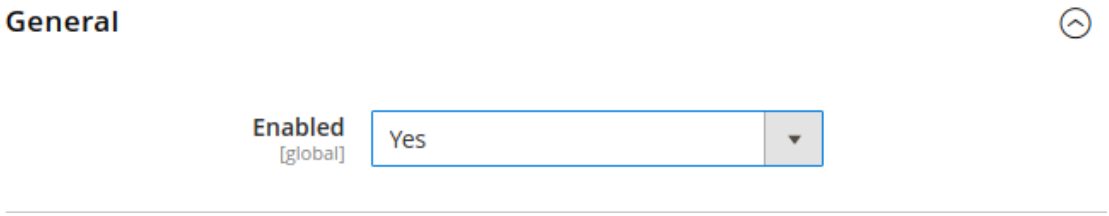

Figure 15. General configuration of Product Integration

The “General” group consists of one configuration field which is a selection with “Yes” and “No” values. The selection allows store administrators with applicable access rights to disable or enable all functionalities implemented in this module.

6.2 Product update in general

The product update is one of the main functionalities implemented in the BcProduct module. It is used to import products from an external system to a Magento 2 installation. The update is also used to update the information of existing products in the Magento 2 installation, based on their identifier stock keeping unit (SKU). In a Magento 2 installation, the product update is triggered by a cron job, in specified frequency, or by executing a command, from the Magento root directory, using a command-line interface (CLI). Both, the cron and the command, are implemented in the module. The frequency of triggering the product update by the cron can be configured from the store configuration page.

The configuration of the product update is in the “Product BC Integration” section under the “SOLTEQ” tab, in the store configuration. All configuration fields, that are used only by the product update, are in the group called “Product Update”. The configuration group on the store configuration page is pictured in Figure 16.

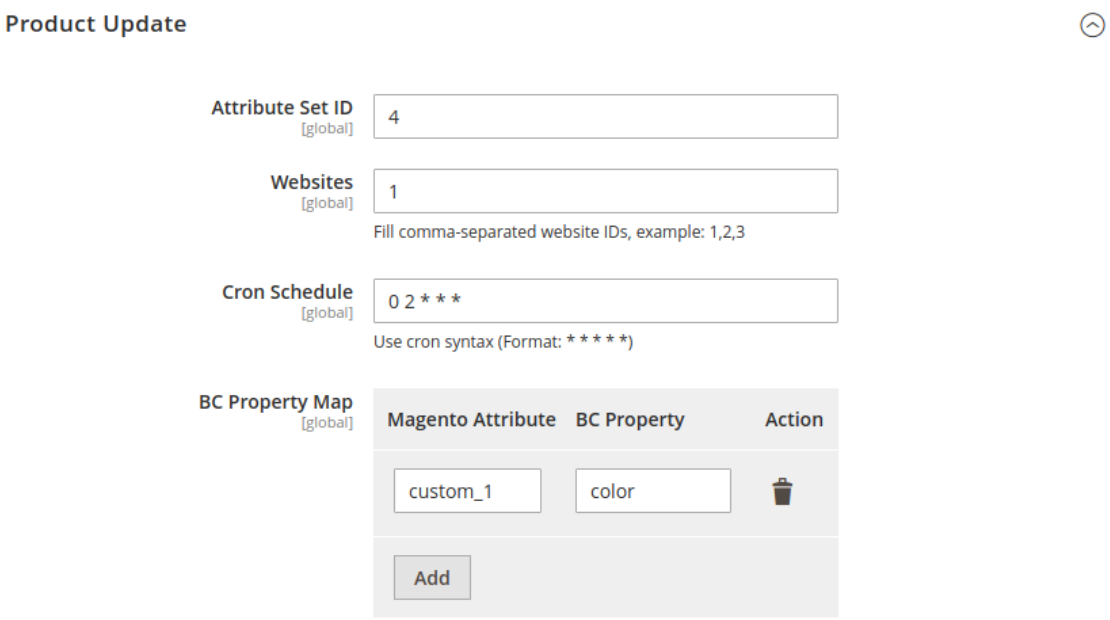

Figure 16. Product update configuration

In Table 3, there are listed configuration paths of product update's configuration values.

Table 3. Product update configuration fields

Name	Path
Attribute Set ID	solteq_bc_product/product_update/default_attribute_set_id
Websites	solteq_bc_product/product_update/schedule
Cron Schedule	solteq_bc_product/product_update/websites
BC Property Map	solteq_bc_product/product_update/configurable_attributes

Attribute Set ID

The Attribute Set ID is a numeric identifier of the attribute set which is assigned to the product. The attribute set defines which attributes are saved for the product. It can be considered as a template for the product record.

Websites

The field contains comma-separated numeric values. The values are identifiers of websites in which the product should be shown.

Cron Schedule

This field defines how often the product update is triggered by the cron job. The field accepts text value in a valid cron syntax.

BC Property Map

This field allows admin users to map values that are returned from the external system. Admin users can map attributes that are not saved in Magento's database by default into custom product attributes. A new row can be added by clicking on the "Add" button. The row contains two fields, "Magento Attribute" and "BC Property". The value filled in the "Magento Attribute" field is the code of Magento's EAV attribute which stores a value retrieved from the product data returned from the external system, under key defined in the "BC Property" field.

Besides the custom configuration fields, on the store configuration page, there is also an additional field on the admin category page, added by the module. The category page is accessible from the admin panel by clicking on the item with title "Categories" under the "CATALOG" tab located in the admin panel's left menu. The category configuration page contains a list of store categories. Each category has its own additional configuration. The configuration contains a custom field called "BC Category Codes", added by the BcProduct module. The field is hidden in a wrapper with the title "Solteq Product Integration" which can be seen in Figure 17.

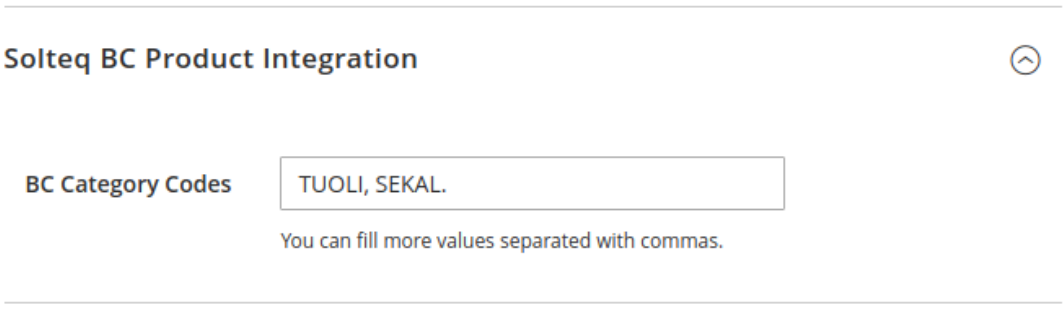

Figure 17. Category mapping configuration

In the custom field, there are filled category codes from the external system. Multiple categories from the external system can be mapped into a single category in Magento 2 installation by filling multiple comma-separated values into the field. All

products which come from the external system and have category code that is filled in the field are assigned to the category. The module also allows assigning multiple categories to a single product by filling the same category code to multiple Magento's categories. The value is stored in Magento's database as a category's custom attribute which is defined by the module.

6.3 Product update implementation

The full product update is done by calling the standard API with the following expression as the method's parameter:

```
| /items?$select=<product_attributes>
```

The *<product_attributes>* is a comma-separated list of product attributes that are saved in the Magento 2 online store's database. The part of the product attribute list is static; hence, it cannot be changed by an admin user. However, admin users can specify additional attributes.

As a response, there is returned a JSON that contains the product data. The response consists of multiple records where each record represents one product. For each product, there are returned only attributes that are specified in *<product_attributes>*.

By default, the following product attributes are saved in Magento's database:

- Name
- SKU
- Price
- Type Id
- Attribute Set Id
- Status
- Website Ids
- Visibility

Name

It is one of the required product's attributes. The name is taken from the external system. It is retrieved from the returned product data where is persistent under the key "displayName".

SKU

Each product can be identified by its unique SKU, often known as a scannable bar code that can be printed on product labels in a retail store. From the external system, the SKU comes under the key “number”.

Price

There are more values that come from the external system and can be used for the product price calculation. In the module, the value with the key “unitPrice” is used for product price calculation.

Type Id

In the basic version of the module, all products imported from the external system are simple products; hence, their type id is “simple”.

Attribute Set Id

The attribute set ID is taken from the product integration configuration. The configuration value with path “solteq_bc_product/product_update/default_attribute_set_id” is used for all products which are imported from the external system.

Status

The product status determines whether the product is enabled or disabled in the store. Newly created products have status 0 which means that they are disabled. The products can be enabled during other processes that are running in the Magento installation.

Website Ids

The value is taken from the product integration configuration. The product is assigned to the websites whose Id is filled in the configuration field with path “solteq_bc_product/product_update/schedule”.

Visibility

As the visibility, there can be set any numeric value from 1 to 4.

Table 4. Product visibility values

Value	Meaning
1	The product is not visible.
2	The product is visible in the catalog.
3	The product is visible in search.
4	The product is visible in both, catalog and search

Newly created products are not visible straight away, after the product import. The product visibility can be changed by other processes that are running in the Magento installation.

There are more values returned from the external system which can be mapped to custom product attributes. The map can be configured from product integration configuration. The configuration path of the map is “solteq_bc_product/product_update/configurable_attributes”.

Besides saving values to product attributes, the product update also maps products into categories. Besides other values, the product data returned from the external system during the update contains the product’s category code. The product is assigned to all categories which have the category code filled in the custom field “BC Category Codes” which is pictured in Figure 17.

The logic of the product update is split into multiple classes. The relations between the main classes which contain the logic of the product update are illustrated in Figure 18.

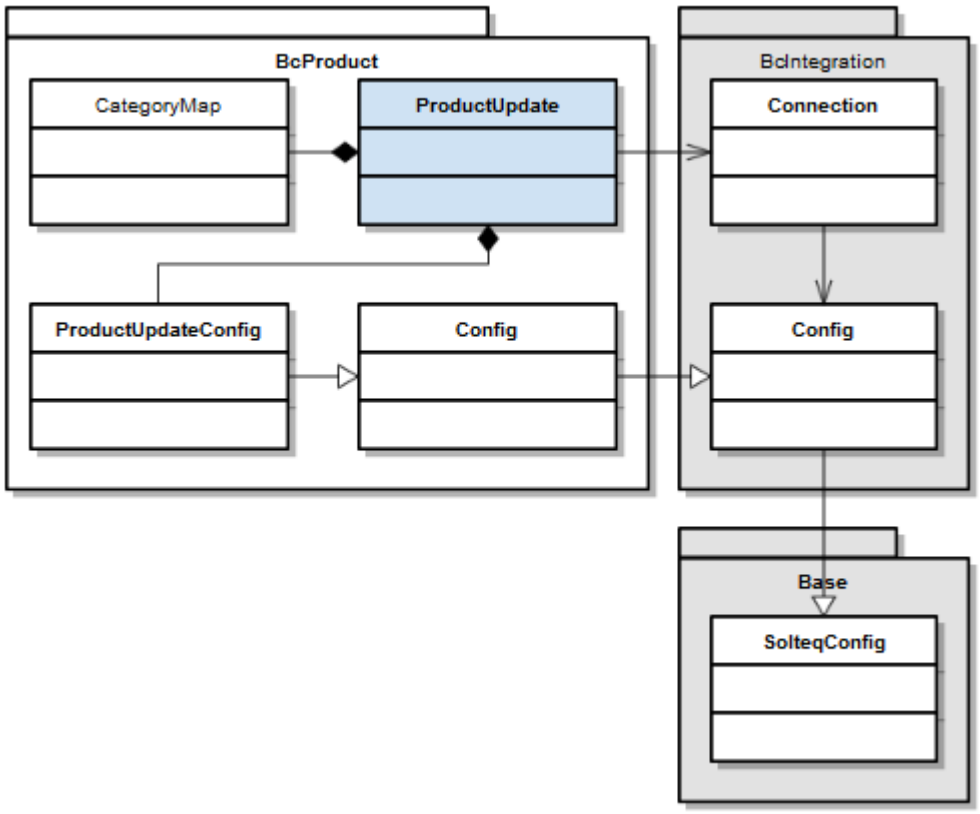

Figure 18. Relations between classes which provide product update

ProductUpdate class

Most of the module logic is implemented in the ProductUpdate class. The class requests the product data by calling methods implemented in the Connection class. After receiving the product data, they are processed and saved in a Magento's database.

CategoryMap class

The CategoryMap class contains a logic that goes through all categories in a Magento installation and builds a category map. The category map can be presented as an associative array, where the key is a category code, and the value is an array of category Ids that are assigned to products with the category code. Once the map is built during a product update, it is kept and reused next time it is needed during the product update.

ProductUpdateConfig class

The ProductUpdateConfig class contains the logic of accessing product update configuration values. Once, a value is retrieved from the database, it is kept and served next time it is needed, during the product update.

6.4 Inventory update in general

In most cases, the update of inventories needs to be run in shorter intervals than full product update since Magento 2 online store may be only one of many channels that changes product availability. The sellable quantities of products can be affected by other systems connected to the MS Dynamics 365 BC ERP system and it may lead to situations when a specific product gets sold out, but this information is not in Magento 2 online store yet. The probability of this conflict can be lowered by defining higher buffer in the inventory map in the module's configuration. However, defining a greater number in the buffer configuration field is not efficient if the interval of the inventory update is too long. Because of this reason, the inventory update is separated from the product update. The update is scheduled by cron job; however, it can be triggered by running a CLI command at any time.

The configuration of the inventory update is in the "Product BC Integration" section under the "SOLTEQ" tab, in the store configuration. All configuration fields, that are specific for inventory update, are in the group called "Inventory Update". The configuration group is pictured in Figure 19.

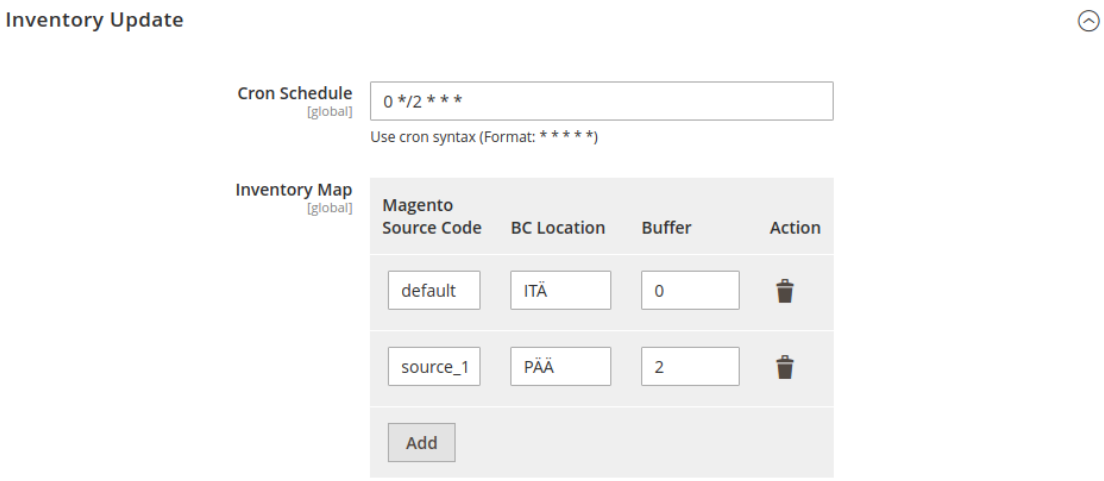

Figure 19. Inventory update configuration

The configuration fields and their paths are listed in Table 5.

Table 5. Inventory update configuration fields

Name	Path
Cron Schedule	solteq_bc_product/inventory_update/schedule
Inventory Map	solteq_bc_product/inventory_update/inventory_map

Cron Schedule

This field defines how often the inventory update is triggered by a cron job. The field accepts text value in a valid cron syntax.

Inventory Map

The inventory map is used to map locations in the external system into sources in Magento. A new row can be added to the inventory map by clicking on the “Add” button. A row consists of three fields. The first field is a code of Magento’s source. The second field is a location name in the external system. The last field is a buffer that accepts any numeric value. During an inventory update execution, the buffer value is subtracted from the product quantity per the location retrieved from the external system and the result is saved as sellable quantity in the Magento 2 store. The

buffer can be used for several purposes. One of them is avoiding situations when the product is sold out from another system, but the availability has not been updated yet.

6.5 Inventory update implementation

The inventory update is executed in two steps. In the first steps, there is called BcIntegration's method calling the external system's OData web service endpoint with parameter matching the following pattern:

```
| /ItemLedgerEntries?$select=Entry_No,Item_No,Location_Code
&$filter=(<location_filter>) and Entry_No gt <last_ledger_en-
try_no>
```

The "/ItemLedgerEntries" as a parameter of the BcIntegration's method returns data containing changes in the product's availability per location. Without adding anything to the parameter, there is a huge amount of data returned from the external system and most of the data is not useful for the integration. To lower the amount of returned data the "?$select=Entry_No,Item_No,Location_Code" part is added to the parameter. The part specifies attributes that are useful for the integration and only these attributes are returned in the response from the external system. Since in the external system, there may be locations that are not mapped to any online store's sources, the next part of the method's parameter is a location filter (<location_filter>). The location filter specifies locations that are mapped into an online store's sources. As a result, only changes in the product's availability of these locations are returned from the external system. The location filter has the following format:

```
| Location_Code eq "<location_code>"
```

Multiple location filters can be defined by adding "OR" between them. At this point, the result contains the whole history of changes in the product's availability per location. Adding "Entity_No gt <last_ledger_entry_no>" limits the result to the latest changes. The <last_entry_no> is the Entry_No value of the last record returned from the external system. Only records with Entry_No greater than <last_ledger_entry_no> are returned from the external system.

As a response, there is returned data in a JSON format that contains multiple records. Each record consists of the following values which are also defined in the parameter:

- **Entry_No**
- **Item_No**
- **Location_Code**

Entry_No

It contains a numeric value which is used as a unique identifier of a ledger record. A newer record always has greater Entry_No value than older records. This characteristic is used to retrieve only records that are older than the oldest record from the last request. The Entry_No value of the last retrieved ledger record is used in the call parameter as <last_ledger_entry_no> value.

Item_No

This value matches the SKU of an existing product in the Magento online store. The availability of the product with the SKU needs to be updated.

Location_Code

The Location_Code has an external system's location code which is mapped into the Magento online store's source. The value specifies in which source the product's availability needs to be updated.

The response is processed and used to build a structure that is used in the second step of the inventory update. The structure is an associative array of arrays where the key is a location code, and the value is an array of products that need to be updated in the online store per the location (source).

For each location, there is called the external system's OData endpoint with parameter matching the following pattern:

```
| /ItemCard?$filer=Location_Filter eq <location_code> &$select=No,Inventory &$filter=(<product_filter>)
```

The “/ItemCard” as a parameter returns information about all products in the external system. Appending “?$filer=Location_Filter eq <location_code>” specifies location whose availability is returned in the response. The <location_code> is a unique code of a specific location in the external system that is mapped into a source in the Magento online store. The “&$select=No,Inventory” part limits the result only to the values that are needed for the online store. The <product_filter> defines products whose availability needs to be updated in the online store. The product filter has the following format:

```
| No eq "<product_sku>"
```

The <product_sku> is an SKU of the Magento online store’s product whose inventory needs to be updated. There can be multiple product filters separated with “OR” in the parameter.

As a response, there is returned a text value in JSON format containing multiple records where each record has the following values:

- **No**
- **Inventory**

No

The No is a unique value that identifies a specific product. In the Magento online store, the value matches a product’s SKU.

Inventory

The Inventory is a numeric value that is saved in the Magento online store’s database as a product availability in a specific source (location).

The logic of the inventory update is split into multiple classes. The relations between the classes is pictured in Figure 20.

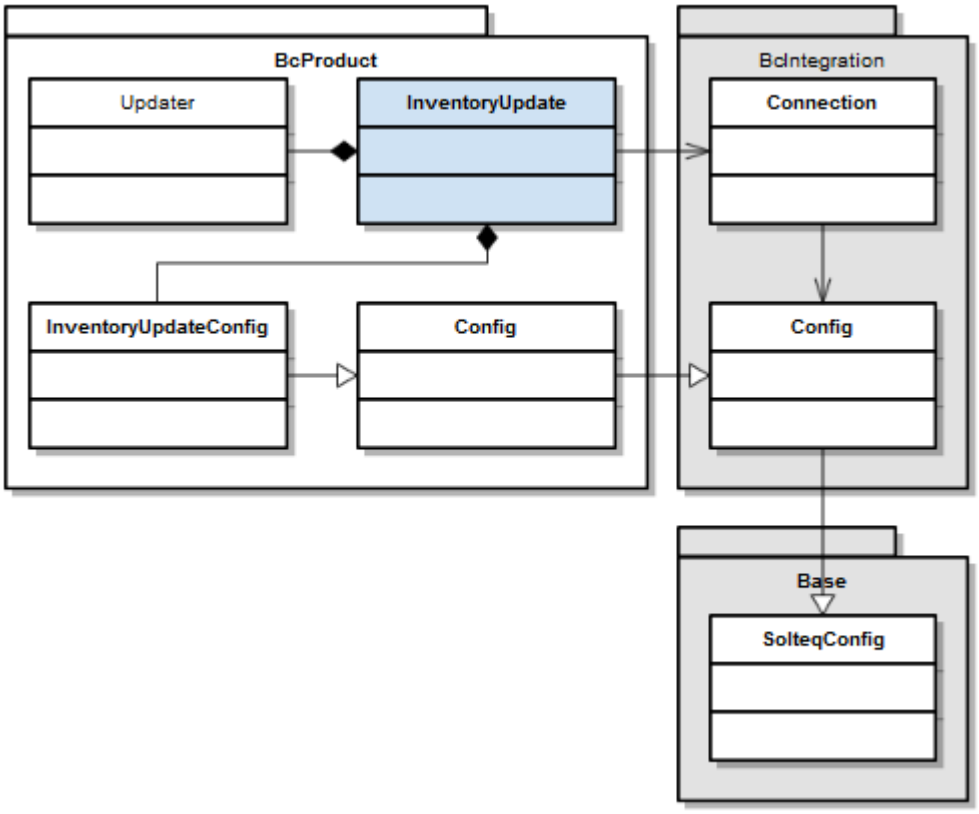

Figure 20. Relations between classes which provide inventory update

InventoryUpdate class

The main class of the Inventory update is called InventoryUpdate. The class uses the Connection class implemented in the BcIntegration module to request inventory data from the external system. The retrieved data are processed and saved in Magento's database.

Updater class

The Updater class is used by the InventoryUpdate class to format and save data for specific inventory.

InventoryUpdateConfig class

The InventoryUpdateConfig class extends the Config which is also extended by the ProductUpdateConfig which provides access to product update configuration values. The InventoryUpdateConfig class provides access to inventory update configuration

values and it allows changing configuration values during the inventory update. Besides accessing configuration values, the module builds a location map when it is required and once the map is built, it keeps it during the whole inventory update execution and reuses the next time it is needed. The location map provides quick access to the code of Magento's source and inventory buffer value by the external system's location name.

7 Order integration

7.1 Order integration overview

Orders created in Magento 2 online store are sent to the centralized external system. After orders are saved in the external system, they are accessible by other systems that are connected to the same external system. The centralization allows orders to be processed by other systems that are not directly connected with the Magento 2 online store, and it also allows to keep all orders, created in different sales channels, in one place; hence, merchants can have a global view of their sales.

The order integration configuration is in an additional configuration section named "Order BC Integration", in the "SOLTEQ" tab. The new section contains two configuration groups with titles "General" and "Order Send". The configuration of the order integration is pictured in Figure 21.

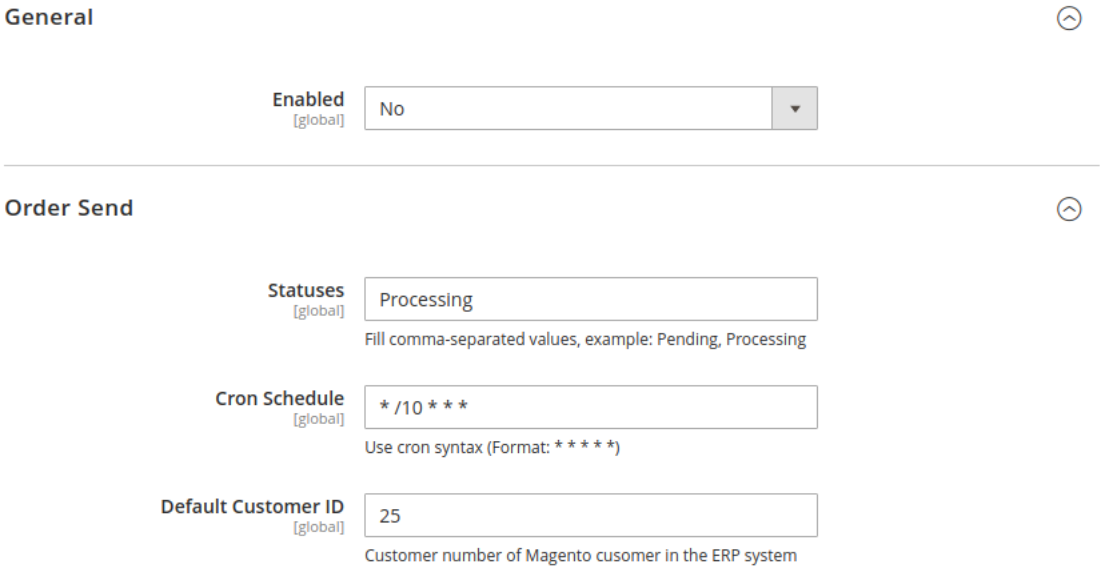

Figure 21. Order integration configuration

The configuration fields with their paths are listed in Table 6.

Table 6. Order integration configuration fields

Name	Path
Enabled	solteq_bc_order/general/enabled
Statuses	solteq_bc_ordert/order_send/statuses
Cron Schedule	solteq_bc_ordert/order_send/schedule
Default Customer ID	solteq_bc_ordert/order_send/customer_id

Enabled

The order integration can be disabled from the admin panel. If the inventory update is triggered, the value of this field is checked. If the inventory update is disabled, the execution is interrupted. As a default, the order integration is disabled and must be enabled in the admin panel after the module is installed.

Statuses

By default, a newly created order has status “Pending”. A successful payment changes the status to “Processing”, or it can be changed to other, also custom, state

by a third-party module. For this reason, there is a field in the configuration which allows admin users to specify order statuses. Orders with specified status are sent to the external system. The default value of the field is “Processing”.

Cron Schedule

The Cron Schedule configuration field allows admin users to specify how often the orders created in the online store should be sent to the external system. The field accepts values matching a valid cron syntax.

Default Customer ID

It is a customer number of a generic customer in the ERP system to who are assigned all orders created in a Magento 2 online store.

7.2 Sending orders to external system implementation

Orders created in a Magento 2 online store are sent to an external system by calling BcIntegration’s method sending a POST request to the external system’s standard API with the following parameter:

```
| /salesOrders?$select=id
```

The “/salesOrders” as the parameter returns information about a newly created order in the external system (the order is created in the external system based on the values sent from the Magento online store). Adding “?$select=id” lowers the amount of returned data to only one attribute. The value of the returned attribute is a unique identifier of the newly created order in the external system. The identifier is saved in a custom column which is added to Magento’s database table that contains order’s data. The column name is bc_external_id and can be used to check the order status in the external system and update it in the Magento online store. The functionality of updating order statuses is not part of this integration. However, the future custom Magento modules can use the bc_external_id value to add this functionality. The BcOrder module uses the value saved in the bc_external_id column to identify new orders that should be sent to the external system.

The order data is sent to the external system in the POST request body. The data are sent in a JSON format. The JSON consists of the following attributes:

- **orderDate**
- **customerNumber**
- **salesOrderLines**

orderDate

In this attribute, there is a date when the order was created in the Magento online store.

customerNumber

The customer number identifies the customer in the external system to who is the order assigned. In the base version of the module, this value is static, and it is defined in the module configuration. In the external system, all orders created in the Magento online store are assigned to the single generic customer with this identifier.

salesOrderLines

The value of this attribute is a JSON array. Each JSON in the array contains data of a single order item. The order item can be an ordered product or a fee, such as shipping costs. The product data contains a product identifier, a quantity of ordered product, product price, tax code, and item type. The item type attribute is used to recognize product items from fee items.

A Magento 2 online store does not keep information about product identifiers that are used in the external system. For this reason, the identifier is requested from the external ERP system by sending a GET HTTP request to the standard API with the following parameter:

```
| /items?$filter=number eq '<product_sku>'&$select = id
```

The <product_sku> is a product's SKU used also in Magento online store. The "/items" as the parameter returns data of all products in the external system. Appending "?$filter=number eq '<product_sku>'" limits the response to the data of product with attribute number matching the product SKU. The product data contains many attributes that are useless for order integration. Because of this reason, there

is appended “&$select = id” part to the parameter. This part specifies attributes that are returned in the response from the external system. In this case, it is just “id” which is the needed identifier. The returned identifier is kept during the whole order integration execution and it is reused if needed.

Order integration is implemented in classes that are pictured in Figure 22.

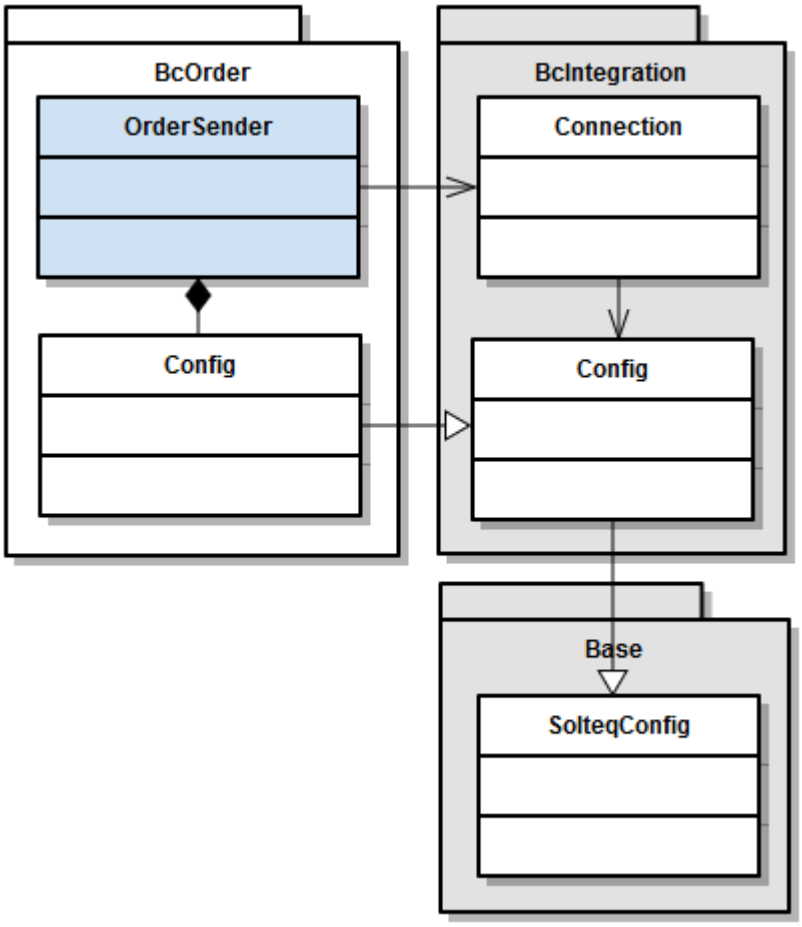

Figure 22. The relation between classes which provide order integration

OrderSender

The OrderSender class is the main class of the BcOrder module. It contains the main part of the logic allowing an Magento 2 online store to send orders to an external system. The logic of the class formats order data to a specific format and uses logic

implemented in the BcIntegration module to send the order data to an external system. A response from the external system is processed and needed information is saved in an online store's database.

Config

The Config class implemented in the BcOrder class extends the BcIntegration's Config class and adds functions specific for an order integration. The class provides an access to the configuration values needed for the order integration.

8 Conclusion

Online shopping enhances the convenience of shopping and allows customers to compare and buy different kinds of products from their homes. Customers can visit a website or open a mobile application and list the products they want to buy. However, it is only one part of the whole system. E-commerce brings convenience not only to the end customers, but also to merchants, logistic companies, manufacturers, and other businesses. Usually, in the chain between the point where the products are manufactured to the point they are delivered to the end customer, different systems are used automating many tasks and increasing effectiveness. Using different systems brings one main problem. Different systems need to communicate with each other. The main goal of this bachelor thesis was to investigate the possibility of connecting two different systems and to implement a universal reusable and scalable solution. The connected systems are ERP system Microsoft Dynamics 365 Business Central and e-commerce platform Magento 2. Usually, merchants do not have the capabilities to maintain and develop these two systems. Because of this reason, merchants find vendors who develop and maintain their systems. Commonly, an ERP system and online store do not have the same vendor. This used to be a problem since existing solutions used to integrate Microsoft Dynamics 365 Business Central into Magento 2 required development work on both sides; hence, close cooperation of multiple development teams was needed.

The solution implemented as the result of this thesis allows Magento 2 vendors to integrate an ERP system in Magento 2 online store by installing modules in a Magento

2 installation. The module uses the endpoints of the built-in ERP system to import product data, update product quantities, and send order information from an online store to an ERP system; hence, the implemented solution does not require any additional development work on the ERP system's side. The modules can be installed in any Magento 2 project. Thanks to the design of the modules, their functionality can be easily extended.

During the Magento module development, a new version of Magento 2 was released. Magento version 2.3.x introduced important changes that affected the implemented solution. The main change introduced in the new Magento version was a multi-source inventory (MSI) that allowed products to have multiple quantities, each for different sources representing physical warehouses mapped into stocks and sales channels. Support of MSI was not originally planned to be implemented as part of this bachelor's thesis; however, it was implemented as a part of this solution since the MSI became Magento's core feature. Since the MSI is a relatively new feature in Magento, the part of the implementation allowing source mapping is not supported by older versions of Magento.

In the future, the implementation can be extended following the needs of a specific customer. The design of the implemented modules allows to reuse their logic to add additional integrations. The future versions of the implemented modules can support multiple store mode that would also allow adding support of locations; thus, versions of an online store in different languages. Additionally, the support of mapping custom attributes created in the ERP system into custom Magento's product attributes can be added to the existing modules.

Most importantly, the implemented solution proved that Microsoft Dynamics 365 Business Central ERP system can be integrated into Magento 2 without complicated development work on the ERP system's side.

References

Admin Panel Placement and Design. 2019. Page on Magento documentaton website. Accessed on 23 January 2020. Retrieved from https://devdocs.magento.com/guides/v2.3/ext-best-practices/admin/placement-and-design.html.

Kristen L. 2019. *What Is Magento?* Page on Commonplaces website. Accessed on 18 November 2019. Retrieved from https://www.commonplaces.com/blog/what-is-magento/.

Managing Sources. 2019. Page on Magento documentation website. Accessed on 27 April 2020. Retrieved from https://docs.magento.com/m2/ce/user_guide/catalog/inventory-sources.html.

Managing Stock. 2019. Page on Magento documentation website. Accessed on 27 April 2020. Retrieved from https://docs.magento.com/m2/ce/user_guide/catalog/inventory-stock.html

Module overview. 2019. Page on Magento documentation website. Accessed on 20 December 2019. Retrieved from https://devdocs.magento.com/guides/v2.3/architecture/archi_perspectives/components/modules/mod_intro.html.

Scope. 2019. Page on Magento documentation website. Accessed on 26 January 2020. Retrieved from https://docs.magento.com/m2/ce/user_guide/configuration/scope.html.

Shaun U. N.d. *Magento 2 Commerce vs Magento 2 Open Source - which is best for who?* Page on Sozo website. Accessed on 24 January 2020. Retrieved from https://sozodesign.co.uk/blog/ecommerce-websites/magento-2-commerce-vs-magento-2-open-source/.

Single Store Mode. 2019. Page on Magento documentation website. Accessed on 26 January 2020. Retrieved from https://docs.magento.com/m2/ce/user_guide/stores/store-mode-single.html.

Websites, Stores, and Views. 2019. Page on Magento documentation website. Accessed on 23 January 2020. Retrieved from https://docs.magento.com/m2/ce/user_guide/stores/websites-stores-views.html.

Welcome to Dynamics 365 Business Central. 2020. Page on Microsoft Docs website. Accessed on 08 February 2020. Retrieved from https://docs.microsoft.com/en-us/dynamics365/business-central/index.