



**jamk.fi**

# **Magento Module Development**

Richárd Gergő Madarász

Bachelor's thesis

May 2020

Programme of Information and Communication Technology

Jyväskylän ammattikorkeakoulu

JAMK University of Applied Sciences

Author(s) Madarász, Richárd Gergő	Type of publication Bachelor's thesis	Date May 2020 Language of publication: English
	Number of pages 57	Permission for web publication: x
Title of publication <b>Magento Module Development</b>		
Degree programme Programme of Information and communication Technology		
Supervisor(s) Manninen, Pasi & Salmikangas, Esa		
Assigned by Solteq Oyj		
Abstract  <p>In the company there was a need for a module which makes it possible to edit webpage translations directly, since the only way to make changes was to edit the translation files directly, which required a redeploy of the environment every single time.</p> <p>The task was to create a module which works as a panel in the configuration menu of the admin panel. This way the customer can make changes themselves without the need of creating support tickets and having the company make changes and deploys.</p> <p>The implementation is done using Magento 2 Platform. The module directory mainly consists of blocks, controllers models and configuration files. It is written in HTML, PHP languages and XML in the configuration files, while also using SQL for database management and Composer as module manager.</p> <p>The study resulted in a working custom module, that has the ability to edit localization files directly from the admin menu, with the added feature of saving changes to the project's database. The module is ready to be implemented in any Magento 2 Project.</p> <p>The module is ready to be used, albeit it is not featured in any of the company's project as of May 2020, yet is has been considered for many various old and new projects.</p>		
Keywords/tags ( <a href="#">subjects</a> ) eCommerce, HTML, JavaScript, Magento, PHP, SQL, Webstore		
Miscellaneous		

## Contents

<b>1</b>	<b>Introduction .....</b>	<b>9</b>
1.1	Project Goal .....	9
1.2	Background of the author and Solteq .....	9
<b>2</b>	<b>E-commerce .....</b>	<b>10</b>
2.1	Introduction to E-commerce .....	10
2.2	Ecommerce categorization.....	11
2.2.1	Business-to-Customer (B2C) .....	11
2.2.2	Business-to-Business (B2B).....	11
2.2.3	Consumer-to-Consumer (C2C) .....	11
2.2.4	Consumer-to-Business (C2B) .....	11
2.2.5	Business-to-Administration (B2A) .....	12
2.2.6	Consumer-to-Administration (C2A).....	12
2.3	Ecommerce platforms .....	12
<b>3</b>	<b>Magento eCommerce .....</b>	<b>15</b>
3.1	Overview.....	15
3.2	Magento Community Edition .....	16
3.3	Magento Commerce Edition .....	16
3.4	Key Differences of Community and Commerce Edition.....	17
<b>4</b>	<b>Magento Environment.....</b>	<b>18</b>
4.1	Used Languages.....	18
4.1.1	PHP.....	18
4.1.2	HTML.....	19
4.1.3	JavaScript .....	20
4.2	Magento Environment Prerequisites .....	21
4.2.1	Web Server .....	21

4.2.2	Composer.....	21
4.2.3	Database .....	22
4.2.4	PHP.....	22
4.2.5	Authentication keys.....	22
4.3	Magento Environment Install.....	22
4.3.1	New Project .....	22
4.3.2	Cloning Existing Projects from Git .....	25
4.4	File Structure .....	27
<b>5</b>	<b>Magento Modules .....</b>	<b>28</b>
5.1	Overview.....	28
5.2	Module Relations .....	28
5.2.1	Overview.....	28
5.2.2	A module uses B, C customizes B .....	29
5.2.3	A module reacts to B, C customizes B .....	29
5.2.4	A module and C customize B .....	29
5.2.5	A module replaces B .....	30
5.3	Module Dependencies .....	30
5.3.1	Overview.....	30
5.3.2	Hard dependency .....	31
5.3.3	Soft dependency.....	31
5.4	Module Areas .....	32
5.5	Module Management.....	33
5.5.1	Composer Install .....	33
5.5.2	Manual Install .....	34
5.5.3	Managing modules .....	34
5.6	Module structure.....	36

<b>6</b>	<b>Translation Module .....</b>	<b>38</b>
6.1	Overview.....	38
6.2	Initial Configuration.....	39
6.3	Frontend View .....	41
6.4	Background Logic.....	42
	6.4.1 Controller.....	42
	6.4.2 Loading files.....	43
	6.4.3 Editing translations.....	45
	6.4.4 Database connection.....	46
	6.4.5 Saving and Loading to database.....	48
<b>7</b>	<b>Conclusion.....</b>	<b>49</b>
	<b>References.....</b>	<b>50</b>

## Figures

Figure 1.	Ecommerce sales trend .....	10
Figure 2.	Market share of Ecommerce platforms .....	12
Figure 3.	The pricing of Bigcommerce with the category differences showcased .....	14
Figure 4.	Magento Admin panel.....	15
Figure 5.	Example of embedded PHP code in HTML.....	19
Figure 6.	HTML code and how it's rendered .....	20
Figure 7.	Basic JavaScript functionality .....	21
Figure 8.	Magento composer installation .....	22
Figure 9.	Magento version specific installation .....	23
Figure 10.	Magento setup with command line.....	23
Figure 11.	Magento web setup wizard.....	24
Figure 12.	Magento homepage on a fresh install .....	24
Figure 13.	Magento project env.php configuration.....	25
Figure 14.	Magento project WordPress configuration .....	25

Figure 15. Commands to create a DB dump and import .....	26
Figure 16. Magento local database fix .....	26
Figure 17. Magento Admin user creation .....	27
Figure 18. A uses B, C customizes B .....	29
Figure 19. A reacts to B, C customizes B .....	29
Figure 20. A and C customize B .....	30
Figure 21. A replaces B .....	30
Figure 22. Hard dependency example .....	31
Figure 23. Soft dependency example .....	32
Figure 24. Module installation with Composer .....	33
Figure 25. Authentication during module installation .....	33
Figure 26. composer.json example .....	34
Figure 27. composer.lock example of a specific module .....	35
Figure 28. Change module state .....	35
Figure 29. config.php declaring module states .....	36
Figure 30. Translation module in admin panel .....	38
Figure 31. module.xml of translation module .....	39
Figure 32. registration.php of translation module .....	39
Figure 33. menu.xml of translation module .....	39
Figure 34. Effects of menu.xml of translation module .....	40
Figure 35. routes.xml of translation module .....	40
Figure 36. Structure of view directory that contains all display logic .....	41
Figure 37. layout .xml file of translation module .....	41
Figure 38. Controller of translation module .....	42
Figure 39. Controller checking POST parameters to call functions from Block .....	43
Figure 40. findLanguageFiles() function .....	43
Figure 41. Dropdown menu code to display and select files .....	44
Figure 42. openLanguageFile() function .....	44
Figure 43. Table of translation lines .....	45
Figure 44. saveLanguageFile() function .....	46
Figure 45. InstallSchema .....	46
Figure 46. Model of a Module .....	47
Figure 47. Resource Model of a Module .....	47

Figure 48. Collection of a Module .....	47
Figure 49. saveToDatabase() function .....	48
Figure 50. loadFromDatabase() function .....	48

## **Tables**

Table 1. eCommerce platform differences .....	13
Table 2. Magento edition differences .....	17
Table 3. Magento directory structure .....	27
Table 4. Magento module directory structure.....	37

**Acronyms:**

API	Application Programming Interface
CLI	Command Line Interface
Cron	Time-based job scheduler
DB	Database
Ecommerce	Electronic Commerce
HTML	Hypertext Markup Language
i18n	Internationalization and localization
JS	JavaScript
MVC	Model, View, Controller model
OOP	Object-oriented programming
PHP	Hypertext Preprocessor
SQL	Structured Query Language
XML	eXtensible Markup Language



# 1 Introduction

## 1.1 Project Goal

The project's main goal was to address the issue of web translations in many of the company's customer webstores. Since many of these webstores utilise not only Finnish but also Swedish and English languages there is a need for the easy management of translations. The translations are kept in CSV files which are part of the Magento projects' filesystem and the git repositories. This means that the customers can't directly reach them and it comes to a developer to make changes to it which causes unnecessary workload. For this reason a tool is needed that allows editing of said files through a simple tool found in the admin panel of the webstore's magento backend, which is accessible by anyone with admin access.

The idea of this development project and study comes from the employer of the author, Solteq Oyj. The author is currently employed in said company as Software trainee part of the Magento Ecommerce team. This project allowed the author to get familiar with the Magento platform, development processes and practises used in everyday tasks.

## 1.2 Background of the author and Solteq

The author started his studies in Jyväskylä University of Applied Sciences (Jyväskylän ammattikorkeakoulu) as part of a Double Degree programme agreement with his own university, University of Debrecen (Debreceni Egyetem). After a recruitment process the author started his practical training at Solteq in January 2020.

Solteq (Solteq Oyj Website, 2020) is a Nordic IT service provider and software house that specializes in digital business solutions and vertical software markets. The company offers comprehensive digital services for omnichannel business, information management, analytics and business management and delivers digital commerce solutions for eCommerce, product information management, and omnichannel order

and supply chain management. In addition to Finland, the company has offices in Sweden, Norway, Denmark, Poland and the United Kingdom with over 580 employees worldwide. (Solteq Company Information, 2020)

## 2 E-commerce

### 2.1 Introduction to E-commerce

Also known as electronic commerce or internet commerce. It refers to the activity of buying and selling goods or services online. The technologies that allow ecommerce platforms are electronic funds transfer, supply chain management, internet marketing, online transaction processing, electronic data interchange, inventory management systems, data collection. Additional technologies that can support businesses may include live chat support, chat bots, data gathering and the use of demographic data to boost sales.

As seen on **Error! Reference source not found.**, Ecommerce has been growing immensely, and thanks to customers enjoying the comfort of purchasing goods online, this trend is not stopping. (eMarketer Sales Information, 2020)



Figure 1. Ecommerce sales trend

Recently, at the time of writing with the prevalence of the SARS-CoV-2 (Coronavirus), which resulted in people staying home, webstores see an even bigger rise in user numbers. Solteq's own data suggests that some webstores see a peak traffic bigger than during Black Friday of previous years. Some suggests this number will stay, thanks to the fact that even older people are pressured into online shopping, while previously the act of online consumerism was mainly present in younger demographic groups.

## 2.2 Ecommerce categorization

Ecommerce businesses can be categorized by several factors regarding the business model they apply and the service they provide. Each category of ecommerce platforms has different technological and market needs.

### 2.2.1 Business-to-Customer (B2C)

The most common type of relation where the transaction is made between a business and consumer. Buying a product from an online retailer counts as B2C.

### 2.2.2 Business-to-Business (B2B)

B2B ecommerce relates to sales between business entities. Often it refers to the sale of raw materials to other business that are assembled or bundled before facing consumer sales.

### 2.2.3 Consumer-to-Consumer (C2C)

The earliest form of ecommerce, that happens between consumers. Main examples would be sites like eBay or Tori.

### 2.2.4 Consumer-to-Business (C2B)

C2B refers to individual consumers making their own products available for businesses to purchase. This includes the sale of photos online.

### 2.2.5 Business-to-Administration (B2A)

Covers transactions made between businesses and administrations. Best example is services regarding legal documentation and government services.

### 2.2.6 Consumer-to-Administration (C2A)

Consumers selling to administration. It includes online education, consulting, tax preparation among others.

## 2.3 Ecommerce platforms

There are multiple different platforms on the market that each offer different integrations and features along with their own limitations. The market share of the top 5 platforms can be seen on Figure 2. It is important for a retail company to choose the perfect platform that fits their business, campaign strategies and vision. Among the several differences the most important when it comes to decision is the price of up-keep, development processes, the size of business and the ability to grow.

(Ecommerce popularity, 2020)

**Popularity of different eCommerce platforms across the entire web.**

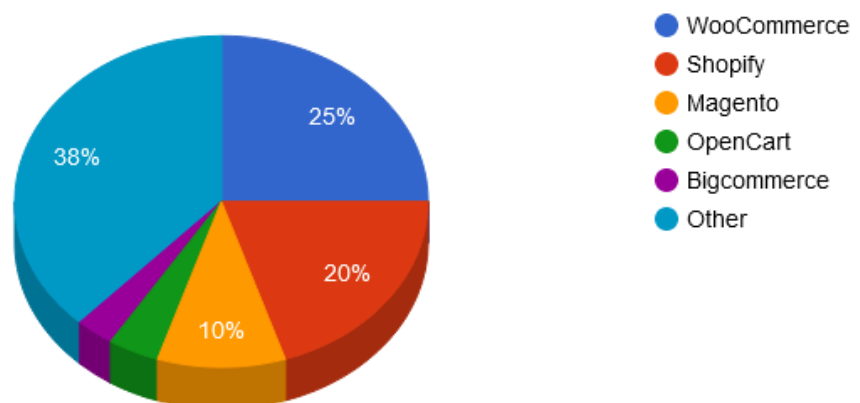


Figure 2. Market share of Ecommerce platforms

The main differences in basic features between the top 5 platform can be summed up on Table 1. (Ecommerce Comparison, 2020)

Table 1. eCommerce platform differences

Shopify	WooCommerce	BigCommerce	Magento	Wix eCommerce
Subscription-based service	Standalone software	Standalone software + subscription-based service	Subscription-based service	Subscription-based service
A subdomain and hosting space for your eCommerce store included + you can hook up your own domain name.	It's part of your existing WordPress website, and it's deeply integrated with it.	A subdomain and hosting space for your eCommerce store included + you can hook up your own domain name.	A subdomain and hosting space for your eCommerce store included + you can hook up your own domain name.	A subdomain and hosting space for your eCommerce store included + a free custom domain name (or connect your own)
Multiple payment gateways	Multiple payment gateways	Multiple payment gateways	Multiple payment gateways	Multiple payment gateways
Basic customer management	Customer management only via third-party plugins	Customer management through customer groups and other	Basic customer management	Basic customer management
100+ professional themes, plus hundreds more third-party	Thousands of themes available on the web (free and paid).	Some themes available	100+ professional themes	500+ professional templates
Mobile-friendly + you can edit HTML and CSS directly	Mobile-friendly to the extent made available by the theme	Mobile-optimized structure	Mobile-friendly	Mobile-friendly
Unlimited bandwidth	Bandwidth depends on the host	Bandwidth depends on the host	Unlimited bandwidth	Unlimited bandwidth
Good reports on sales and store activity	Good reports on sales and store activity	Advanced and in-depth reports on sales and store activity	Great analytics module, with reports, purchase	Okay reports on sales and store activity
Different sales channels, including Point of Sale, Facebook module	Thousands of extensions available	More than 5,000 extensions available	Lets you sell in person with Square, and also sell on marketplaces and social platforms	Hundreds of Wix apps available

As it is seen from this table the difference in main features between the platform are not that big. The differences start to show with development work being done. Some platforms require zero to very little technical, coding knowledge. Platforms like Wix or Shopify for example require zero coding to set up a webshop, and are very accessible. These platforms include easy to use premade themes and WYSIWYG editors that allow people with no expertise in web development to easily make changes and apply them right away to their store. Others allow for extensive customization in just about every aspect, however this requires coding skills and a wide knowledge of the platform. This ties to the question of whether to use an open source platform or a paid software. Open source allows for deeper customization, but if it is ease of access, pre-built templates and customer support feels your needs, paid software is the

one to go for. The next thing to consider is your market. The categories described in 3.2 dictate the needs of your platform. The main markets are B2B and B2C. B2B businesses will need to have the ability to set up business groups, customizable pricing and configuration options. On the other hand, B2C stores needs better marketing tools, advertisement feeds on other websites, better performance to allow for the surge of customers during sales and weekends without interruptions, user recommendations and support for a wide range of payment options and a more available customer support.

Retailers also need to keep in mind the cost of upkeep and maintenance of their webshop. The price of the software, cloud hosting, third party modules all add up to the initial price. However companies also need to consider development and support costs. A pricing example can be seen on Figure 3. (BigCommerce Pricing, 2020)

	Standard	Plus <small>MOST POPULAR</small>	Pro	Enterprise
<a href="#">Monthly</a> <a href="#">Annual</a> \$29.95/mo    \$79.95/mo    \$299.95/mo <a href="#">Contact sales or call 1-866-991-0872 for</a>				
Customer groups and segmentation ⓘ		✓	✓	✓
Abandoned cart saver ⓘ		✓	✓	✓
Persistent Cart ⓘ		✓	✓	✓
Stored credit cards		✓	✓	✓
Google customer reviews			✓	✓
Faceted search (product filtering) ⓘ			✓	✓
Custom SSL ⓘ			✓	✓
Custom facets (product filtering) ⓘ				✓
Price Lists				✓
Unlimited API Calls				✓
Online sales per year <i>Calculated on a trailing 12-month basis</i>	Up to \$50k	Up to \$180k	Up to \$400k**	Custom

Figure 3. The pricing of Bigcommerce with the category differences showcased

## 3 Magento eCommerce

### 3.1 Overview

Magento is an open-source ecommerce platform written in PHP, built upon the Zend Framework. The software was originally developed by Varien Inc, and first published in 2008. Since then it was sold to eBay, later Premira and now it is part of the Adobe software family. It is one of the most popular ecommerce platform on the market. By 2017 out of the top 100,000 website 31.4% ran on Magento. Its wide popularity is thanks to high customizability and scalability.

The main feature set of Magento includes the ability to search and order products in different ways, it provides support for multiple different payment methods, enables the shipping of one order to multiple addresses, accessible admin panel that allows for customization and easy management of products, categories and orders. It has built in Search Engine Optimization, product status and history view, while also supporting multiple languages, currencies and different tax rates. It is highly customizable with many different third party extensions available from many vendors and has the ability to install custom themes and configure them to the needs of every storefront. Figure 4 shows an example of Magento's admin panel.

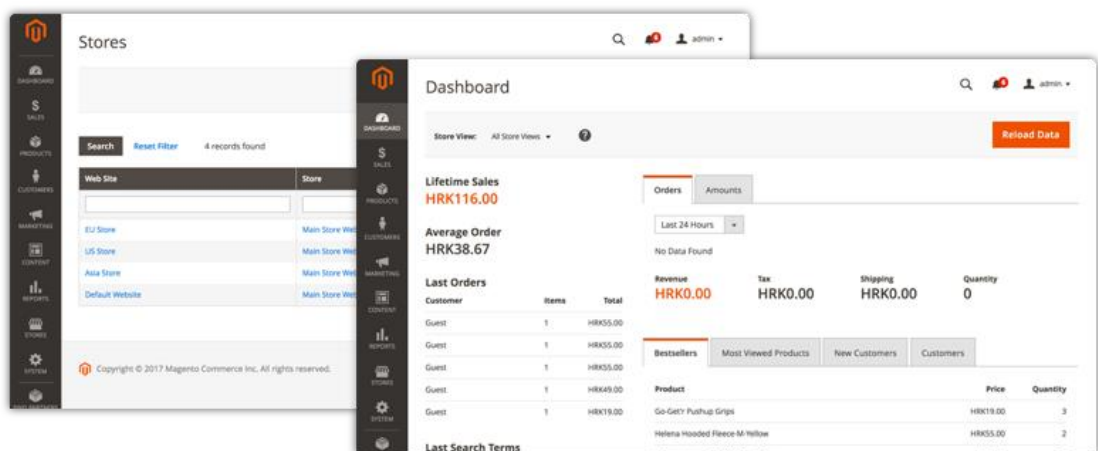


Figure 4. Magento Admin panel

While all of this sounds good on paper, unfortunately it comes with its own negatives. Thanks to its feature rich environment it uses large disk space and memory, and is in need of a hosting platform with high computing ability, otherwise the users can face many problems regarding functionality. Thanks to this it is also relatively slower than other platforms. Whilst it is highly customizable platform it also means that development is no easy task especially for beginners, since the Magento framework is deeply complex.

Magento is constantly in development, meaning that new versions and patches are released frequently. This transforms into more work as updates are to be installed manually, and often so can break functionality, which results in more development work. As it is an in-development platforms as old issues disappear with new releases, new ones might and will appear. As of writing the Magento Github has currently 1200+ open issues.

### 3.2 Magento Community Edition

Community Edition, also known as Opensource Edition is the free version of Magento available to download and use for everyone. Users of the platform are free to make adjustments, further develop and configure the software to their needs. New modules, extensions are constantly being developed by the community.

### 3.3 Magento Commerce Edition

The Commerce Edition, albeit built on the same code stack allows larger companies with higher traffic, larger product inventory and business complexity to perform up to standards thanks to the added benefit of more customizability features and technical support from Magento. Commerce also offers Magento/Adobe's own cloud solution.



### 3.4 Key Differences of Community and Commerce Edition

The main differences of Magento editions are found in the features it offers for retailers. While they offer the same functionality, several tools listed on Table 2 are missing from the free Community edition.

The tools listed on Table 2, are expanded business features. ElasticSearch is an open source search and data analytics engine that is used for website and enterprise search, performance monitoring, metrics and data visualization. Bluefoot is a tool that helps to create and manage website content such as product pages and blogs. Others help the retailer manage more sides of their business straight from Magento, such as shipping.

Table 2. Magento edition differences (Magento Version Comparison, 2020)

Feature	Community	Commerce
License Costs	Free	Revenue based tiered license cost
Responsive Ecomm website	✓	✓
Promotions Engine / Product & Catalog Management	✓	✓
Checkout, Payment, Shipping & Order management	✓	✓
Site management (admin)	✓	✓
ElasticSearch	×	✓
Bluefoot CMS	×	✓
Magento Order Management	×	✓
Content Staging & Preview	×	✓
Magento Shipping	×	✓
Out of the box B2B Functionality	×	✓

As it is seen from the feature list Commerce Edition is truly for larger retail companies with needs for data analytics and tools to boost sales and help further grow their businesses, while smaller might not find need for such tools.

## **4 Magento Environment**

### **4.1 Used Languages**

#### **4.1.1 PHP**

PHP or PHP: Hypertext Preprocessor is a general purpose scripting language mainly suited for web development. PHP code is processed on a web server by an interpreter and the results of the ran code, which can be any type of data or generated HTML content would be a part of the HTML response and is returned to the browser and displayed as plain HTML on the rendered page. This way the client receives only the results of the scripts, without knowing what the underlying code was.

As such, PHP is used to generate dynamic page content, and thanks to its functionality that extends beyond HTML is used to make database operations, collect data and can do file operations on the server. As PHP is free to use and runs on various platform while supporting a wide range of servers and web browsers it is a popular choice in web development.

The main advantage of PHP is it can be embedded into any HTML file and code. The code is enclosed in special start and end instructions as seen on Figure 5.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>

    <?php
      echo "Hi, I'm a PHP script!";
    ?>

  </body>
</html>
```

Figure 5. Example of embedded PHP code in HTML

#### 4.1.2 HTML

HTML, or HyperText Markup Language is the standard markup language for web pages. It can be described as the foundation of a website as its content defines the structure of how a page is built. It uses markup to display text, images and other elements in a web browser. The markup includes special elements, represented by HTML tags that tell the browser how to display the contents, while said tags are hidden and only the referred content is rendered for the user.

As HTML itself only defines the structure, other technologies are used besides it to describe a page's appearance, such as CSS (Cascading Style Sheet) or functionality, behaviour and data behind it, such as JavaScript and PHP (Figure 6).

```

<!DOCTYPE html>
<html>
<body>

<h2>HTML Example</h2>
<p>This is an example HTML only page</p>

<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
</tr>
<tr>
<td>Jill</td>
<td>Smith</td>
<td>50</td>
</tr>
</table>

<button>Button</button>
<a href="https://www.google.com">This is a link</a>

<p>List</p>
<ul>
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ul>

</body>
</html>

```

## HTML Example

This is an example HTML only page

Firstname	Lastname	Age
Jill	Smith	50

Button [This is a link](https://www.google.com)

List

- Coffee
- Tea
- Milk

Figure 6. HTML code and how it's rendered

As seen on Figure 6, HTML only code is only used to define general structure and without any additional technology even simple display features like table borders are missing, which would be added with CSS.

### 4.1.3 JavaScript

JavaScript is a high-level, multi-paradigm programming language that is "just-in-time" compiled, meaning the execution of the code is made during run time rather than before execution, unlike PHP. This allows for functions to run, events to trigger functions and contents to be changed in realtime by manipulating HTML and CSS codes, such allowing for interactive web design.

JavaScript code is ran client-side meaning the browser's API has to support JS to execute its code, but all popular web browser have built in support. Javascript is applied to a HTML page with elements, similar to CSS. In the code it uses the <script> element to let the interpreter know the code under the element is JS. JS code can also be in an external file. To achieve this we need the following added to our HTML code: <script src="script.js" defer></script> where the <script> element defines our file which stores all the JS code instead of explicit JS codes (Figure 7).

```

<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>

<p>The sort() method sorts an array alphabetically.</p>
<button id="button" onclick="sortFruits()">Try it</button>

<p id="paragraph"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("paragraph").innerHTML = fruits;

function sortFruits() {
  fruits.sort();
  document.getElementById("paragraph").innerHTML = fruits;
  document.getElementById("button").innerHTML = "Sorted";
}
</script>

</body>
</html>

```

## JavaScript Array Sort

The sort() method sorts an array alphabetically.

Try it

Banana,Orange,Apple,Mango



Clicking the button

## JavaScript Array Sort

The sort() method sorts an array alphabetically.

Sorted

Apple,Banana,Mango,Orange

Figure 7. Basic JavaScript functionality

## 4.2 Magento Environment Prerequisites

### 4.2.1 Web Server

Magento 2.3 requires Apache 2.4 or nginx 1.x version to be installed. These act as a web server and load balancer for the webstore as well as allowing us to host the Magento store on our own machine as localhost for development purposes.

### 4.2.2 Composer

Composer is a dependency manager tool in PHP. It allows for libraries to be declared that a projects depend on and will install, update them. In Magento it is used to install and Magento core packages, modules extensions. Each project includes a composer.json file which stores the installed module descriptions, and is the recommended way of updating existing packages.

### 4.2.3 Database

Magento uses MySQL version 5.6 or 5.7 for its database. For development purposes the user can install phpMyAdmin as a tool to perform database operations via an user interface in the local development environment.

### 4.2.4 PHP

Magento requires PHP versions 7.2.0 or 7.3.0 along with Zend Framework as these are the technologies Magento is built on and allow for basic functionality.

### 4.2.5 Authentication keys

A Magento Marketplace account is required to generate a pair of authentication keys in order for the user to access the `repo.magento.com` repository where Magento 2 and third-party composer packages are stored.

## 4.3 Magento Environment Install

### 4.3.1 New Project

Once we have all the prerequisites in place we can create a new Magento project with composer. This way composer will automatically install Magento metapackages and their dependencies. This can be achieved with the commands shown on Figure 8.

#### Magento Open Source

```
$ composer create-project --repository-url=https://repo.magento.com/ magento/project-community-edition <install-directory-name>
```

#### Magento Commerce

```
$ composer create-project --repository-url=https://repo.magento.com/ magento/project-enterprise-edition <install-directory-name>
```

Figure 8. Magento composer installation

Depending on whether we want the Community or Enterprise Edition it will install the latest edition of Magento in the directory declared in place of <install-directory-name>.

We can also specify a minor release version to install with the command shown on Figure 9.

```
$ composer create-project --repository-url=https://repo.magento.com/ magento/project-enterprise-edition=2.3.0 <install-directory-name>
```

Figure 9. Magento version specific installation

After we have the Magento metapackage and dependency packages installed, we still need to initialize the project. This setup step will initialize the database and admin user to access the admin panel, along with general store information such as language, currency and time zone. This can be done in a terminal with the command shown on Figure 10.

```
$ bin/magento setup:install \
--base-url=http://localhost/magento2ee \
--db-host=localhost \
--db-name=magento \
--db-user=magento \
--db-password=magento \
--admin-firstname=admin \
--admin-lastname=admin \
--admin-email=admin@admin.com \
--admin-user=admin \
--admin-password=admin123 \
--language=en_US \
--currency=USD \
--timezone=America/Chicago \
--use-rewrites=1
```

Figure 10. Magento setup with command line

Alternatively it can be done with a web setup wizard which can be accessed at <http://localhost/<project-name>/setup> (Figure 11).

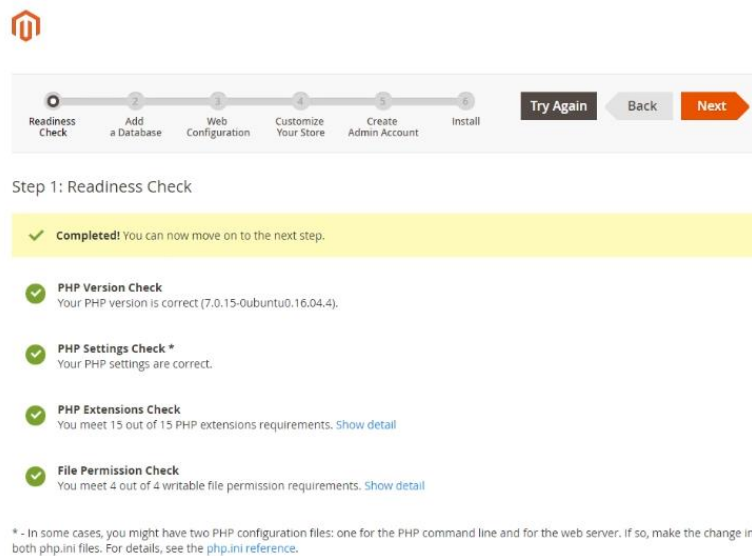


Figure 11. Magento web setup wizard

After the readiness check the setup wizard will walk you through the same configurations as defined in the command line install process, when the process is done we are greeted with the homepage shown on Figure 12.

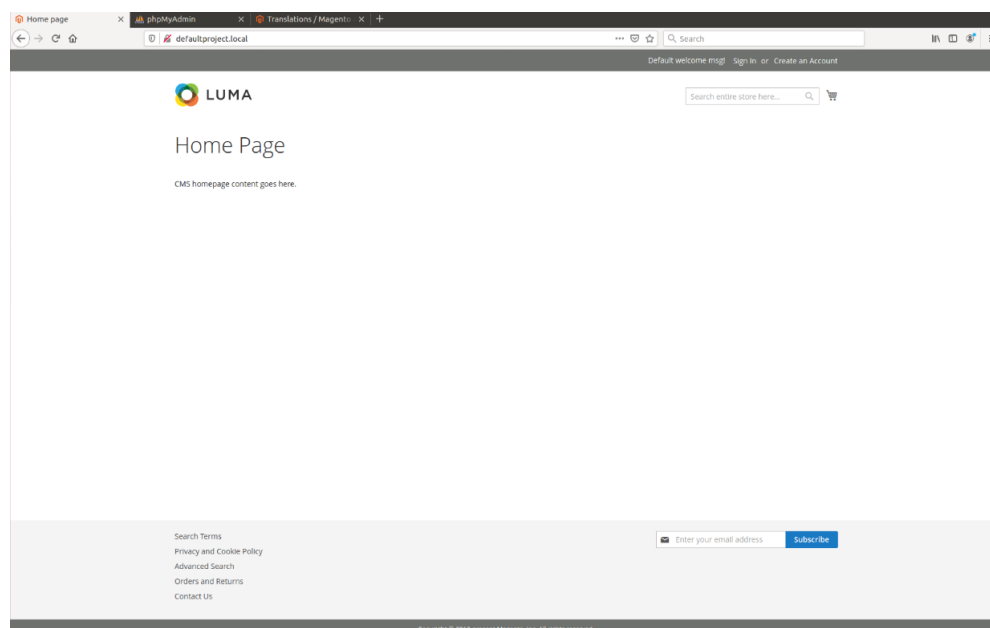


Figure 12. Magento homepage on a fresh install



### 4.3.2 Cloning Existing Projects from Git

Generally Magento repositories will not include base Magento packages and environment specific files, these will have to be setup by the developer.

When the project is pulled to local environment we need to setup the environment file for the webserver. The file is <root>/app/etc/env.php (Figure 13).

```
<?php
return [
    'backend' => [
        'frontName' => 'admin'
    ],
    'crypt' => [
        'key' => '1763d6711fda480f459c34845913c559'
    ],
    'db' => [
        'table_prefix' => '',
        'connection' => [
            'default' => [
                'host' => 'localhost',
                'dbname' => 'helloworld',
                'username' => 'root',
                'password' => 'root',
                'active' => '1'
            ],
            'wordpress' => [
                'host' => 'localhost',
                'dbname' => 'helloworld-wp',
                'username' => 'root',
                'password' => 'root',
                'active' => '1'
            ]
        ]
    ]
];
```

Figure 13. Magento project env.php configuration

The main configurations needed to be done here are the crypt-key, which is used by the database to encrypt sensitive information, and the local database specifics are needed to be configured in order for Magento to use our own local DB for development purposes. If the project uses WordPress, the WP database connection configuration also needs to be setup to use local database. It is found as <root>/wordpress/wp-config.php (Figure 14).

```
<?php
/** The base configuration for WordPress ...*/
// ** MySQL settings - You can get this info from your web host ** //
/* The name of the database for WordPress */
define( 'DB_NAME', 'helloworld-wp' );

/** MySQL database username */
define( 'DB_USER', 'root' );

/** MySQL database password */
define( 'DB_PASSWORD', 'root' );

/** MySQL hostname */
define( 'DB_HOST', 'localhost' );

/** Database Charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8' );

/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );
```

Figure 14. Magento project WordPress configuration

After the project is set up to use local database, we need data to work with. As it is an already existing project, there is more than likely a database for it hosted. This can be copied to our local environment by creating a database dump and importing it to our local database (Figure 15).

```
mysqldump -u root -p DB-NAME > /output_location/dump_name.sql

mysql -u root -p PROJECTNAME < dump_name.sql
```

Figure 15. Commands to create a DB dump and import

Once the database is set up, a fix is needed to replace original domain names with the local one. It can be done easily by executing the SQL query shown on Figure 16.

```
UPDATE
  core_config_data
SET
  value = REPLACE(value, 'original domain', 'projectname.local')
WHERE
  value LIKE '%original domain%';
```

Figure 16. Magento local database fix

After the project is fixed for local use Magento installation can be run with the following steps:

- **composer install**  
Installs core Magento packages and additional ones found in composer.json
- **php bin/magento setup:upgrade**  
With the module installed and enabled the database schema will need to be updated
- **php bin/magento setup:di:compile**  
General code compiler
- **php bin/magento indexer:reindex**  
Reindexes all Magento indexes
- **php bin/magento cron:run**  
Enables CRON jobs to run
- **php bin/magento setup:static-content:deploy**  
Deploys static view files such as images and CSS

After the setup is done Magento will be accessible on our local machine, however and admin user still needs to be added for access to the admin panel (Figure 17).

```
solteq@MagVM:/var/www/defaultproject$ php bin/magento admin:user:create --admin-user="admin1"
Admin password:
Admin email: admin@example.com
Admin first name: Admin
Admin last name: Admin
Created Magento administrator user named admin1
```

Figure 17. Magento Admin user creation

## 4.4 File Structure

Magento's each directory contains files related to one specific business feature as seen on Table 3.

Table 3. Magento directory structure

Directory	Purpose
/app	This is where the core php code is located. It contains all coded configurations, custom modules and themes
/bin	Contains all Magento 2 CLI executable scripts. The Magento CLI utility commands help install and manage modules, cache and indexers along with many more command features.
/dev	Stores automated functional tests which are run by the Magento Test Framework.
/generated	The folder where Magento's generated code is stored. As default configuration if a class is injected in a constructor, the code will be generated by Magento to create non-existent factory classes.
/lib	This directory contains the Magento core code along with the software's PHP library, basically all the non-module based Magento code.
/phpserver	Contains the Router.php file which is PHP's built in web server that provides a router script to use with server rewrites.
/pub	Contains all publicly accessible files such as static files and site media like images and videos. It also contains the used theme's generated static files.
/setup	Installation setup files are located inside this folder used for installation processes mainly, alongside with a pre-included performance toolkit.
/update	Contains files used by Magento during upgrade processes.
/var	Includes generated classes, sessions and caches. The content is generated here when <code>php bin/magento setup:di:compile</code> is run. <code>var/log</code> contains log and error files, <code>var/cache</code> contains all of Magento's cache.
/vendor	The vendor directory contains the framework core of base Magento modules and this is where all additionally installed third party modules, that are defined in <code>composer.json</code> are installed. The contents of this directory are generated at install from <code>composer.json</code> .

## 5 Magento Modules

### 5.1 Overview

Modules and themes act as the front for customization in Magento. A module encapsulates one specific feature and has minimal dependencies on other modules. They provide specific business features, with supporting logic. Themes serve the same purpose, but unlike modules they affect the storefront appearance and influence user experience.

The purpose of one module is to implement specific features needed for one e-commerce store by implementing new functionality or by extending functionality of already existing modules. Each module is designed to function independently from others if they don't act as extensions to others, so that the workings of one module does not affect the functionality of others.

### 5.2 Module Relations

#### 5.2.1 Overview

As described above modules can extend other modules, thus creating a relation between them. (Magento Module Relations, 2020)

- **uses:** module A uses module B if it invokes behavior of module B
- **reacts to:** module A reacts to module B if its behavior is triggered by an event in module B without module B knowing about module A
- **customizes:** module A customizes module B if it modifies the behavior of module B
- **implements:** module A implements module B if it implements some, not necessarily all, behavior that is defined in module B
- **replaces:** module A replaces module B if it provides its own version of the API exposed and implemented by module B

### 5.2.2 A module uses B, C customizes B

As seen on Figure 18, module A uses module B and module C customizes module B, the customizations in module C must not break the API of module B so that module A still functions properly.

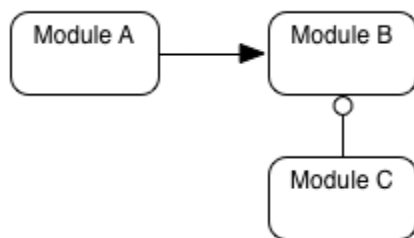


Figure 18. A uses B, C customizes B

### 5.2.3 A module reacts to B, C customizes B

As seen on Figure 19, module A reacts to module B and module C customizes module B, the customizations in module C must not interfere with the events in module B that module A depends on.

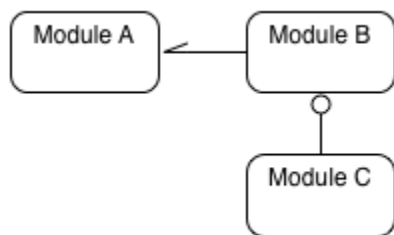


Figure 19. A reacts to B, C customizes B

### 5.2.4 A module and C customize B

As seen on Figure 20, if both module A and C customize module B, be careful about how these customizations are implemented so that you avoid conflicts

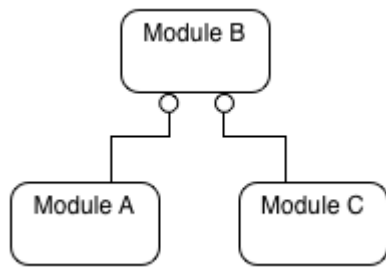


Figure 20. A and C customize B

### 5.2.5 A module replaces B

As seen on Figure 21, if module A replaces module B, it needs to be able to do so in such a way that other modules are not affected. That will mean not having direct hard dependencies on module B, but rather dependencies on a third module, module C, that both module A and B implement.

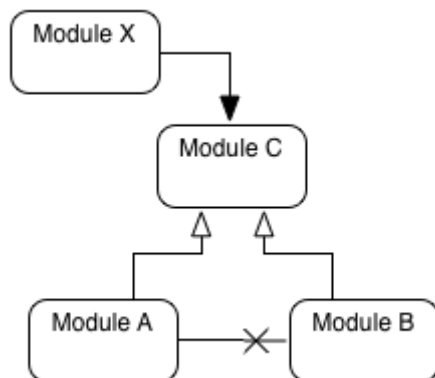


Figure 21. A replaces B

## 5.3 Module Dependencies

### 5.3.1 Overview

Software dependency identifies a software component's reliance on another for proper functioning. A core principle of Magento architecture is the minimization of software dependencies. Instead of being closely interrelated with other modules,

modules are optimally designed to be loosely coupled. Loosely coupled modules require little or no knowledge of other modules to perform their tasks.

Each Magento module is responsible for a unique feature. This means that several modules cannot be responsible for one feature, while one module can only be responsible for one feature. Module dependencies must be declared explicitly with any other component dependency such as theme, language package or library. Disabling one module should not result in disabling others.

### 5.3.2 Hard dependency

A module with a hard dependency on another module cannot function without the module it depends on. These modules contain code that uses logic from another module, such as class constrains, static method, interfaces etc, or dependent on other modules' classes, methods. Hard dependency can also be achieved if one modules uses the database table declared by another module.

The module's own composer.json file contains hard dependency definitions as seen on Figure 22.

```
{
  "name": "amasty/module-salesrulewizard",
  "version": "1.0.0",
  "dist": {
    "type": "zip",
    "url": "https://composer1.amasty.com/enterprise/packages/module-salesrulewizard-1.0.0-ee.zip"
  },
  "require": {
    "amasty/base": ">=1.5.5",
    "amasty/promo": ">=2.4.0",
    "php": "~5.6.5|7.0.2|7.0.4|~7.0.6|^7.1.0"
  },
}
```

Figure 22. Hard dependency example

### 5.3.3 Soft dependency

A module with a soft dependency on another module can function properly without the other module, even if it has a dependency on the other module. It is present when one module checks another's availability, extends its configuration or layout.

The module's `composer.json` file declares soft dependency as seen on Figure 23.

```

"name": "amasty/module-salesrulewizard",
"version": "1.0.0",
"dist": {
  "type": "zip",
  "url": "https://composer1.amasty.com/enterprise/packages/module-salesrulewizard-1.0.0-ee.zip"
},
"suggest": {
  "magento/module-graph-ql": "*",
  "magento/module-graph-ql-cache": "*",
  "magento/module-store-graph-ql": "*"
}

```

Figure 23. Soft dependency example

## 5.4 Module Areas

An area is a logical component that organizes code for optimized request processing. Magento uses areas to streamline web service calls by loading only the dependent code for the specified area. Each of the default areas defined by Magento can contain completely different code on how to process URLs and requests.

Magento is organized into these areas:

- **Admin panel (adminhtml):** Admin area contains all the code needed for Magento's admin panel (store management).
- **Storefront (frontend):** The storefront (or frontend) contains template and layout files that define the appearance of your storefront.
- **Basic (base):** used as a fallback for files absent in adminhtml and frontend areas.
- **Cron (crontab):** Contains cron jobs that are scheduled activities, such as reindexing, currency rate updates, newsletter sendout etc.

Modules define which resources are visible and accessible in an area, as well as an area's behavior. The same module can influence several areas, but each area must have separate behavior and view components. Areas can be disabled within a module and the developer must ensure it does not depend on other modules' areas.



## 5.5 Module Management

### 5.5.1 Composer Install

Installing a module via Composer is the recommended way. Using the extension name and version we can install the module, which will also update the require section of the project's `composer.json` file (Figure 24).

```
composer require mageplaza/magento-2-blog-extension
php bin/magento setup:upgrade
php bin/magento setup:static-content:deploy
```

Figure 24. Module installation with Composer

The commands above will install the module alongside updating dependencies. `Setup:upgrade` is necessary to update the database scheme with additional tables from the new module.

Often times, modules will require authentication. This is the case with payed modules, where the developer will need to enter the authentication key pair to access the module's repository. The key pair will be added to `auth.json`, so later updates will not require manual authentication (Figure 25).

```
magento@server:~/magento/sam/ce216$ composer remove mageplaza/module-core
Loading composer repositories with
Authentication required (repo.magento.com):
Username: 
```

Figure 25. Authentication during module installation

## 5.5.2 Manual Install

Some modules don't support Composer and will have to be installed manually. This is simply done by downloading the module's package and copying it into our project's /code directory and running the commands shown on Figure 24. without "composer require" as the module lacks support.

## 5.5.3 Managing modules

The composer.json file contains all modules installed via composer alongside their required version number (Figure 26).

```
{
  "name": "magento/project-community-edition",
  "description": "eCommerce Platform for Growth (Community Edition)",
  "type": "project",
  "license": [
    "OSL-3.0",
    "AFL-3.0"
  ],
  "config": {
    "preferred-install": "dist",
    "sort-packages": true
  },
  "require": {
    "magento/product-community-edition": "2.3.3"
  },
  "require-dev": {
    "allure-framework/allure-phpunit": "~1.2.0",
    "friendsofphp/php-cs-fixer": "~2.14.0",
    "lusitanian/oauth": "~0.8.10",
    "magento/magento-coding-standard": "~3.0.0",
    "magento/magento2-functional-testing-framework": "2.4.5",
    "pdepend/pdepend": "2.5.2",
    "phpmd/phpmd": "@stable",
    "phpunit/phpunit": "~6.5.0",
    "sebastian/phpcpd": "~3.0.0",
    "squizlabs/php_codesniffer": "~3.4.0"
  }
}
```

Figure 26. composer.json example

Further information of the modules can be found in composer.lock where the download location of said module can be found alongside with a checksum to ensure it downloaded without error. Module dependencies are also found here, see Figure 27.

```

{
  "name": "amzn/amazon-pay-module",
  "version": "3.2.13",
  "dist": {
    "type": "zip",
    "url": "https://repo.magento.com/archives/amzn/amazon-pay-module/amzn-amazon-pay-module-3.2.13.0.zip",
    "shasum": "598005edb135ebf19590fb11810ba08436333ef1"
  },
  "require": {
    "amzn/amazon-pay-and-login-with-amazon-core-module": "^3.2.13",
    "amzn/login-with-amazon-module": "^3.2.13",
    "magento/framework": "^102",
    "magento/module-backend": "^101",
    "magento/module-catalog": "^103",
    "magento/module-checkout": "^100.3",
    "magento/module-customer": "^102",
    "magento/module-directory": "^100.3",
    "magento/module-eav": "^102",
    "magento/module-payment": "^100.3",
    "magento/module-paypal": "*",
    "magento/module-quote": "^101.1",
    "magento/module-sales": "^102",
    "magento/module-store": "^101",
    "php": "~7.1.3||~7.2.0||~7.3.0"
  },
  "suggest": {
    "magento/module-customer": "*",
    "magento/module-scalable-checkout": "*",
    "magento/module-scalable-oms": "*"
  },
  "type": "magento2-module",
  "autoload": {
    "files": [
      "registration.php"
    ],
    "psr-4": {
      "Amazon\\Payment\\": ""
    }
  },
  "license": [
    "Apache-2.0"
  ],
  "description": "Amazon Pay module"
},

```

Figure 27. composer.lock example of a specific module

Modules can be enabled or disabled with the commands shown on Figure 28.

Enable a module.

```
$ bin/magento module:enable <module-name>
```

Disable a module.

```
$ bin/magento module:disable <module-name>
```

Figure 28. Change module state

These commands will change the module's state in the `/app/etc/config.php` file alongside updating the database to reflect changes as shown on Figure 29.

```
<?php
return [
    'modules' => [
        'Magento_AdminAnalytics' => 1,
        'Magento_Store' => 1,
        'Magento_AdvancedPricingImportExport' => 1,
        'Magento_Directory' => 1,
        'Magento_Amqp' => 1,
        'Magento_AmqpStore' => 1,
        'Magento_Config' => 1,
        'Magento_Theme' => 1,
        'Magento_Backend' => 1,
        'Magento_Variable' => 1,
        'Magento_Eav' => 1,
        'Magento_Customer' => 1,
        'Magento_AuthorizenetGraphQL' => 1,
        'Magento_Search' => 1,
        'Magento_Backup' => 1,
        'Magento_AdminNotification' => 1,
        'Magento_BraintreeGraphQL' => 1,
        'Magento_Indexer' => 1,
        'Magento_Authorization' => 1,
    ],
];
```

Figure 29. config.php declaring module states

It is important to note that modules installed via composer will be located in the /vendor directory, and as such will be installed during the project's setup. The project's composer.json and composer.lock file must be part of the git repository as the composer install command reads the composer.lock file to enable the defined dependencies in the remote environment.

## 5.6 Module structure

Magento modules are installed and developed in /app/code directory in the following way: <vendor-name>/<module-name>/... The contents of each directory are explained in detail in Table 4.

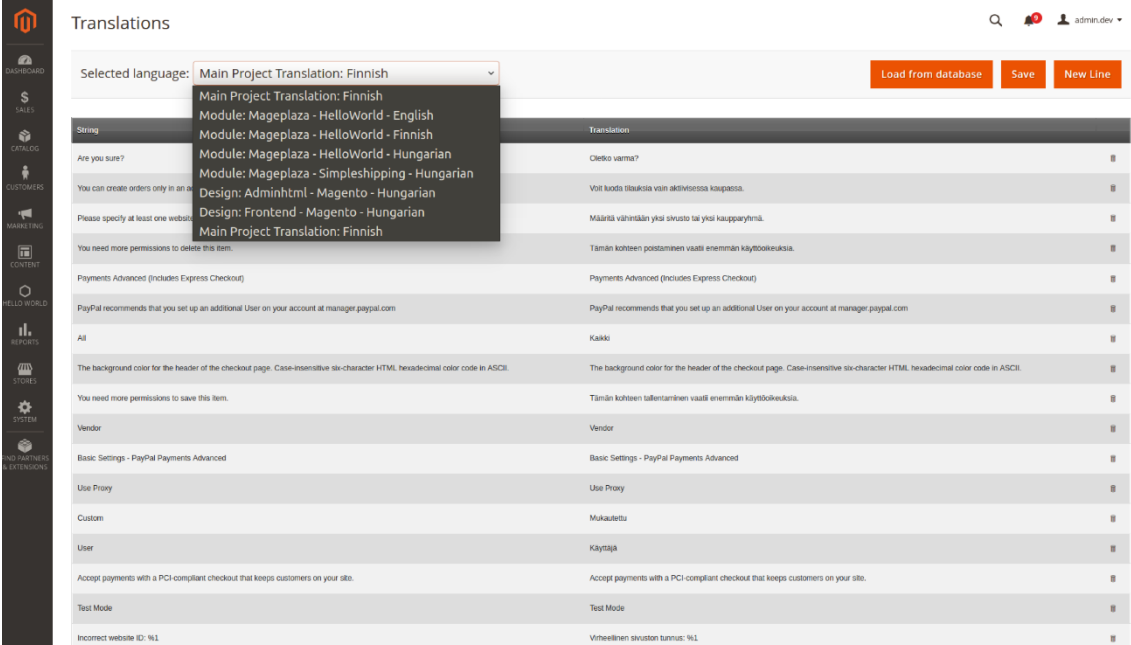
Table 4. Magento module directory structure

Directory	Purpose
/Api	Contains any PHP classes exposed to the API. Magento allows the developer to create an own custom API to perform an action based on response. While Magento supports default API features like product or order APIs, custom responses are to be developed here. Magento supports both SOAP and REST.
/Block	Contains PHP view classes as part of Model View Controller(MVC) vertical implementation of module logic. It contains all application view logic without any html or css.
/Console	Contains custom CLI commands
/Controller	Contains PHP controller classes as part of MVC vertical implementation of module logic. Generally controls how the page is built to render view.
/Cron	This directory contains the Magento core code along with the software's PHP library, basically all the non-module based Magento code.
/etc	Contains module specific configuration that are required for the module to work.
/Helper	Contains helpers, that are used to override or modify functionality of other modules or core Magento. As it is not recommended to modify core files, helpers can be written and their modified functions or classes called when needed.
/Model	Contains PHP model classes as part of MVC vertical implementation of module logic. In Magento 2 CRUD, models have many different functions such as manage data, install or upgrade module.
/Observer	Observers influence general behavior, performance, or change business logic. Observers are executed whenever the event they are configured to watch is dispatched by the event manager.
/Plugin	A plugin, or interceptor, is a class that modifies the behavior of public class functions by intercepting a function call and running code before, after, or around that function call.
/setup	Contains classes for module database structure and data setup which are invoked when installing or upgrading.
/view	Contains view files, including static view files, design templates, email templates, and layout files. HTML and CSS files go here.

## 6 Translation Module

### 6.1 Overview

The purpose of the module is to add the ability to manage translation .csv files straight from the admin panel. This is done by creating a new configuration page which finds and loads all the .csv files found in directories where localization files are placed. The overview of the module can be seen on Figure 30.



The screenshot shows the 'Translations' module in the Magento 2 admin panel. The interface includes a sidebar with navigation options, a top navigation bar, and a main content area. A dropdown menu is open, showing a list of languages and modules. The main content area contains a table of translations with columns for 'String' and 'Translation'. The table lists various strings and their corresponding translations in Finnish.

String	Translation
Are you sure?	Oleko vaima?
You can create orders only in an...	Voit luoda tilauksia vain aktiivisessa kaupassa.
Please specify at least one webos...	Määritä vähintään yksi sivusto tai yksi kaupparyhmä.
You need more permissions to delete this item.	Tämän kohteen poistaminen vaatii enemmän käyttöoikeuksia.
Payments Advanced (includes Express Checkout)	Payments Advanced (includes Express Checkout)
PayPal recommends that you set up an additional User on your account at manager.paypal.com	PayPal recommends that you set up an additional User on your account at manager.paypal.com
All	Kaikki
The background color for the header of the checkout page. Case-insensitive six-character HTML hexadecimal color code in ASCII.	The background color for the header of the checkout page. Case-insensitive six-character HTML hexadecimal color code in ASCII.
You need more permissions to save this item.	Tämän kohteen tallentaminen vaatii enemmän käyttöoikeuksia.
Vendor	Vendor
Basic Settings - PayPal Payments Advanced	Basic Settings - PayPal Payments Advanced
Use Proxy	Use Proxy
Custom	Mukautettu
User	Käyttäjä
Accept payments with a PCI-compliant checkout that keeps customers on your site.	Accept payments with a PCI-compliant checkout that keeps customers on your site.
Test Mode	Test Mode
Incorrect website ID: %1	Virheellinen sivusto tunnus: %1

Figure 30. Translation module in admin panel

The admin has the ability to open files, add new lines, delete or modify already exist- ing ones. A function is added to save to and load changes from the database. This is needed since localization files are part of the git repository and will reset to their original version at every new version deploy. The changes can be loaded from the da- tabase table and will be saved to the deployed files.

## 6.2 Initial Configuration

To allow the module to work and exist in Magento we need certain configurations done. The first required file is `<module>/etc/module.xml` This is required for the module to exist as it contains the name, version and dependencies (Figure 31).

```
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
  <module name="Solteq_TranslationManagement" setup_version="1.0.0">
  </module>
</config>
```

Figure 31. module.xml of translation module

Each module must also have a registration.php file, which tells Magento how to locate the module. It is a standardized file that follows the same pattern for all modules (Figure 32).

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    type: \Magento\Framework\Component\ComponentRegistrar::MODULE,
    componentName: 'Solteq_TranslationManagement',
    path: __DIR__
);
```

Figure 32. registration.php of translation module

Next, the access location of the module needs to be specified. It can be done in the `<module>/etc/adminhtml/menu.xml` file (Figure 33).

```
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend/etc/menu.xsd">
  <menu>
    <add id="Solteq_TranslationManagement::translations" title="Manage Translation" module="Solteq_TranslationManagement"
      sortOrder="11" action="solteq_translationmanagement/translations/index"
      resource="Solteq_TranslationManagement::translations" parent="Magento_Backend::system_other_settings"/>
  </menu>
</config>
```

Figure 33. menu.xml of translation module

This file tells Magento that the module will be found in the Admin Panel, System menu, under Other Settings category (Figure 34).

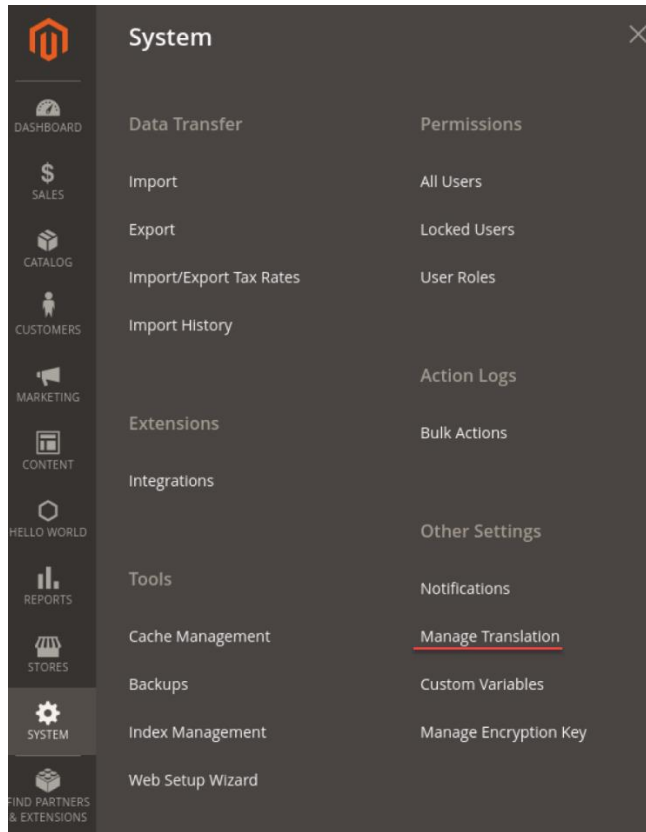


Figure 34. Effects of menu.xml of translation module

In the same directory a routes.xml file is also needed, which maps the module to a specific URL (Figure 35).

```

<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
  <router id="admin">
    <route id="solteq_translationmanagement" frontName="solteq_translationmanagement">
      <module name="Solteq_TranslationManagement"/>
    </route>
  </router>
</config>
  
```

Figure 35. routes.xml of translation module



### 6.3 Frontend View

Magento's view directory contains all the logic connected to the front-end view of the module (Figure 36).

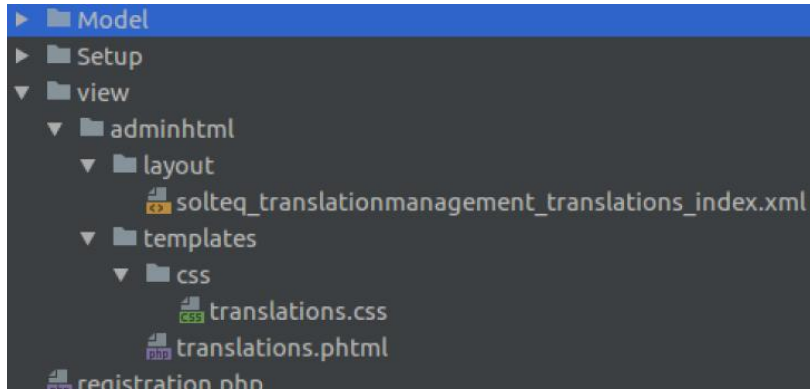


Figure 36. Structure of view directory that contains all display logic

The layout.xml file defines the page structure (Figure 37). It serves the purpose of declaring the Block class, which will contain all background logic such as functions and classes, and declares the template, which is the main HTML file of the module. It contains HTML code alongside JavaScript and PHP that defines the structure of the web page and calls functions from the Block.

```

1 <page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../../../../../../lib/internal
2 <update handle="styles"/>
3 <body>
4     <referenceContainer name="content">
5         <block class="Solteq\TranslationManagement\Block\Index" name="solteq_translationmanagement_translations_index"
6             template="Solteq_TranslationManagement::translations.phtml" />
7     </referenceContainer>
8 </body>
9 </page>
  
```

Figure 37. layout .xml file of translation module

## 6.4 Background Logic

### 6.4.1 Controller

Controller is the most important part of module development. It receives requests, processes and renders the page. Its main purpose is to initialize PageFactory, alongside with Model objects to work with the database. The module's controller is shown on Figure 38.

```

public function __construct(
    Context $context,
    PageFactory $resultPageFactory,
    TranslationFactory $translationFactory,

    Registry $registry
) {
    parent::__construct($context);
    $this->registry = $registry;
    $this->resultPageFactory = $resultPageFactory;
    $this->translationFactory = $translationFactory;
}

public function execute()
{
    $resultPage = $this->resultPageFactory->create();
    $resultPage->getConfig()->getTitle()->prepend(__('Translations'));
    $model = $this->translationFactory->create();

    $block = $this->resultPageFactory
        ->create()
        ->getLayout()
        ->createBlock( type: 'Soliteq\TranslationManagement\Block\Index');

    $postparams = [
        'langFile' => $this->getRequest()->getParam( key: 'lang_file'),
        'newLine' => $this->getRequest()->getParam( key: 'new_line'),
        'deleteLine' => $this->getRequest()->getParam( key: 'delete_line'),
        'editedArray' => $this->getRequest()->getParam( key: 'editedArray'),
        'loadFromDatabase' => $this->getRequest()->getParam( key: 'load_database'),
    ];
}

```

Figure 38. Controller of translation module

The execute() function of controller is a dispatch method that will initialize the web page when it is called to be loaded. In this function PageFactory will render the page, and will define the Block, where functions will be called from. It is also the place to declare HTTP parameters such as GET and POST, as it will check for those parameters, and can call functions based on their values. All of the modules functionality, such as loading, editing files, lines are called by POST requests (Figure 39).

```

if(isset($postparams['langFile'])) {
    $this->registry->register( key: 'currentFile', $postparams['langFile']);
}

if(isset($postparams['newLine'])) {
    $block->newLine($postparams['langFile']);
}

```

Figure 39. Controller checking POST parameters to call functions from Block

#### 6.4.2 Loading files

Upon the HTML is initialized it calls findLanguageFiles() function from Block to load and display all files from the language select dropdown menu seen on Figure 40.

```

public function findLanguageFiles()
{
    $this->listFolderFiles($this->dir->getPath( code: 'app'));
    return $this->_languageFiles;
}

function listFolderFiles($dir)
{
    $files = scandir($dir);
    foreach ($files as $file) {
        if ($file != '.' && $file != '..') {
            if (substr($file, start: -4) == '.csv') {
                $this->_languageFiles[] = $dir . '/' . $file;
            }
            if (is_dir( filename: $dir . '/' . $file)) {
                $this->listFolderFiles( dir: $dir . '/' . $file);
            }
        }
    }
    return;
}

```

Figure 40. findLanguageFiles() function

The findLanguageFiles() functions calls listFolderFiles() with the parameter of directory path of /app folder. This recursive function goes through all folders and subfolders inside /app to find all language .csv files. findLanguageFiles() returns \_languageFiles[] array, which will be displayed in the dropdown menu, that also acts as the selector to open said files. The code is seen on Figure 41.

```

<div class="page-actions-inner" data-title="Edit translations">
  Selected language:
  <form method="post" action="" class="file-selector">
    <select name="lang_file" onchange="this.form.submit()">
      <option value="">
        <?php if ($currentFile != ""): ?>
          <?= $block->languageFileToName($currentFile) ?>
        <?php else : ?>
          <?= 'Select a language' ?>
        <?php endif ?>
      </option>
      <?php foreach ($languageFiles as $language) : ?>
        <option value="<?= $language ?>"> <?= $block->languageFileToName($language) ?> </option>
      <?php endforeach; ?>
    </select>
    <input type="hidden" name="form_key" value="<?= $block->getFormKey() ?>" />
  </form>

```

Figure 41. Dropdown menu code to display and select files

The dropdown menu also calls `languageFileToName()` function for each file, that is a simple string parser function to display each file in an easily readable name instead of displaying the whole directory path. Selecting an item from the dropdown list will act as submitting this POST form with the selected language and `form_key` as the parameters. The `form_key` value is needed to validate the POST request. If it is missing Magento will send an error. After the POST request is sent, the page gets reloaded, calling the Controller's `execute()` function again, which will check if POST parameters are set, and saves the current file to registry for further use and calls `openLanguageFile()` function, which is shown on Figure 42.

```

function openLanguageFile($languageFile)
{
    $langArray = [];
    if (($langFile = fopen($languageFile, 'r')) !== false) {
        while (($data = fgetcsv($langFile, 'length: 0', 'delimiter: ', ',')) !== false) {
            if (isset($data[2]) && isset($data[3])) {
                $langArray[] = array(
                    $data[0],
                    $data[1],
                    $data[2],
                    $data[3]
                );
            }
            else {
                $langArray[] = array(
                    $data[0],
                    $data[1]
                );
            }
        }
        fclose($langFile);
    }
    return $langArray;
}

```

Figure 42. `openLanguageFile()` function

The function uses PHP's built in csv parser to open the files and save them to arrays. Generally .csv files have 2 values separated by a delimiter, however some files can have a third and fourth value defining the module where those strings are used.

These values are also needed to be loaded, so they can be saved to the database later.

### 6.4.3 Editing translations

The values of .csv files are displayed in an editable table, in which each line acts as a form input value (Figure 43).

```

<table id="editable_table">
  <tr>
    <th>String</th>
    <th>Translation</th>
    <th class="delete-col"></th>
  </tr>
  <form id="save-edit" method="post" action="">
    <?php for ($i=0;$i<count($langArray);$i++) : ?>
      <tr>
        <td>
          <input class="csv-field" type="text" name="editedArray[<?=$ i ?>][0]"
            value="<?=$ htmlentities($langArray[$i][0]) ?>" />
        </td>
        <td>
          <input class="csv-field" type="text" name="editedArray[<?=$ i ?>][1]"
            value="<?=$ htmlentities($langArray[$i][1]) ?>" />
          <?php if(isset($langArray[$i][2])) : ?>
            <input type="hidden" name="editedArray[<?=$ i ?>][2]" value="<?=$ langArray[$i][2] ?>" />
          <?php endif; ?>
          <?php if(isset($langArray[$i][3])) : ?>
            <input type="hidden" name="editedArray[<?=$ i ?>][3]" value="<?=$ langArray[$i][3] ?>" />
          <?php endif; ?>
        </td>
        <td>
          <button class="delete-button" title="Click to delete line" type="submit" name="delete_line"
            form="delete-line" value="<?=$ i ?>" />
            &#128465;
          </button>
        </td>
      </tr>
    <?php endfor; ?>
    <input type="hidden" name="form_key" value="<?=$ block->getFormKey() ?>" />
    <input type="hidden" name="lang_file" value="<?=$ currentFile ?>" />
  </form>
</table>

```

Figure 43. Table of translation lines

Here each line of the table edits the `langArray[][]` array that stores all of the opened file's lines. When the Save button is clicked, the array will be sent in a POST request, and as described above, the Controller will check for the parameter and call the `saveLanguageFile()` function in Block to save the changes, which is a simple function that overwrites the whole content of the opened file the contents of the array. The function is shown on Figure 44.

```
function saveLanguageFile($arrayToSave, $languageFile)
{
    if (($langFile = fopen($languageFile, mode: "w")) !== false) {
        foreach ($arrayToSave as $lines) {
            fputs($langFile, $lines);
        }
        fclose($langFile);
    }
    return;
}
```

Figure 44. saveLanguageFile() function

Deleting a line works also calls the same function, but without that one line in the array that is being deleted, while Adding a new line will simply append an empty value to the end of langArray[][].

#### 6.4.4 Database connection

In order to use a database, a connection Model is needed. First an InstallSchema is needed, which will contain the database table's attributes. This file will run when the module is installed, and creates the table (Figure 45).

```
class InstallSchema implements \Magento\Framework\Setup\InstallSchemaInterface
{
    public function install(\Magento\Framework\Setup\SchemaSetupInterface $setup, \Magento\Framework\Setup\ModuleContextInterface $context)
    {
        $installer = $setup;
        $installer->startSetup();
        if (!$installer->tableExists('solteq_translationmanagement')) {
            $table = $installer->getConnection()->newTable(
                $installer->getTable('solteq_translationmanagement')
            )
            ->addColumn(
                'id',
                \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
                255,
                [
                    'nullable' => false,
                    'primary' => true,
                ],
                'comment: 'ID'
            )
            ->addColumn(
                'string',
                \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
                255,
                ['nullable' => false, 'default'=>'],
                'comment: 'String variable name'
            )
            ->addColumn(
                'translation',
                \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
                255,
                ['nullable' => false, 'default'=>'],
                'comment: 'Translation text'
            )
        }
    }
}
```

Figure 45. InstallSchema

A model is to be created in <module>/Model, which will called whenever a model is instantiated, and will define the resource model which will actually fetch the information from the database (Figure 46).

```

<?php
namespace Solteq\TranslationManagement\Model;
class Translation extends \Magento\Framework\Model\AbstractModel
{
    protected function _construct()
    {
        $this->_init( resourceModel: 'Solteq\TranslationManagement\Model\ResourceModel\Translation');
    }
}

```

Figure 46. Model of a Module

The Resource model will contain database logic and executes SQL queries. It will define the database table and the primary key (Figure 47).

```

<?php
namespace Solteq\TranslationManagement\Model\ResourceModel;

class Translation extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    public function _construct(
        \Magento\Framework\Model\ResourceModel\Db\Context $context
    )
    {
        parent::_construct($context);
    }

    protected function _construct()
    {
        $this->_init( mainTable: 'solteq_translationmanagement', idFieldName: 'id');
        $this->_isPkAutoIncrement = false;
    }
}

```

Figure 47. Resource Model of a Module

A Collection will also needs to be created, as this will allow to filter and fetch a collection from the table (Figure 48).

```

<?php
namespace Solteq\TranslationManagement\Model\ResourceModel\Translation;

class Collection extends \Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection{
    public function _construct()
    {
        $this->_init( model: "Solteq\TranslationManagement\Model\Translation",
            resourceModel: "Solteq\TranslationManagement\Model\ResourceModel\Translation");
    }
}
?>

```

Figure 48. Collection of a Module

A Factory object is needed to use the Model. This is used to instantiate the object. It is automatically created by Magento, same way as the pageFactory, it has to be declared in Controller, it is seen on Figure 38 as TranslationFactory.

#### 6.4.5 Saving and Loading to database

Whenever a change is made in the translation files a saveToDatabase() function is called. This function turns the array of translations into a model in accordance to our database schema and saves it. The function is displayed on Figure 49.

```
function saveToDatabase($langArray, $file)
{
    $model = $this->translationFactory->create();
    foreach ($langArray as $line) {
        if(isset($line[2]) && isset($line[3])) {
            $model->addData([
                'id' => hash('ripemd160', $data:$line[0] . $file . $line[2] . $line[3]),
                'string' => $line[0],
                'translation' => $line[1],
                'location' => $file,
                'parent_type' => $line[2],
                'parent_name' => $line[3]
            ]);
        }
        else {
            $sid = hash('ripemd160', $data:$line[0] . $file);
            $model->addData([
                'id' => $sid,
                'string' => $line[0],
                'translation' => $line[1],
                'location' => $file,
            ]);
        }
    }
    $model->save();
}
```

Figure 49. saveToDatabase() function

One thing to note is that the table requires a unique id for each record. The solution was to create hash ids from each line. This way we can have unique ids. (Figure 50).

```
public function loadFromDatabase()
{
    $model = $this->translationFactory->create();
    $collection = $model->getCollection();
    $databaseArray = [];
    foreach ($collection as $data) {
        $databaseArray[$data['location']][] = [
            $data['string'],
            $data['translation'],
            $data['parent_type'],
            $data['parent_name']
        ];
    }
    foreach ($databaseArray as $key => $lines) {
        $this->saveLanguageFile($lines, $key);
    }
    return;
}
```

Figure 50. loadFromDatabase() function



## 7 Conclusion

The main goal of the thesis project was to find a solution where web admins can make changes to localization when need be instead of creating a support ticket and going through a development process.

As this was the first ever big project in Magento it caused some difficulties at the beginning, as the Platform itself is different from any other. During the first few weeks of working on it the focus was mainly on understanding the platform itself, and how changes relate to other parts of the project and getting familiar with Magento's development processes. However it provided a great learning platform to improve and it was generally easy to develop thanks to the documentation and developing tutorials found online, and after taking the first steps and having an understanding of the platform, development proceeded quickly.

The result of the project is a fully working module that can load .csv localization file and the user can edit said files with a user-friendly web editor and save changes locally with the added ability to synchronize changes with the database. The module is ready to be installed in any Magento project. It was considered for use in several of the company's project, however as most of the projects are deployed with fully setup localizations it has yet to be installed anywhere as its priority is low.

Thanks to this project I understand the inner workings of Magento, have a general knowledge about its structure and processes, which I now can use in different projects and tasks I am working on.

## References

BigCommerce Pricing 2020. Accessed on 28 April 2020. Retrieved from <https://www.bigcommerce.com/essentials/pricing/>

Ecommerce Comparison 2020. Accessed on 24 April 2020. Retrieved from <https://www.digitaltechnologylabs.com/2020/02/23/how-to-choose-the-right-ecommerce-platform/>

Ecommerce popularity 2020. Accessed on April 23 2020. Retrieved from <https://www.digitaltechnologylabs.com/2020/02/23/how-to-choose-the-right-ecommerce-platform/>

eMarketer Sales Information 2020. Accessed on 23 April 23 2020. Retrieved from <https://www.emarketer.com/chart/194275/retail-ecommerce-sales-worldwide-2015-2020-trillions-change-of-total-retail-sales>

Magento Module Relations 2020. Accessed on 26 April 26 2020. Retrieved from [https://devdocs.magento.com/guides/v2.3/architecture/archi\\_perspectives/components/modules/mod\\_relationships.html](https://devdocs.magento.com/guides/v2.3/architecture/archi_perspectives/components/modules/mod_relationships.html)

Magento Version Comparison 2020. Accessed on 25 April 2020. Retrieved from <https://magento.com/compare-open-source-and-magento-commerce>

Solteq Company Information 2020. Accessed on 22 April 2020. Retrieved from <https://www.solteq.com/en/services>

Solteq Oyj Website 2020. Accessed on 22 April 2020. Retrieved from <https://www.solteq.com>