DL-Tags

Grant agreement 688061

Magento integration

Deliverable ID:	**D2.2**
Deliverable Title:	**Magento integration**
Revision #:	**1.0**
Dissemination Level:	
Responsible beneficiary:	**UniZG-FER**
Contributing beneficiaries:	
Contractual date of delivery:	
Actual submission date:	**26.11.2018**

Start Date of the Project: 1 January 2016 Duration: 36 Months

Contents

Section 1 - Introduction 4

Section 2 - Magento e-commerce platform 5

Section 3 - Integration with DL-Tags 6

Section 4 - Magento configuration 9

4.1 Adding a product to the store 10

4.2 Disowning the product 12

Section 5 - Conclusions 14

Section 6 - References 14

Document history

Revision	Date	Modification	Authors
v0.1	25.10.2018	ToC	Pavle Skocir
v0.2	5.11.2018	Magento implementation (Section 4), integration with DL-Tags (Section 3)	Federico Matteo Bencic
v0.3	21.11.2018	Integration, sequence diagrams, Magento platform (Section 2)	Pavle Skocir
v1.0	26.11.2018	Additionaldescriptions (Sections 3 and 4)	Pavle Skocir

Section 1 - Introduction

DL-Tags project develops a solution for verifiable management of Smart Tags issued by TagItSmart (TIS) during the product lifecycle. Smart Tags information is shared directly between the involved stakeholders, while relying on the interoperable infrastructure provided by the TIS system and a public blockchain. Since it is based on distributed ledger, the solution is decentralized and privacy-preserving.

DL-Tags proof-of-concept solution aims to demonstrate an enhanced online trading process where physical products are being offered through an e-commerce store. E-commerce stores have become widely used for trading not only digital products such as music or books, but also for physical products. According to Statista's Digital Market Outlook, worldwide e-Commerce revenue will grow from 1,2trillion USD in 2016 to 2,1 trillion USD in 2021 [4]. One of the reasons why consumers tend to shop in offline stores instead of online is because they are concerned that the products could be different than declared [4]. DL-Tags steps into this landscape to devise a solution where product description and product lifecycle can be verified.

The stakeholders in our proof-of-concept solution are product manufacturer, e-commerce store and consumer. Product manufacturer passes the product to the e-commerce store provider. Afterwards, e-commerce store provider offers the product through itse-commerce store, making it available to a wide number of interested consumers. After purchasing the product, the e-commerce store provider initiates its transfer to the consumer's address. For the implementation of the e-commerce store we selected Magento, an open-source e-commerce platform developed in PHP programming language.

Magento is an e-commerce software platform offering e-commerce services for creating virtual stores. It is based on *software as a service* model where e-commerce services as product catalogue, consumer basket, charging, etc. are offered to e-commerce store providers to facilitate the creation and configuration of their e-commerce stores.Magento is highly adaptable and can be modified to fit a variety of project requirements.To showcase our envisioned proof-of-concept solution,Magento e-commerce store needs to be integrated with DL-Tags solution.

This deliverable describes all the necessary steps that were taken to integrate Magento platform with DL-Tags solution presented in *D2.1 DL-Tags implementation*.Magento platformserves as a stakeholder in the proof-of-concept solution demonstrating the management of the entire product lifecycle. The complete proof-of-concept solution is presented in *D3.1 Report on use case deployment and trial evaluation*.

Section 2 presents Magento e-commerce platform. Details about DL-Tags integration with Magento are described in Section 3, while Section 4 presents how the configuration of Magento and integration with DL-Tags were implemented. Concluding remarks are given in Section 5.

Section 2 - Magento e-commerce platform

Magento is an open source e-commerce platform. It provides e-commerce merchants with flexibility and control over the look, content, and functionality of their e-commerce stores [3]. The platform features marketing, search engine optimization, and product management tools, just to name a few, that give users the power to create sites tailored for their specific business needs.

Magento platform incorporates the core architecture principles of object-oriented, PHP-based applications. Prerequisites for the installation are web server (Apache and nginx), PHP extensions, and MySQL database. Magento platform is highly modular and extensible, making it easy to expand a product's feature set. Product architecture relies on Model-View-Controller (MVC) design pattern. In MVC, data that the user and platform work with are represented by the model component. The view component is used for all the user interface (UI) logic of the application. The controller component acts as an interface between models and views to process all the business logic and incoming requests. After carrying out the processing tasks, controller componentforwards the final output to the view component. View layer in Magento is referred to as *Presentation layer*, Model layer is referred to as *Domain layer*, while Controllers layer is referred to as *Service layer*. Additionally, Magento platform contains a *Persistence layer* that maps objects to database rows, communicates with the database, and performs additional business logic if necessary.

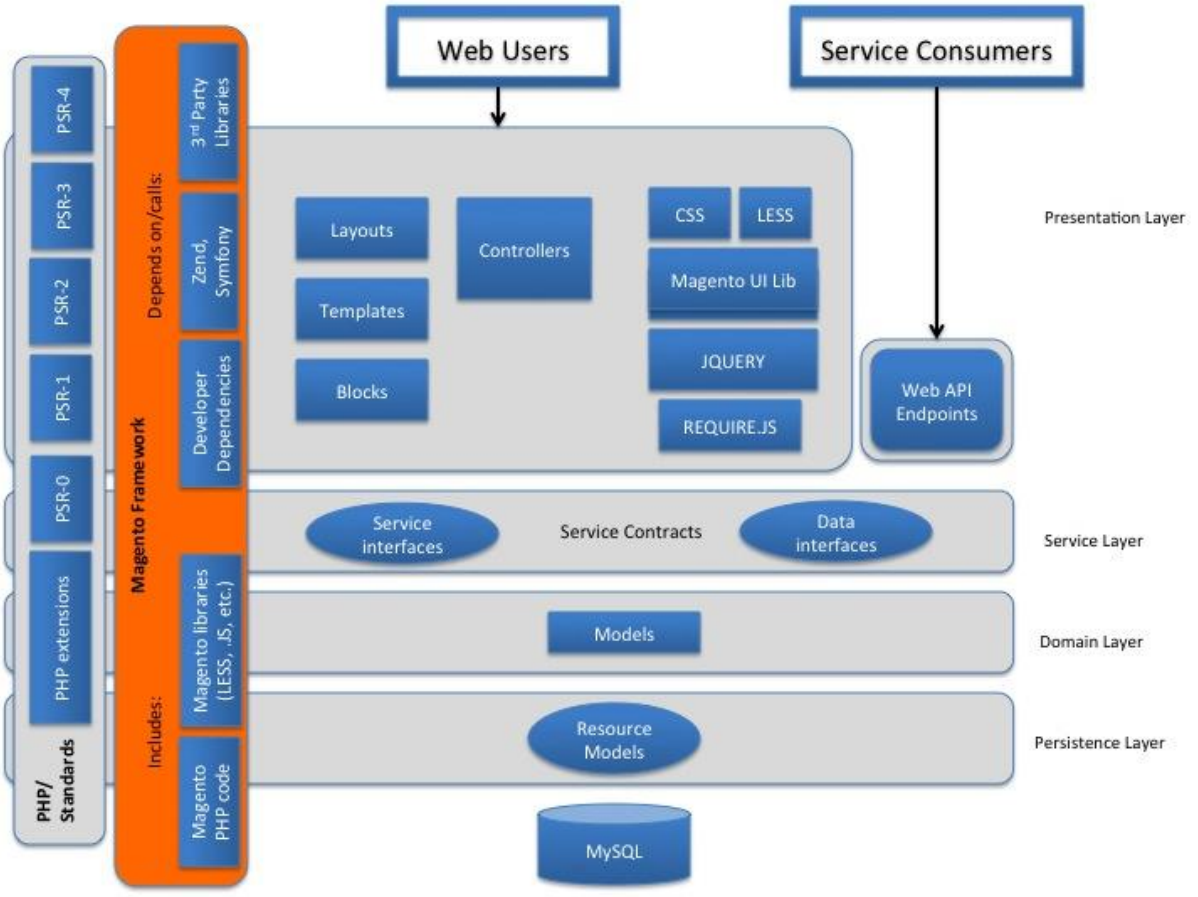

Figure 1Magento architecture layers

Magento platform architecture is shown in Figure 1. Layouts, templates, blocks, controllers, and models are some of theMagento components that are used to build custom websites, applications, and integrated systems. Components can be grouped into modules, themes, and language packages. Blocks, controllers, and models are parts of every module, while layouts and templates belong to themes which provide a consistent look and visual design for entire application area. The Magento framework controls how application components interact, including request flow, routing, indexing, caching, and exception handling. It provides services that reduce the effort of creating modules that contain business logic, contributing to the goal of both making Magento code more modular as well as decreasing dependencies.

Modularity is one of the Magento's most important features. A module is a logical group, a directory containing blocks, controllers, helpers, models that are related to a specific business feature. In order to ensure optimal modularity, a module should encapsulate one feature and should have minimal dependencies on other modules. Developers can rely on Magento Framework and its set of core logic when building modules. Namely, modules can inherit basic functions and libraries provided by the framework and developed in PHP.

The modules provided by Magentoplatform contain basic functionalities needed by an e-commerce platform. Some of the modules are: analytics, authorization, catalogue, checkout, payment, product, reports, sales, search, store, shipping, user management, etc. Within DL-Tags project a basic configuration of Magento platform is needed. Our requirements ask for the possibility to add a product to the store, authorize a seller and a user, search for products, and checkout the chosen product. The event when Magento platform receives and disowns the product needs to be recorded on blockchain by using DL-Tags system. The requirements for blockchain interaction and integration with DL-Tags are presented in Section 3. Configuration details of the Magento platform for the needed proof-of-concept solution are presented in Section 4.

Section 3 - Integration with DL-Tags

In order to use Magento e-commerce platform as a stakeholder in DL-Tags proof-of-concept solution, integration with DL-Tags components needs to be performed. DL-Tags architecture introduced in *D1.1 DL-Tags requirements and initial design*and explained in more detail in *D2.1DL-Tags implementation* is given in Figure 2for the specific case when Magento has the role of a stakeholder (*Platform*). To integrate with DL-Tags solution, Magentoplatform needs to deploy its own version of DL-Tags Proxy and send the required information to blockchain by using Proxy's JSON RPC. Additionally, Advanced Message Queuing Protocol (AMQP) needs to be deployed to enable receipt of the necessary information during execution.

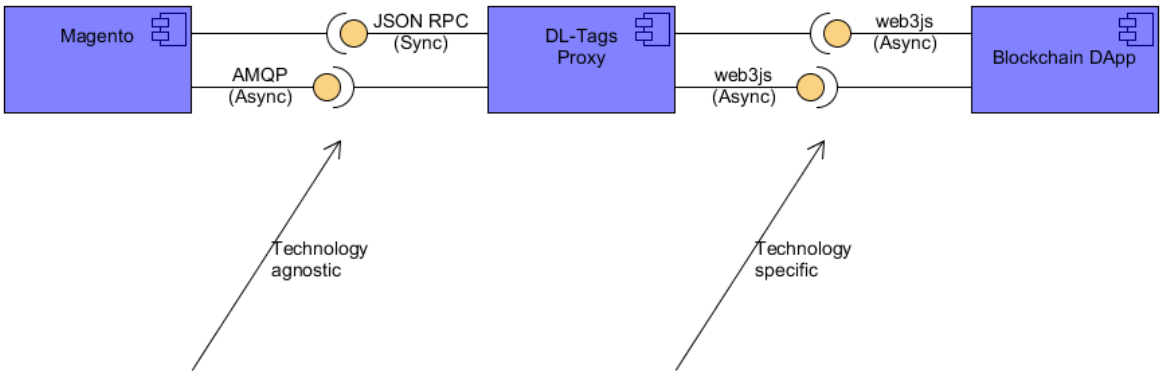

Figure 2 DL-Tags components

The envisioned scenario is as follows. A manufacturer labels its product with a Smart Tag provided by TIS. Afterwards, the product is transferred to Magentostore provider who receives information about being added as a new stakeholder and needs to verify product validity upon receipt. This verification is written to blockchain. If everything is in order with the product, Magento store provider makes the product available via e-commerce store to the users. When a consumer buys the product, Magentostore provider needs to disown the product, and pass it to the designated address. End consumers can scan the received product to ascertain themselves of its validity.

In the following part of this Section all the interactions of Magento platform with DL-Tags Proxy are described. As every other stakeholder,Magento platform needs to interact with the DL-Tags Proxy on two occasions. Firstly when receiving the product to confirm its validity, and secondly when disowning the product.

In the process of receiving the product, Magento platform isinformed about being added as a stakeholder and needs to vote on the product validity. It needs to be confirmed if the product matches its description available through Smart Tag. This action precedes product admittance to the e-commerce store and its offering to the consumers and is shown inFigure 3.

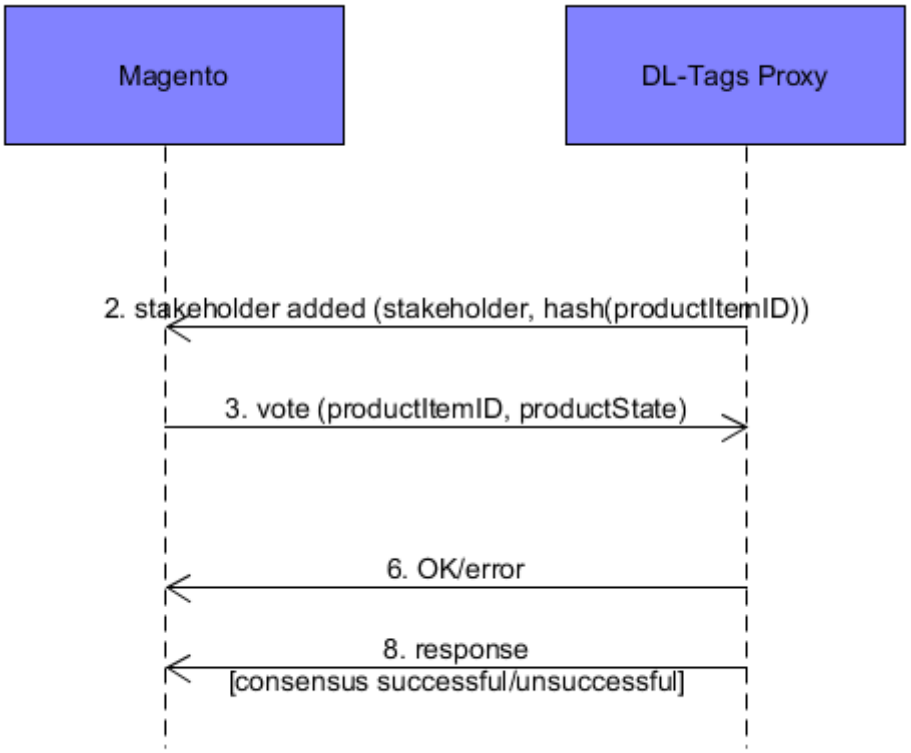

Figure 3Magento receives information about being added as a stakeholder

After a product is sold to a consumer, the Magentostore provider needs to initiate its disowning.The interactions are shown in Figure 4. The consumer will receive the specific product and will be able to check its validity, i.e., if the description provided by the TIS corresponds to the description stored on blockchain during the product lifecycle.Magentostore provider inevitably needs to disown the product. By enforcing this functionality, Magentostore provider is unable to sell the products with duplicated tags.

All the necessary steps to configure Magento platform and integrate it with DL-Tags to enable the aforementioned interactions will be described in Section 4.

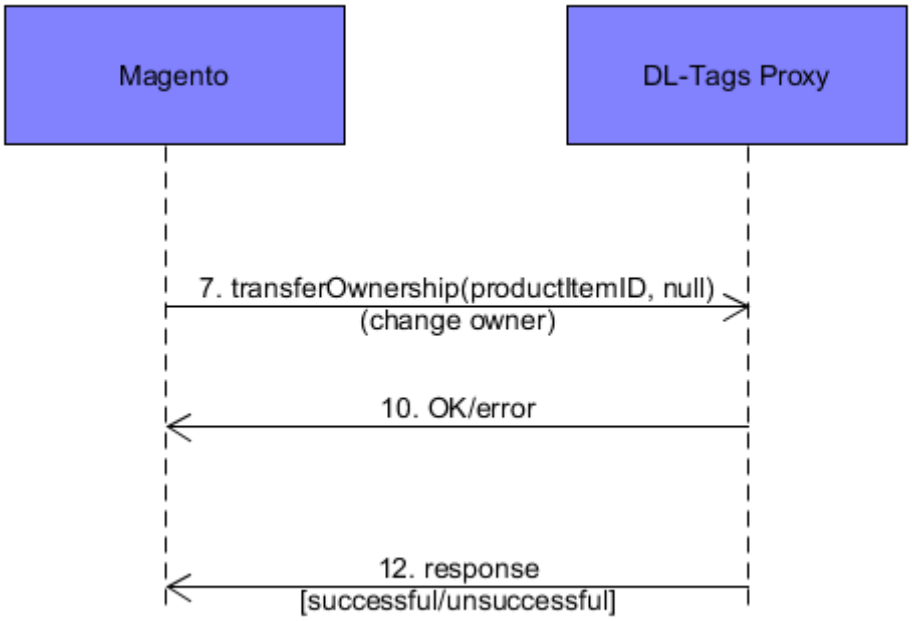

Figure 4 Adding a new stakeholder and disowning the product

Section 4 - Magento platform configuration

To accomplish the envisioned scenario, Magento platform needs to be configured and connected to the DL-Tags Proxy. AMQP listener from the DL-Tags Proxy needs to be implemented in order to receive all the DL-Tags related information. Furthermore, all the methods that need to be invoked by the stakeholder need to be implemented in the Magento platform.

Main functionalities needed for a customized Magento platform are the following:

- adding products with TIS tags to e-commerce store,
- displaying and searching the products, and
- product checkout.

These functionalities are enabled in the core Magento platform deployment and need to be configured by following the Magento installation [1] and configuration [2] guides. The functionalities offered by Magento platform and used in our proof-of-concept solution are authorization,search, store, catalog, product, checkout.

The aforementioned functionalities offered by Magento platform need to be extended in order to enable integration with DL-Tags Proxy according to specification presented in Section 3. The following subsections describe how integration with DL-Tags is implemented to enable i) receiving notifications when adding a product to the store and ii)disowning the product.

4.1 Adding a product to the store

Before adding a product to the store Magento owner needs to confirm product validity. Magento implementation offers a special interface for this purpose, shown in Figure 5. The interface is implemented as a new Magento module. After being added as a stakeholder, a Magento store provider owner needs to vote on validity of this product, as shown in Figure 3. This option is given in the “Vote” column of DL-Tags interface. Voting includes adding a product id in the input field. Therefore, a Magento store provider needs to scan the product and receive its description from TIS system. If Magento store provider concludes that everything is in order with the product, the id of the Smart Tag should be written into the “Vote” field. Upon clicking the Submit button, hash of the id is generated and sent to blockchain. If the generated hash value corresponds with the value already written on blockchain, it can be concluded that the received Smart Tag is the one forwarded by the product owner. All the previous stakeholders (in our use case these are TIS and Product owner) of the product also get the option to vote. When they agree with adding the Magento platform as the stakeholder, Magento interface receives confirmation that the consensus about the product has been reached, as shown in Figure 6.

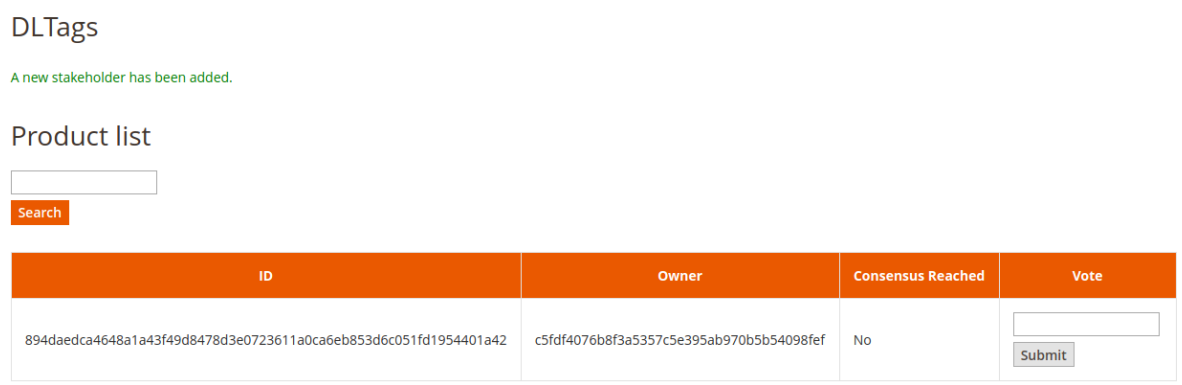

Figure 5Magento added as stakeholder

Figure 6 Consensus reached

After reaching a consensus, the product will be offered through the Magento product search module. The interface is shown in Figure 7. The users have the option to check out the

product, buy it and arrange shipping to their address. This process will be described in Section 4.2.

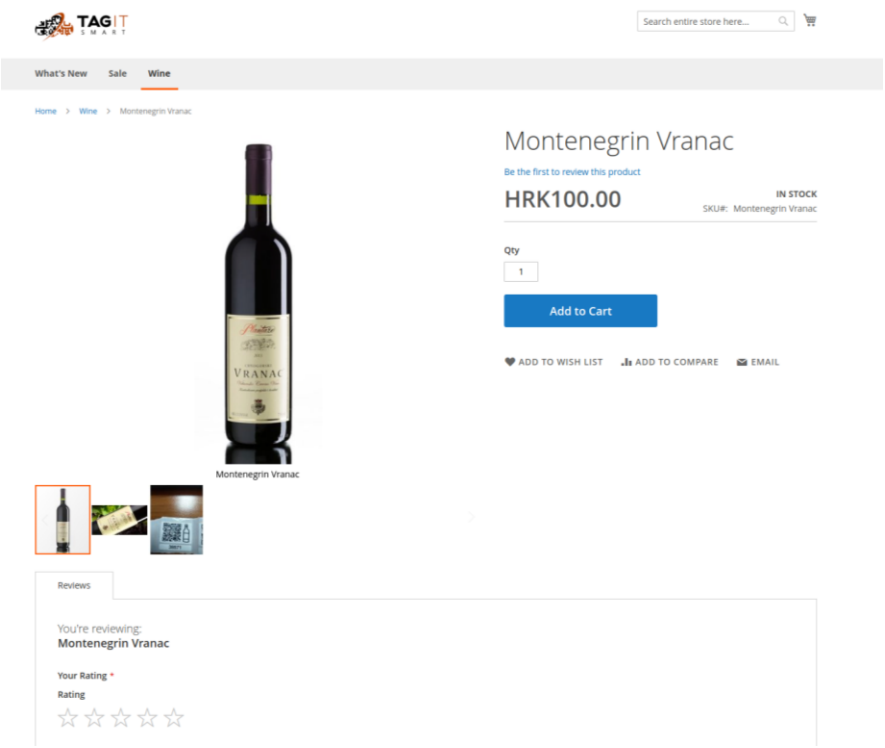

Figure 7 Magento store interface

Input to the Magento DL-Tags interface is received from blockchain, i.e., from the Magento's DL-Tags Proxy deployment. To allow receiving information from this component, AMQP protocol needs to be implemented since interaction between DL-Tags Proxy and a Platform is implemented in that way, as shown in Figure 2.

AMQP is an open standard application layer protocol for message-oriented middleware. RabbitMQ is an open source message broker software that implements AMQP that was used in implementation.Code 1shows code snippet which implements reading from the RabbitMQ.

```
try{
$connection = newAMQPStreamConnection('10.0.0.8', 5672, 'guest', 'guest');
$channel = $connection->channel();
$queueName = 'default';
$channel->queue_declare($queueName, false, false, false, false);

while($resultRabbit = $channel->basic_get($queueName)){
  ...
switch($messageBody["name"]){
case"StakeholderAdded":
    ...
case"OwnershipTransferred":
```

```
    ...
case"ConsensusReached":
    ...
case"VoteCast":
    ...
case"Error":
    ...
$channel->close();
$connection->close();

}catch(AMQPExpection$e){
...
}
```

Code 1 Configuration of RabbitMQ listener

After connecting to the RabbitMQ stream, the messages are retrieved using the *basic_get* function. The information about type of message is carried in *$messageBody["name"]* parameter. Considering what we read from that parameter, we choose the appropriate case inside the switch/case. For example, if the new product is added to the store, its value in *$messageBody* parameter would be "*StakeholderAdded*".Messages containing information about the product, owner, along with its state of consensusare stored into the database.

If parameter "name" contains the message "*ConsensusReached*", all the stakeholders for the given product have voted and a final verdict has been made. The value of the verdict (true or false, respectively) is then stored in the database. This verdict shall be changed to false every time a new stakeholder is added to the given product (i.e. the message contained the value "*StakeholderAdded*"). Once the newly added stakeholder casts a vote, another "*ConsensusReached*" message will be transmitted and its value once more stored in the database as the final vote.

4.2 Disowning the product

Disowning of the product is executed manually by the Magento store provider after a certain product has been sold. Consumer can buy product after finding it through the interface shown in Figure 7. Subsequently, shipping address needs to be inserted as shown in Figure 8, followed by payment method shown in Figure 9 and purchase confirmation shown in Figure 10. The process of disowning the product is not automatic, it needs to be invoked by Magentostore providerafter the physical product was shipped.

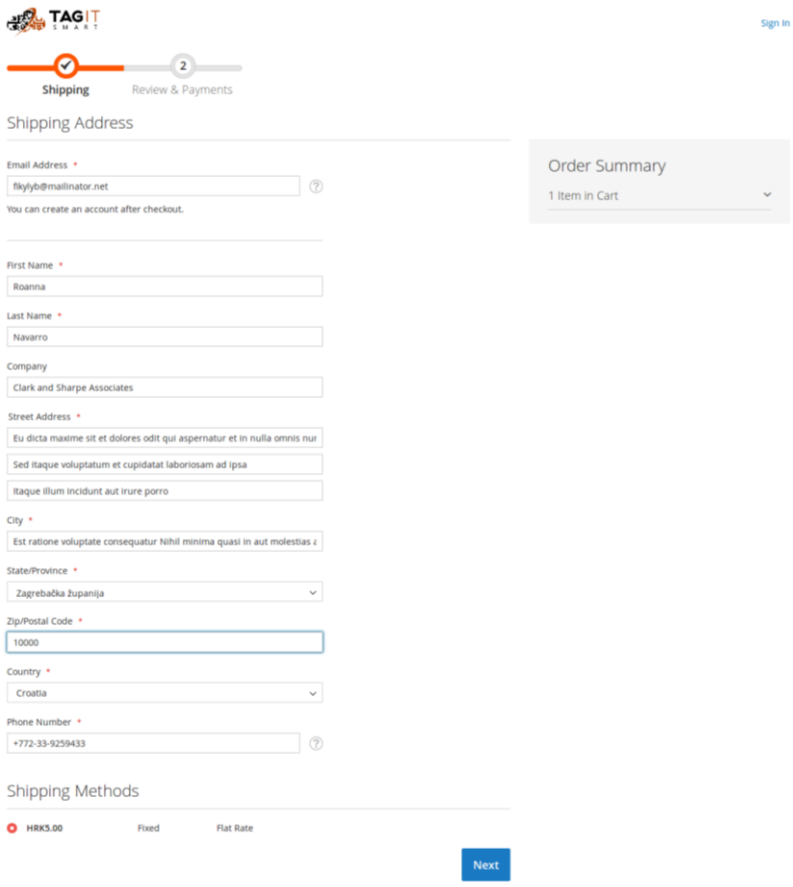

Figure 8 Shipping details

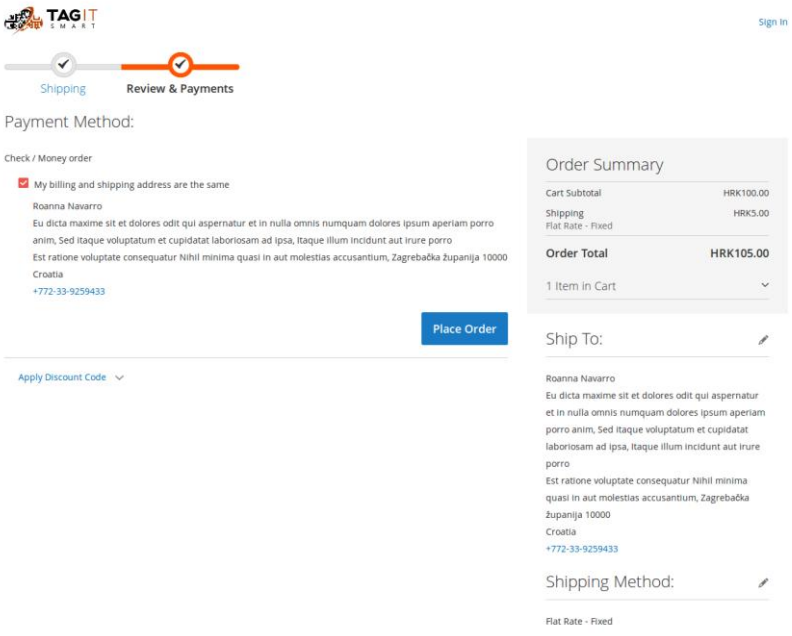

Figure 9 Payment details

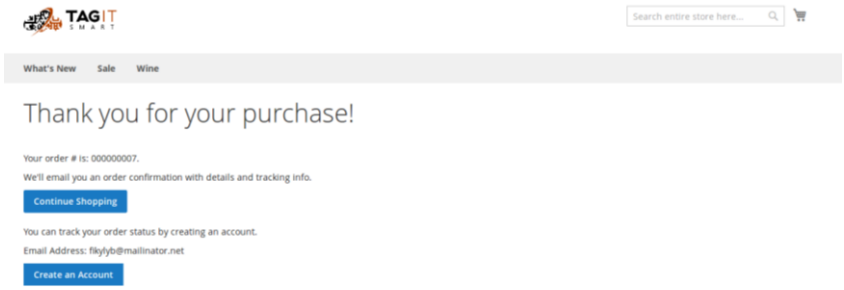

Figure 10 Purchase confirmation

Section 5 - Conclusions

This deliverable presents the integration of the Magento e-commerce platform basic deployment with the DL-Tags solution. The described solution makes Magento platform one of the DL-Tags enabled stakeholders, thus making interaction with Magento platform verifiable and privacy-preserving for both product providers and end consumers.

This integration will enable DL-Tags proof-of-concept solution showcasing a product lifecycle from the product producer via online store to the end consumer. Proof-of-concept solution is described in *D3.1 Report on use case deployment and trial evaluation.* Magento integration with DL-Tags could have a beneficial effect on user adoption of online stores since it facilitates validation of the products, thus reducing the risk of counterfeit products to be offered and sold. The solution could encourage sceptic users to frequent e-commerce stores.

Section 6 - References

[1] "Magento installation guide", https://devdocs.magento.com/guides/v2.2/install-gde/bk-install-guide.html

[2] "Magento configuration guide", https://devdocs.magento.com/guides/v2.0/config-guide/bk-config-guide.html

[3] "Magento open source user guide", https://docs.magento.com/m2/2.1/ce/user_guide/magento/magento-community-edition-2.1-user-guide.html

[4] Statista: "Offline vs. Online Retail: Development – Comparisons – Consumer View", Whitepeper, 2017