Bachelor's thesis

Information Technology

NINFOS14

2018

Nikolay Baychenko

IMPLEMENTING A MASTER/SLAVE ARCHITECTURE FOR A DATA SYNCHRONIZATION SERVICE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2018 | 28 pages

Nikolay Baychenko

IMPLEMENTING A MASTER/SLAVE ARCHITECTURE FOR A DATA SYNCHRONIZATION SERVICE

A data synchronization service called "Tehden connector" is an application developed by Codaone Ltd to synchronize data between online stores and Tehden Ltd. The application is being used by a certain number of customers and supports a variety of E-commerce platforms.

The purpose of this thesis was to implement optimization of the "Tehden connector" in a master/slave architecture to make the application more scalable, robust, and decentralized.

The application operates on the Ubuntu operating system, implemented via the PHP framework – Symfony and supported by the MySQL database management system. The optimization was conducted as a refactoring of the current application, with the addition of using HTTP as data transfer protocol and JSON as data structure for an internal communication within the master/slave structure.

The logic of the optimization is implemented in a way that instead of storing and operating all tasks on one machine, it stores settings and customer data on a master machine and the master assigns executable tasks to slave machines

The database of the application was extended with different structure both for the master and the slaves in a way that the master stores settings and connections between customers and slave, and the slaves store credentials necessary for communicating with the master.

The refracturing of the code was implemented as an addition of the current application with master/slave communication features and creation of relations between the main code and the communication features.

The tests revealed that optimization was implemented successfully and that the "Tehden connector" can work within a master/slave communication structure. However, the optimization was not released into the production environment due to the missing features that are out of the scope of the thesis.

The implementation of the optimization within the scope of this thesis has proven that the "Tehden connector" is able to work within a master/slave communication architecture. However, to provide production ready solution, more work will be performed by Codaone Oy in the future.

KEYWORDS:

Master/slave, optimization, PHP, Symfony, MySQL

CONTENTS

FIGURES

CODE SNIPPETS

LIST OF ABBREVIATIONS AND TERMS

API	Application programming interface
CUI	Console User Interface
CLI	Command-line Interface (Text mode programs (CUI: Console User Interface)" Wine User Guide. Retrieved Sep 22, 2013.)
Controller	A controller, in a computing context, is a hardware device or a software program that manages or directs the flow of data between two entities (A definition of controller by Margaret Rouse)
Entity	An entity is an object that exists and is distinguishable from other objects (Entities and Entity Sets by Osmar R. Zaï ane)
Optimization	In computer science, program optimization or software optimization is the process of modifying a software system to make some aspect of it work more efficiently or use fewer resources. In general, a computer program may be optimized so that it executes more rapidly, or is capable of operating with less memory storage or other resources, or draw less power. (Robert Sedgewick, Algorithms, 1984, p. 84)
OS	Operating system
Sync	Synchronization
PHP	Recursive acronym: Hypertext Preprocessor (PHP: What is PHP – Manual)
HTML	Hypertext Markup Language
LAMP	Linux Apache MySQL PHP
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol

UI	User Interface
UNIX	A family of general purpose, multi-user operating systems originated from the original AT&T Unix (Ritchie, D.M; Thompson, K., The UNIX Time-Sharing System, July 1978)
IP	Internet Protocol
ID	In database science, Identity Column
GET	An HTTP method request that requests a specified resource (Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim, Hypertext Transfer Protocol – HTTP/1.1, June 1999)
POST	An HTTP method request that requests a server to accept an entity enclosed in the request as a new subordinate of a web resource (Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim, Hypertext Transfer Protocol – HTTP/1.1, June 1999)

1 INTRODUCTION

1.1 Background of the topic

In computer networking, master/slave is a model for a communication protocol in which one device or process (known as the master) controls one or more other devices or processes (known as slaves) [1] (Figure 1).

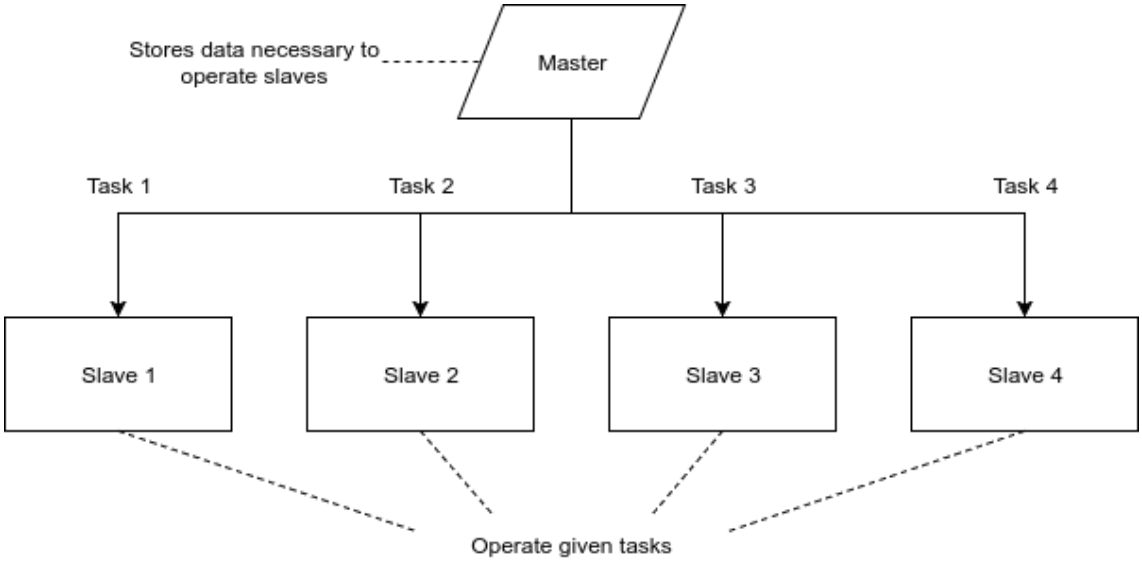

Figure 1. Master/slave basic structure.

Master/slave optimization that is being applied to a sync application (Tehden connector) is a process of refactoring of the application, changing its structure and the way of working to achieve optimization goals. The Tehden connector is an application that performs synchronization of customer data between online stores which allows these customers to reduce the manual work of synchronization of the content, product information and stocks between multiple stores as it shown in Figure 2.

The Tehden connector consists of one application which is written in PHP using the Symfony framework. The application is supported by a MySQL database and it runs on the Ubuntu operating system which allows it to run on the LAMP (Linux Apache MySQL PHP) environment, and automates the process of work using Cron.

The interface of the application is separated into 2 types. The web interface, which is accessible in an Internet browser, is mostly meant for workers of Tehden Oy who operate with customer data personally. The CLI set of commands is used for different sync operations and used mostly by Cron and developers of Codaone Oy.

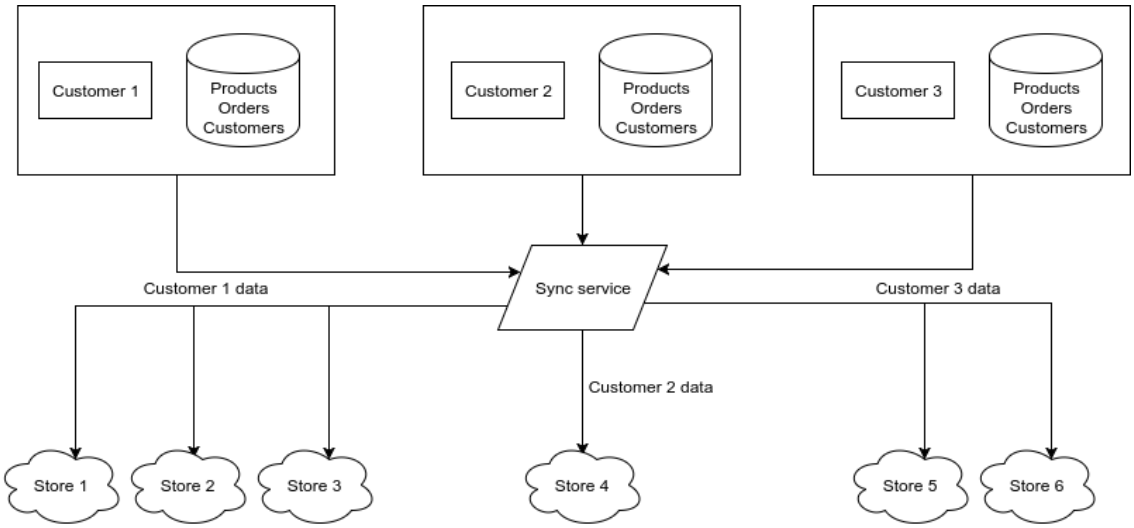

Figure 2. Structure of Tehden connector before optimization.

1.2 Statement of the problem

The Tehden connector is a complex software solution that requires a great amount of processing power to work fast. With a growing number of customers who wish to synchronize their data across online stores, it is essential that the current solution is flexible to growth. Master/slave technology model optimization will help the application to be scalable to the growing number of customers.

1.3 Scope and objectives

The objective of this thesis is to implement an optimization for the Tehden connector in a master/slave technology model. The application will be modified in a way that it could be configured to be either master or slave. The master and its slaves will be placed on different machines (physical or virtual) and work individually from each other, yet at the same rate, the slaves will be exchanging information with the master that will give them instructions to sync data.

The optimization will make the application flexible to the growth of customers and make it more secure when physical failures occur.

1.4 Justification of the project

Even though the main justification of the project is an optimization of the synchronization service that will benefit only to the companies that use it. Potentially, this thesis could be used as an example or materials for other people who would like to use master/slave technology model to optimize their application.

2 RELEVANT TECHNOLOGIES

This chapter introduces the main technologies used in this thesis project. The main technologies are: Ubuntu operating system, PHP programming language, Symfony framework, MySQL database, JSON data format.

2.1 Ubuntu operating system

Ubuntu is an open source operating system which is a Linux distribution based on Debian's architecture and infrastructure [2]. Ubuntu was chosen as the host operating system for numerous reasons. One of the reasons is that Ubuntu provides great amount of Web server software such as: Apache HTTP Server, Nginx, Varnish, Cherokee, Lighttpd and OpenLiteSpeed, however only Apache was chosen for hosting the application due to the fact that it provides great support from the community of developers and an easy configuration. Besides the fact that Linux is safe in terms of viruses [3] and stability - “these systems can easily run for months or years without the need for restart and requiring only minimal maintenance” [4], which is a good combination for a server that requires stability and security.

Another reason why Ubuntu was chosen as an OS for this solution is its job scheduler - Cron. “Cron is the name of program that enables unix users to execute commands or scripts (groups of commands) automatically at a specified time/date.” [5], which makes it a great solution for the Tehden connector to run sync scripts (both customer and master/slave related) automatically in a scheduled time.

2.2 PHP programming language

PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used Open Source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.” [6]. The PHP programming language was chosen because Tehden connector is not only a software that synchronizes customer data but also displays its interface as a web application, which can be well implemented with Symfony framework. Considering the fact that the Tehden connector combines both these functions as a synchronizer and an interface, it is the easiest option to implement

master/slave related code refactoring and additions in the same application environment. Having all the structure: entities, controllers and services implemented within one application, makes it simple to retrieve data for master/slave optimization. Other options, such as using external application, would still require creating an Application Programming Interface (API) in the current application to communicate with external one.

2.3 Symfony framework

"Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony." [7] Symfony is a tool that combines multiple PHP components into one framework. Symfony was chosen for its abilities to create entities based on table in database, functionality of controllers, services, ability to create custom APIs which would allow to establish communication exchange between master and slaves.

2.4 MySQL database

MySQL is an open-source relational database management system (RDBMS) [8]. MySQL is widely used and has a very well written documentation as well as a large community of developers who use it, which makes developing and supporting systems with MySQL easy.

MySQL integration is also available in PHP and Symfony provides a third-party library called Doctrine which is used in the application.

2.5 JSON data format

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition [9]. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal

data-interchange language [10]. In this application JSON is used as a format for data exchange between master and slaves via the HTTP protocol.

3 APPLICATION DESCRIPTION

3.1 Main idea and structure

The main idea behind the Tehden connector after master/slave optimization, as it expressed in Figure 3, is that the main application will be used both on master and slaves. However, different configurations will be applied. The master will have certain configurations that will specify its role as a master in the structure. Also, slaves, in their turn, will get certain configurations that will specify them as slaves. Customers will be interacting directly only with master and the master will store their configurations that specify stores and how to operate with these stores. A certain customer or several customers will be linked to a specific slave that will operate their synchronization. Slaves will receive configurations from the master with specifications of which customers they need to sync and how.

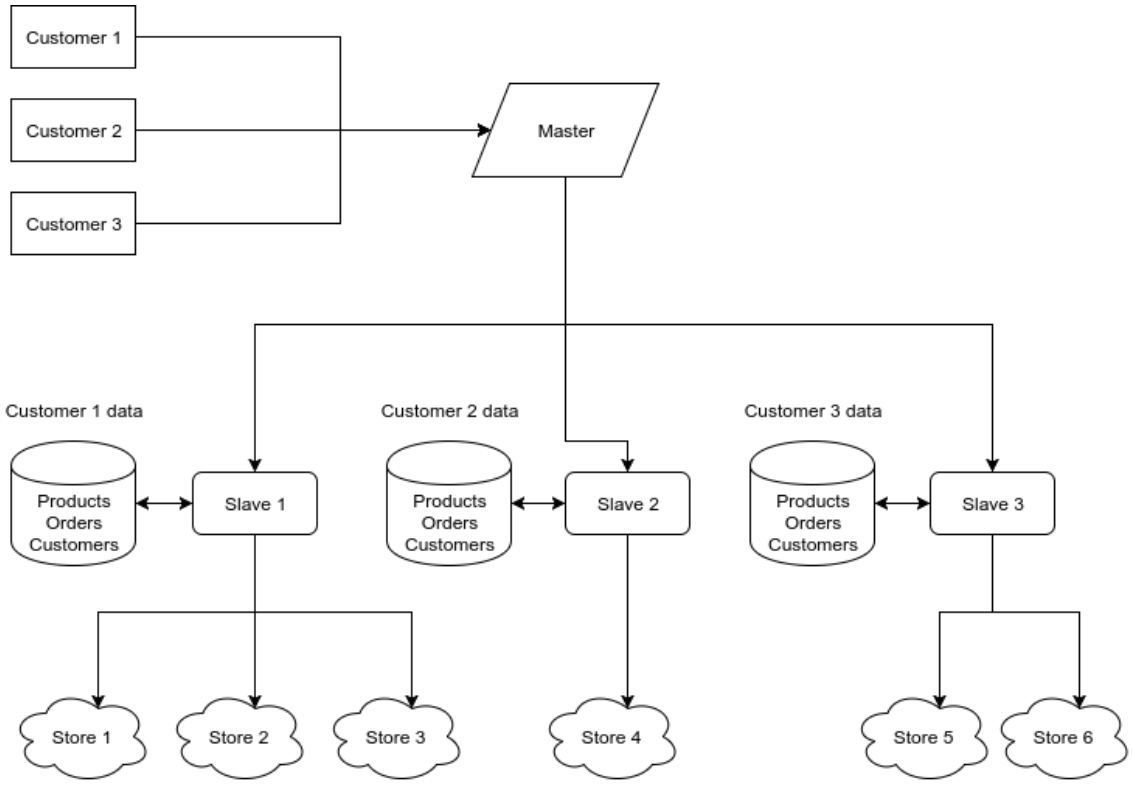

Figure 3. Structure of Tehden connector after optimization.

3.2 Logic structure and algorithm

If the master is already configured and set, the process starts with creating a new slave (Figure 4). When the new slave has been created and set up with enough information to know how to communicate with the master, it starts attempting to authorize to the master. After the authorization request has been sent, on the master's application User Interface (UI) appears a pending request from the slave. To proceed with the process, the master's user must confirm the slave from the UI manually. When the slave is not confirmed, it is sending authorization requests and receives "401 Unauthorized" responses/messages. After the slave has been confirmed, its authorization request receives response as an API token that slave will store in its database. As soon as the API token has been saved, the slave will start attempting to retrieve settings from the master. After the master accepts the request from the slave, it responds with current settings relevant to the slave.

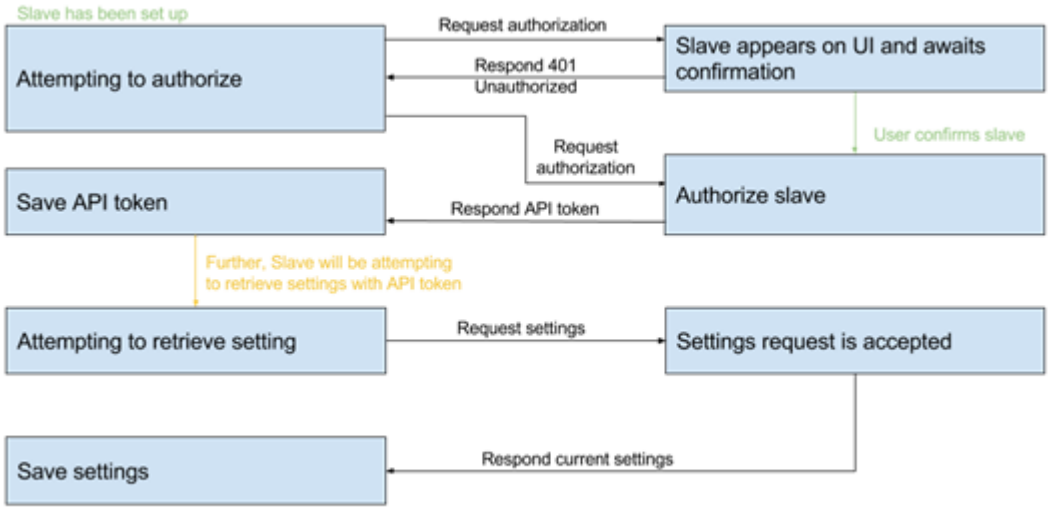

Figure 4. Logic structure of creating a new slave.

After the slave has successfully acquired settings from the master and started to sync, the following communication exchange runs on a regular basis as it shown on Figure 5. The slave checks if there are any new settings that it should receive from master. It sends a request for settings to the master, if the master performs a check if the settings for the specified slave were changed by a customer. If the settings were not changed, the master responds with a "200 OK" response, which the slave will interpret as the master

is alive, nothing was changed and continue to sync. However, if the settings were changed, the master will get new settings and send them back to the slave. On acquiring new settings, the slave will update settings that were changed and continue to sync.

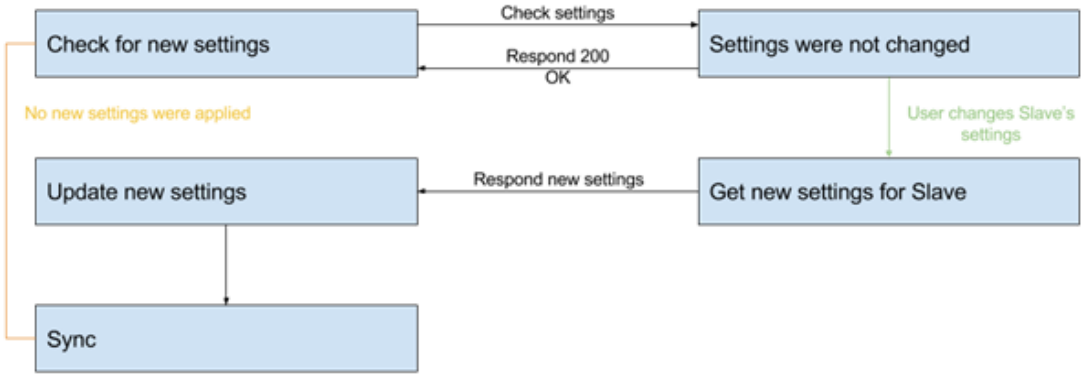

Figure 5. Logic structure of, regular communication exchange.

3.3 Master communication algorithm

From the master's point of view, a logic structure, while processing the slave's requests could be expressed in more steps as it shown on Figure 6.

After retrieving a request, the master checks if the requesting client (client as a request client) exists in the database. If the client does not exist, the master records the client's data in to the database and responds back to the slave with a "401 Unauthorized" response. When the master records a new client into the database, the record appears as a pending slave in the master's application UI, which the user can later confirm to allow the slave to authorize. This simple mechanism prevents unwanted client's access to restricted data since the user will recognize the client as slave by its information.

When the master retrieves a request and from the data from the request it can be concluded that the client is a slave that already exists in the database, the master will perform a check to determine if the slave is enabled (confirmed) by a user. If the slave is not confirmed, the master will respond to the slave with "401 Unauthorized". However, if the slave is enabled, the master will perform a check to determine if the slave's request headers contain an API token. If the API token does not exist, the master will generate it for the slave and send a response that will contain an API token, which will authorize the slave for future requests. If the slave is already authorized (headers contain an API

token), the master will check if the slave needs new settings. If the slave needs new settings, the master will return them as a response and if the slave does not need new settings the master will respond “200 OK”.

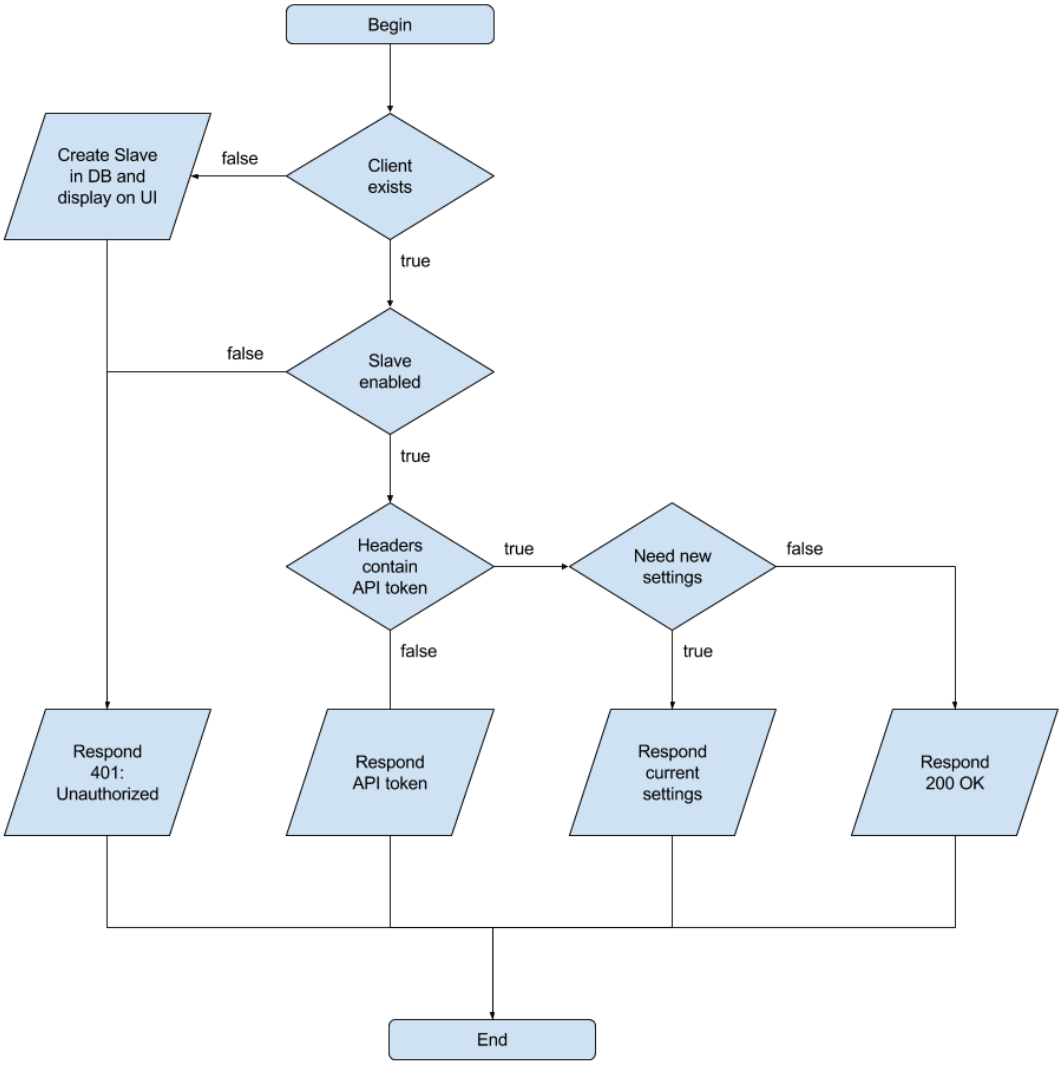

Figure 6. Master communication algorithm.

3.4 Slave communication algorithm

From the slave’s point of view, a logic structure, while attempting to get settings from the master could be expressed in more steps. In the Figure 7 algorithm shows an execution

of a Cron every time the slave runs synchronization and the algorithm determines if the synchronization should be performed or not.

First, the slave conducts a check to determine if an API token exists in its database. If the API does not exist in the database, it means that slave is not authorized and it then it performs an authorization request to the master. If, because of authorization request, an API token was retrieved, the slave will save it to the database.

If the slave is already authorized (it has an API token), the slave will check if the sync settings exist and this will help to determine if it is a new set up or an updating of settings. The only difference between having sync settings already and getting new ones is that when there are already some settings, the slave will perform a check to determine if new settings are available. If there are no new settings available, the slave will not request any settings from the master and continue performing synchronization. However, if new settings are needed or there are no settings at all, the slave will request settings from the master while also including the API token in the request's headers to authorize. If the result of the request was getting new settings, the slave will save the settings and might perform sync on its next iteration if no settings changed since the last check.

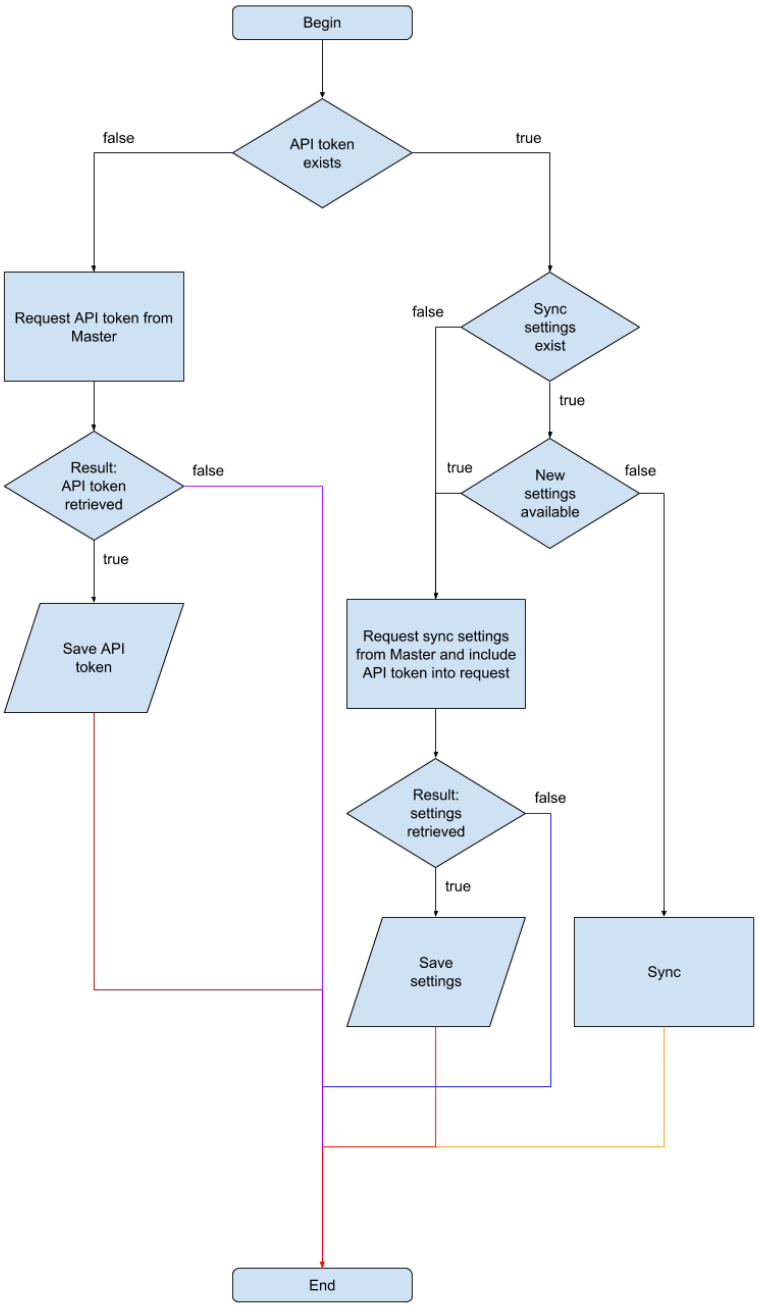

Figure 7. Slave communication algorithm.

4 DATABASE DESIGN

Both the master and slave use databases for storing data necessary for their communication. Because master and slave use the same application, their database structure is also the same even though the data is different. This chapter describes the structure of databases' tables that only involve master/slave optimization.

The table "slave" only used by master and stores data about slaves that are already confirmed and operational and those that are still pending for confirmation. The field "secret" represents a slave's name, "is_enabled" defines if slave can operate, "settings_last_update" stores a UNIX time stamp for comparison of settings between master and slave, "address" is an Internet Protocol (IP) address of a slave (Figure 8).

Column	Type
id	int(11) *Auto Increment*
api_token	varchar(255) *NULL*
secret	varchar(255) *NULL*
is_enabled	int(11)
settings_last_updated	int(11)
address	varchar(255) *NULL*

Figure 8. Database slave table structure.

The table "master" used on each slave. It stores data necessary to establish connection with the master. The slave also keeps the master's Uniform Resource Locator (URL) address in the "url" field as it is necessary for slave to know where it should forward its requests (Figure 9).

Column	Type
id	int(11) *Auto Increment*
api_token	varchar(255) *NULL*
secret	varchar(255) *NULL*
settings_last_updated	int(11)
url	varchar(255) *NULL*

Figure 9. Database master table structure.

The table “customer” is mainly used for the synchronization process. However, the field “slave” serves as reference to the slave Identity Column (ID) in table “slave”, hence the master determines which slave is used for a certain customer using this table (Figure 10).

Column	Type
id	int(11) *Auto Increment*
email	varchar(255)
name	varchar(255)
username	varchar(255)
branch_id	int(11) *NULL*
stock_id	int(11) *NULL*
send_mcf_confirmation_email	tinyint(1)
sync_products	tinyint(1)
sync_categories	tinyint(1)
sync_orders	tinyint(1)
registered_at	int(11)
create_root_category	tinyint(1)
sync_pictures	tinyint(1)
root_category_id	int(11) *NULL*
enable_syncing	tinyint(1)
locked	tinyint(1)
product_back_order_enabled	tinyint(1)
variation_back_order_enabled	tinyint(1)
discount_product_id	varchar(255) *NULL*
ignore_product_status_for_stock_update	tinyint(1)
slave_id	int(11) *NULL*

Figure 10. Database customer table structure.

5 IMPLEMENTATION

As it has been mentioned previously, the application itself can serve both as a master or a slave. The type is determined by configurations.

5.1 Slave structure

Slave structure consists of several components where a Cron schedule triggers the SlaveCommand, SlaveCommand triggers SlaveService which contains the logic for the slave's behavior related to settings and uses SlaveApiService as an interface for communication with the master.

5.1.1 Cron

On the slave side, the factor that initializes slave to sync settings with the master is Cron. Cron is set up on Linux slave machine and scheduled to run slave command every n interval. A Cron task can be set up by running the following command in a terminal: "crontab -e". After running this command, an editor with Cron tasks will be opened where tasks could be inserted. Example: "*/15 * * * * php /var/www/html/app/console slave:sync", will run slave synchronization command every 15 minutes [11].

5.1.2 SlaveCommand

The next layer of the slave's work is the SlaveCommand which is called with Cron as the "slave:sync". SlaveCommand is simply a command built within the application that reacts to CLI with combination of letters as the slave:sync. SlaveCommand, after being called, executes settings synchronization within the application's SlaveService.

5.1.3 SlaveService

SlaveService is a service of the application responsible for the slave's logic during settings synchronization. The service contains necessary methods that allow it to

validate settings with the master, retrieve it (if necessary) and update on the slave (Figure 11). To establish communication with the master, the slave uses SlaveAPIService.

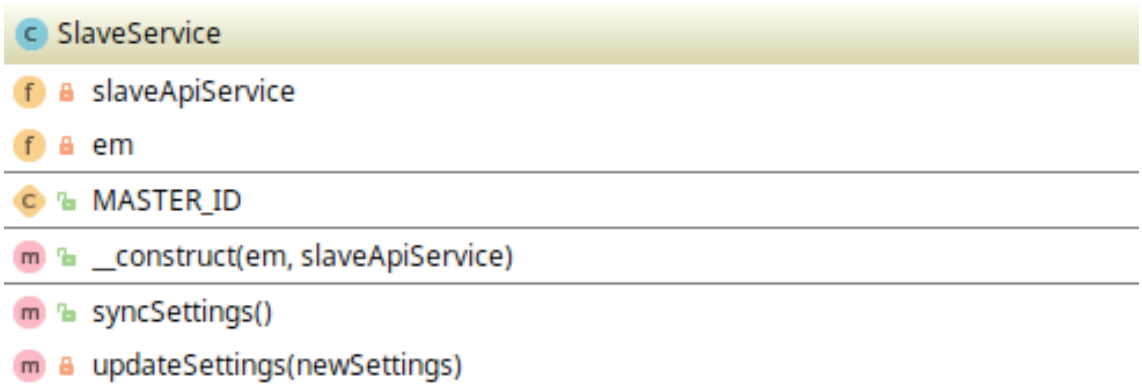

Figure 11. SlaveService structure.

slaveApiService in the scope of SlaveService is an instance of the service, which serves to call serves' methods.

em is an Entity Manager, it is part of the Doctrine (third party integration that Symfony uses to handle databases). In the scope of SlaveService, em is used to create/update entities necessary for a slave's synchronization, settings that master sends back to the slave.

MASTER_ID is simply a constant that defines master ID in a slave's database.

__**construct** function injects dependencies of em and slaveApiService.

syncSettings is a function that is being called from SlaveCommand and it's main responsibility is to validate if a slave needs new settings and act accordingly to the situation.

updateSettings is a function that is being called by syncSettings when it receives new settings from master. It updates existing settings to new settings from the master or creates new if settings do not exist on a slave.

5.1.4 SlaveAPIService

SlaveAPIService contains necessary methods that serve as a communication interface between the slave and master (Figure 12). When SlaveService executes some

operations that require data exchange with the master, it uses SlaveAPIService to generate an HTTP request understandable for the master and sends it. SlaveAPIService is responsible for operations/tasks such as: authentication process, construction of URL path, handling a request's headers, constructing and sending GET and POST requests and handling HTTP response errors.

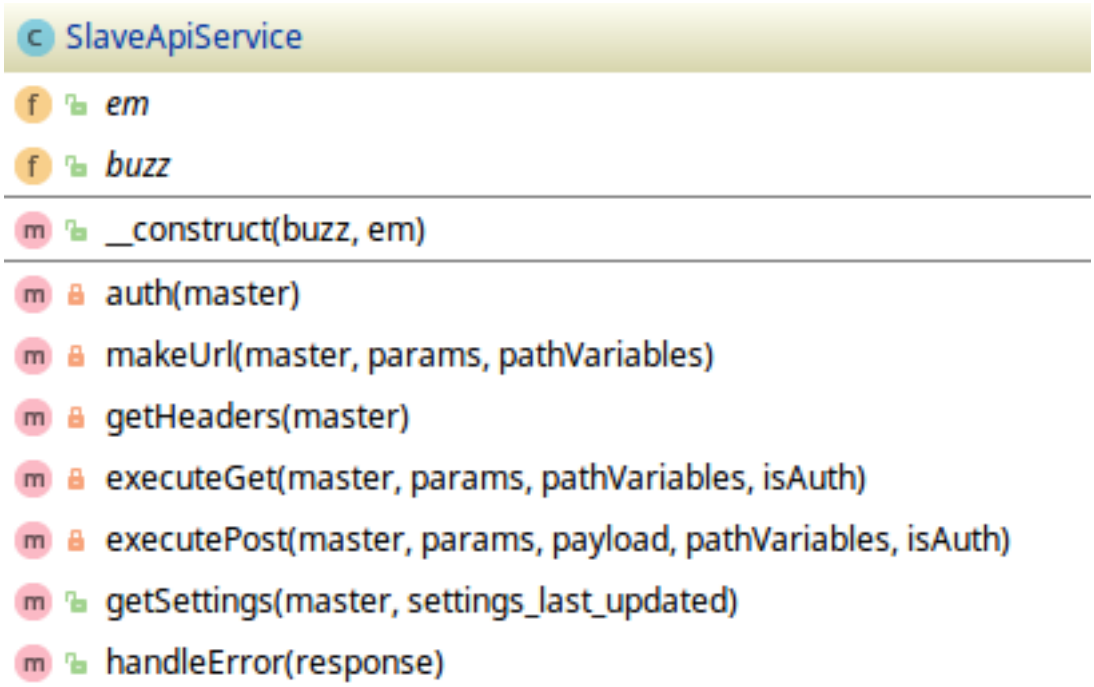

Figure 12. SlaveApiService structure.

em as in SlaveService is an Entity Manager. In SlaveApiService it is used for changing information about the master stored in the slave's database.

buzz is an HTTP client used for constructing request that are being send to the master.

__**construct** function injects buzz and em dependencies

auth is a function that is responsible for authentication of slave with the master. It checks if a slave has an API token in the database and if it does not, it sends a request for a token to the master. This function is executed every time before executing any other requests in the SlaveApiService and if authentication is not passed, other requests will not be executed.

makeUrl is a function made for convenience of creating new requests, it uses necessary URL notation to create a valid request to the master.

getHeaders is a function made for convenience of creating new requests, it creates necessary request header to create a valid request to the master.

executeGet is a function for creating GET requests.

executePost is a function for creating POST requests.

getSettings is a function that creates a request to the master that requests settings.

handleError is a function that handles response error. Every time there is an error in response, this function checks the code of an error and reports about it.

5.2 Master Structure

Master structure consists only of the MasterController which defines the logic of responses on slave's requests, generates slave's settings and creates secure API tokens.

5.2.1 MasterController

MasterController is a controller which is responsible for handling requests from slaves and act accordingly. The controller contains 2 types of methods: those that act directly from certain requests and those that work internally within the controller (Figure 13).

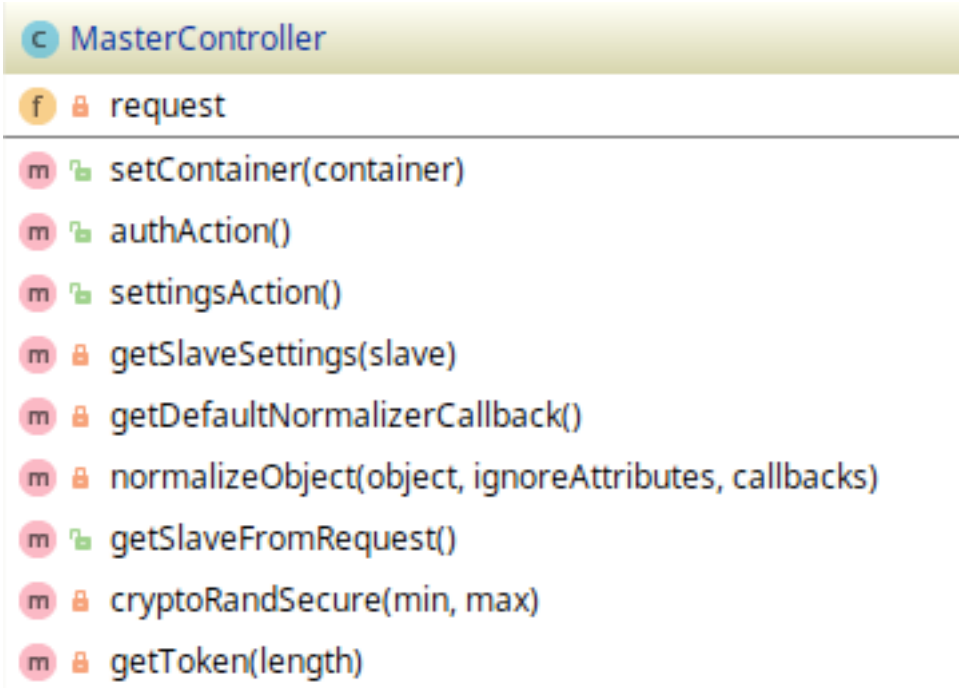

Figure 13. MasterController structure.

The methods that act directly to slaves' request would be: `authAction()` which executes when a slave request a `/auth` path in a request URL and is responsible for authentication of slave, and `settingsAction()` which responds to `/settings` path and is responsible for validation of slaves' settings and their sync (see more in Figure 6 - Master communication algorithm).

During an authentication of slave, MasterController may trigger other methods responsible for verifying a slave's IP address and generating secure tokens.

When a slave has been validated and it needs settings to return, MasterController triggers a set of methods that first retrieve necessary data from master's database, convert it into a JSON format and return it back as a request's response.

request is a part of Symfony core components, it is responsible for handling incoming requests and responding to them.

setContainer is a function that executes on every incoming request. In MasterController it is used for getting request data.

authAction is a function which executes when a slave requests "/auth" path in a request URL. It is responsible for authentication of slave.

settingsAction is a function which executes when a slave requests “/settings” path in a request URL. It is responsible for validation of slaves’ settings and their sync data (see more in Figure 6 - Master communication algorithm).

getSlaveSettings is a function which executes if validation in settingsAction passed and a slave needs new settings. The function collects all necessary data for slave’s synchronization process and returns it back. This function uses several other functions within MasterController that help it to prepare data for a request. First, it collects data from necessary entities and converts this data into an array using normalizeObject function. However, sometimes entities have references to other entities and in this case callbacks are necessary. Instead of getting a reference to another entity, getDefaultNormalizerCallback constructs a callback that will fetch referenced entity’s id.

getSlaveFromRequest is a function that collects data from slave’s request and matches it’s “secret” with data stored in the master database.

cryptoRandSecure and **getToken** are function that generate a random secure API token. Those function are using during authentication process, when the master confirms slave’s rights for authentication and grants it an API token.

6 TESTING

In the testing example, a master and a slave were created on the same machine. Sending a HTTP request will be handled manually via slave's command (which will be used by Cron in the production version).

6.1 Configuring a slave

First, it is necessary to give a slave some settings that would allow it to connect to the master (Figure 14).

Figure 14. Configuring new slave.

Only 2 values need to be set for a new slave manually: url and secret. The slave will use url as a path URL for sending requests to the master and use secret as its name. Other parameters will be initially set by default and are not necessary at this stage.

Since this is the first slave created on the master and id value is an Auto Increment, the id of the slave is 1. The next step is to assign the slave to a customer by adding **1** to **slave_id** field (Figure 15).

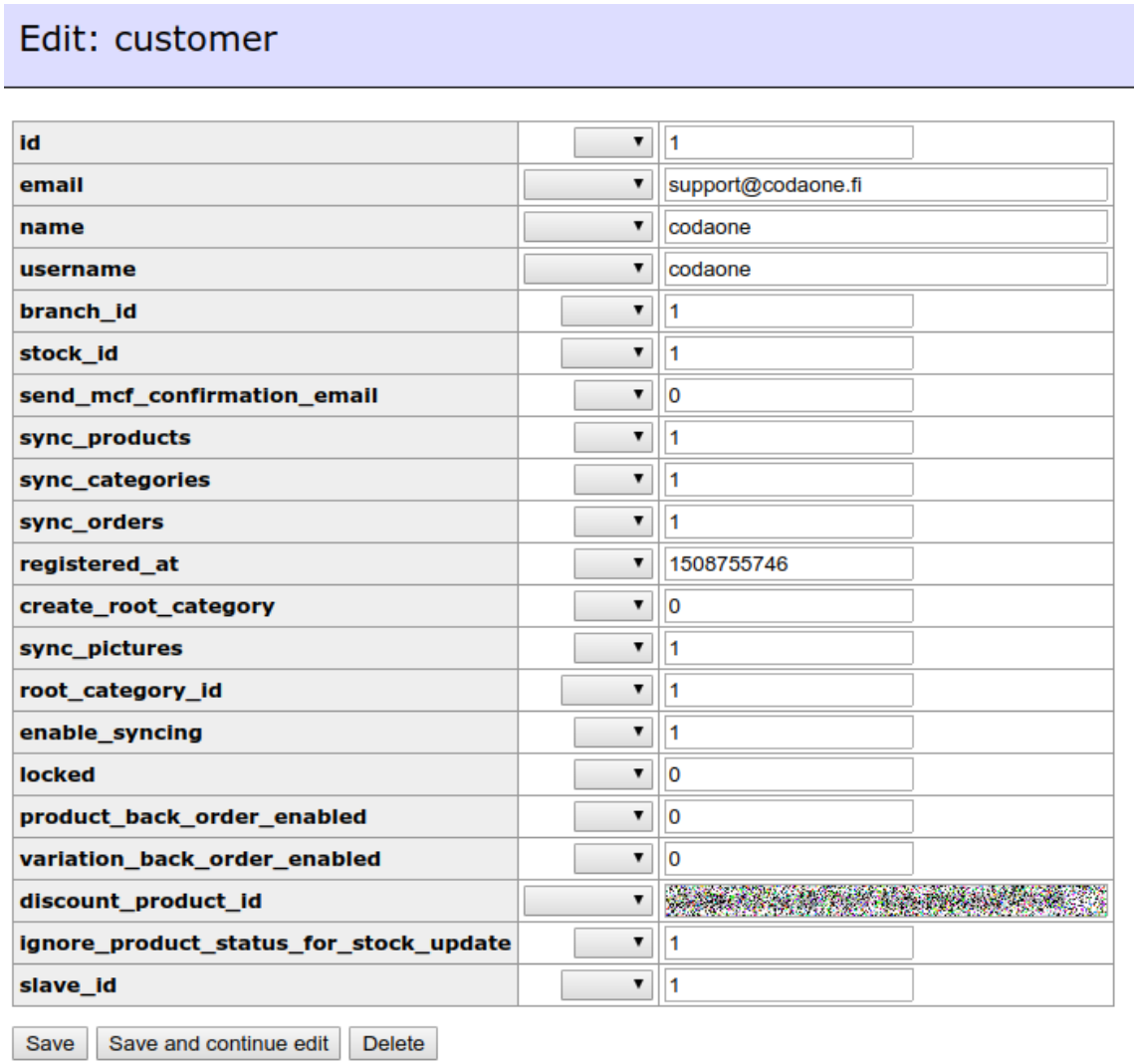

Figure 15. Add slave to a customer.

6.2 Sending first request to the master

To send a first request to the master, the following command is executed on the slave: “php app/console slave:sync”. This command triggers synchronization mechanisms, which in absence of an API key will invoke authentication process (see more in Figure 7 - Slave communication algorithm). The result of such process is: **401: Unauthorized**, which means that the slave is not yet authorized to the master. However, the master has recorded the slave’s activity (Figure 16).

id ↓	api_token ↓	secret ↓	is_enabled ↓	settings_last_updated ↓	address ↓
1	*NULL*	slave-1	0	0	127.0.0.1

Figure 16. Slave is pending for authorization.

Now the slave exists in the state of pending for authentication. In the production version, it will be scheduled to send request until it obtains a permission to authorize.

6.3 Authenticating the slave

To authenticate the slave, it is necessary to allow to sync and for that the value in the **is_enabled** field must be changed from **0** to **1**. After this is done, the next slave's request for authentication will be successful. The slave receives in a master's response to an authentication request (Code snippet 1):

Code snippet 1. Headers data of master's response for slave authentication request.

```
[0]=>string(30) "content-type: application/json"
[1]=>string(15) "Secret: slave-1"
[2]=>string(24) "Settings-Last-Updated: 0”
[3]=>string(27) "Authorization: L9VTeprTJMoM”
```

The slave notices that response's headers contain API token in Authorization field and saves it to its database (Figure 17).

id ↓	api_token ↓	secret ↓	settings_last_updated ↓	url ↓
1	L9VTeprTJMoM	slave-1	0	http://localhost/app_dev.php/master

Figure 17. Slave after authenticating to the master.

6.4 Updating settings

Until no new settings available on the master and the slave does not have any settings available, it continues to send requests to the master checking for new settings.

When new settings for the slave applied on the master, it will change **setting_last_updated** field to a timestamp when those changes were made as it shown on Figure 18.

id	api_token	secret	is_enabled	settings_last_updated	address
1	L9VTeprTJMoM	slave-1	1	1515616668	*NULL*

Figure 18. Updated slave's timestamp after changed settings.

Now, when the slave makes another request to the master, the master compares **settings_last_updated** passed by the slave and its own. In the current situation settings on the master were changed and the master prepared settings in a JSON format to respond it to the slave.

Code snippet 2. Simplified view of customer data before master sends it to slave.

```
Object

(
    [1] => Object
    (
        [customer] => Object
        [customerHasServices] => Object
        [branches] => Object
        [branchServiceIdentifiers] => Object
        [stocks] => Object
        [stockServiceIdentifiers] => Object
        [shopVersions] => Object
        [shopVersionServiceIdentifiers] => Object
        [activeShopVersions] => Object
        [shippingMethods] => Object
        [shippingMethodServiceIdentifiers] => Object
        [paymentTerms] => Object
        [paymentTermServiceIdentifiers] => Object
        [paymentMethods] => Object
        [paymentMethodServiceIdentifiers] => Object
        [shelfs] => Object
        [shelfServiceIdentifiers] => Object
        [services] => Object
        )
)
```

When the new setting was saved on the slave, it updated the **settings_last_updated** field to the same value as on master which would allow master to compare for its values for the future settings updates as it shown on Figure 19.

id	api_token	secret	settings_last_updated	url
1	L9VTeprTJMoM	slave-1	1515616668	http://localhost/app_dev.php/master

Figure 19. Slave's updated settings.

7 CONCLUSION

The main idea of the thesis was to optimize synchronization process by distributing work of one machine into several via implementing master/slave communication structure, was fulfilled. There is a master machine that contains customer data and slave's settings and then distributes synchronization work to slaves by giving them necessary settings. The application was tested in a development environment and showed positive results of work.

The application was optimized as planned, based on the master/slave structure. Master and slave applications run on Ubuntu operating system and are scheduled to work by Cron. The application itself was implemented in the PHP programming language using the Symfony framework, MySQL was used for database and JSON data format for data transfer between master and slaves. The application's logic was implemented according to the logic and communication algorithms as well as general structure design. The main challenges of the thesis were to understand the general structure of the application implementation and to understand and describe the application process in a master/slave communication structure.

The application is not yet used in the production environment due to the missing features that are out of the scope of the thesis. The application will be completed within the organization scope and then will be put to production environment. The features that will be added to the application before production: an interface that would allow monitoring and control of slaves, additional control interactions of syncing process from master on slave, automation of a slave creating process.

REFERENCES

1. *Rouse, M. (2008). What is master/slave? – Definition from WhatIs.com. [online]* searchnetworking.techtarget.com*. Available at:* http://searchnetworking.techtarget.com/definition/master-slave

   [Accessed 19 Feb. 2018].

2. *Canonical Ltd, (2018). Debian is the rock on which Ubuntu is built. [online] ubuntu.com. Available at:* https://www.ubuntu.com/community/debian

   [Accessed 19 Feb. 2018].

3. *Granneman, S. (2003). Linux vs. Windows Viruses. [online] securityfocus.com. Available at:* https://www.securityfocus.com/columnists/188

   [Accessed 24 Feb. 2018].

4. *Kleinman, S (2012). Unix Fundamentals. [online] cyborginstitute.org. Available at:* http://cyborginstitute.org/projects/administration/unix-fundamentals/

   [Accessed 24 Feb. 2018].

5. *cogNiTioN, (1999). Newbie: Intro to cron. [online] unixgeeks.org. Available at:* http://www.unixgeeks.org/security/newbie/unix/cron-1.html

   [Accessed 24 Feb. 2018].

6. *The PHP Group, (2001-2018). PHP: Preface – Manual. [online] secure.php.net. Available at:* https://secure.php.net/manual/en/preface.php

   [Accessed 24 Feb. 2018].

7. *Potencier, F. (2018). What is Symfony. [online] symphony.com. Available at:* https://symfony.com/what-is-symfony

   [Accessed 24 Feb. 2018].

8. *solid IT gmbh, (2018). MySQL System Properties. [online] db-engines.com. Available at:* https://db-engines.com/en/system/MySQL

   [Accessed 24 Feb. 2018].

9. *Ecma International, (2017). ECMAScript® 2017 Language Specification. [online] ecma-international.org. Available at:* https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

   [Accessed 24 Feb. 2018].

10. *Ecma International, (2017). Introducing JSON. [online] json.org. Available at:* https://json.org/ *[Accessed 24 Feb. 2018].*

11. *gweatherby, (2016). CronHowto – Community Help Wiki. [online] help.ubuntu.com. Available at: https://help.ubuntu.com/community/CronHowto*

    [Accessed 24 Feb. 2018].