



SAVONIA

THESIS – Bachelor's Degree Programme

Technology, Communication and Transport

DISTRIBUTED MANUFACTURING

Lucas-nulle Smart Factory

AUTHOR/S:

Xiaoyang Zhang

| | |
|--|------------------------|
| Field of Study Technology, Communication and Transport | |
| Degree Programme Degree Programme in Information Technology, Internet of Things | |
| Author(s) Xiaoyang Zhang | |
| Title of Thesis Distributed Manufacturing with Smart Factory | |
| Date 16 August 2022 | Pages/Appendices 55 |
| Client Organisation /Partners Xiaoyang Zhang, Arto Toppinen, Markku Kellomäki | |
| <p>Abstract</p> <p>This thesis's aim was to build an environment for split manufacturing that is running on two remote smart factories. Current manufacturing process is fully automated inside, so integrating it to the global internet is a very important topic nowadays. The project was made to be highly customizable, so it could be used as a base for a bigger solution.</p> <p>Modern technologies are used in the project: the communication between servers is created by an OpenCart API and the client-server connection is generated using Python requests library. In addition, a large number of original backend files of OpenCart solution that are written in the PHP programming language were modified in order to enable the API key access.</p> <p>As the result of the project, a system was created where the user selects products that smart factories need to manufacture by entering the third-party OpenCart server's web interface. After the order is completed, commands are sent from the third-party OpenCart server to the smart factories by running the Python script program. The project demonstrates an example of what could be implemented on real world manufactures in the nearest future.</p> | |
| <p>Keywords</p> <p>Distributed Manufacturing, OpenCart Server, Python requests library, PHP programming for backend files, XAMPP package, open-source cross-platform web server solution stack package</p> | |

CONTENTS

| | | |
|-----|---|----|
| 1 | INTRODUCTION | 4 |
| 2 | OPENCART SERVER FOR THE PROJECT | 5 |
| 2.1 | Introduction to Smart Factory | 5 |
| 2.2 | The gist of how Smart Factories are connected | 11 |
| 2.3 | Tosibox VPN | 13 |
| 2.4 | Operating environment | 13 |
| 2.5 | Setting for API keys environment in ERP system | 16 |
| 3 | SOFTWARE FOR THE PROJECT | 18 |
| 3.1 | Postman for API test | 18 |
| 3.2 | XAMPP package | 22 |
| 3.3 | COMMON OPENCART | 26 |
| 3.4 | COMMON OPENCART website | 29 |
| 3.5 | Get API token with Python | 39 |
| 3.6 | Python program for logical test | 40 |
| 3.7 | Extract order data from COMMON OPENCART databases | 43 |
| 3.8 | Python script | 51 |
| 4 | CONCLUSION | 52 |

1 INTRODUCTION

For a long time, many people have been worried about the assembly line work in the factory. Doing the mechanical and repetitive assembly work every day is indeed a very tedious thing. Besides, hiring manpower to manufacture products is also a huge burden on the budget.

Hence, a proposal came out: creating a Python script which is connected to the third-party OpenCart (hereafter it is named as "common OpenCart") and then the Python script will be connected to both the OpenCart server from Kuopio as well as Iisalmi. When someone makes an order through the third-party server correctly, the order will be generated right away. And then the user only needs to press the running button for the Python script and the detailed information from the order will be sent individually to a single Smart Factory or simultaneously to two factories located in different cities. The Python script gives the user the ability to make orders remotely from his home by paying a visit to the webpage corresponding to the server and easily submit his orders to the ERP (Enterprise Resource Planning) system of the Smart Factories. After the ERP system has received the order, it will send the order to the MES (Manufacturing Execution System) system and the MES system will help process the order and do the manufacturing.

The thesis will explain in detail what is Smart Factory, all software that had been used for the projects and how the principle above is implemented, including the analysis of code for the Python script.

2 OPENCART SERVER FOR THE PROJECT

2.1 Introduction to Smart Factory

Smart Factory is a series of Industry 4.0 machines controlled logically by IMS (integrated management system) and powered by Mindsphere which is a product of Siemens. And all the machinery parts are produced by Lucas-nulle company.

For the educational purpose, the products consist of three parts and they will be assembled in the following order: bottom -> top -> bolt. Before the first round of product manufacturing begins, both Kuopio and Iisalmi's Smart Factory will make a sample test run and no manufacturing activities will be carried out during this period.

The main feature of the Smart Factory is the communication between products and the machinery which manufactures them. Figure 1 shows the RFID sensor. With the help of the RFID chip, the passage of products through the MES system and every step during the manufacturing period can be easily controlled.

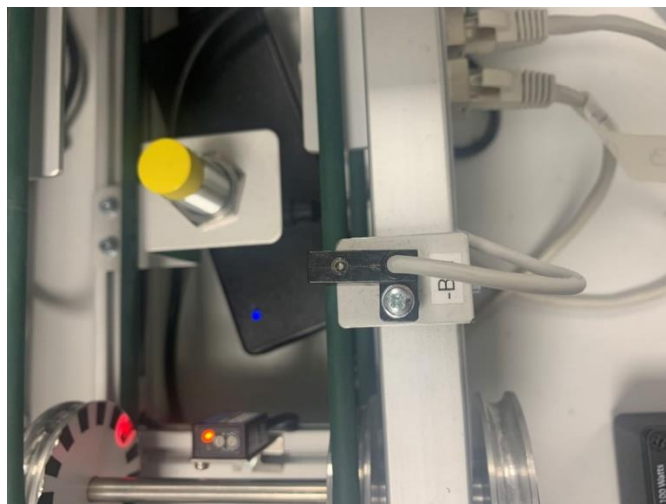


FIGURE 1. RFID sensor (Xiaoyang Zhang 2022)

When an order is generated and going to be executed, the plate with the identical stick needs to be at the RFID sensor before the first station (IMS5). Equally, the side with the identical stick must be on the sensor side. For the first round after the Smart Factory is engaged, the plate will be carried by the conveyor belt (the green bouncy rubber band) and take a tour of the Smart Factory without any manufacturing process.

Figure 2 shows the position of the start point of product manufacturing.



FIGURE 2. START POINT of IMS5 (Xiaoyang Zhang 2022)

If the plate with an identical stick has been detected by the RFID sensor, the red light flashes as Figure 3 shows. Then the order manufacturing will be executed.

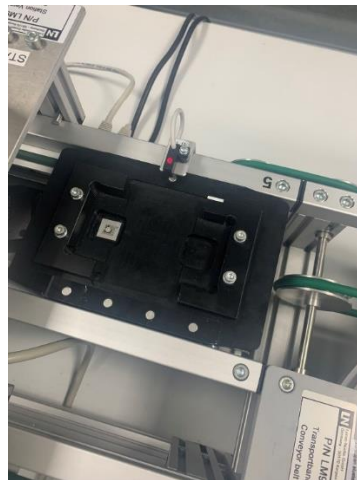


FIGURE 3. PLATE WITH STICKERS (Xiaoyang Zhang 2022)

As mentioned above, a product consists of three parts: top, bottom and bolt. Hence, there are six different machine modules which are used to manufacture the products which is commanded by the MES system. Each module is powered by a gas tank and the power will go through to each module's clamps or piston.



FIGURE 4. GAS TANK (Xiaoyang Zhang 2022)

Figure 5 to Figure10 show all sorts of stations that are mentioned above and they are divided in three different groups according to their types.



FIGURE 5. BLACK BOTTOM (Xiaoyang Zhang 2022)



FIGURE 6. WHITE BOTTOM (Xiaoyang Zhang 2022)

Figure 5 and Figure 6 show the structure of the bottom manufacturing station. These two stations are the first station to assemble the products. Both of these two stations contains a plastic clamp

and a small white roller. When the plate comes through, it will stop in place to catch the bottom part. As long as the roller moves, the clamp will open itself and help the station to push the bottom part down. After the bottom part drops on the plate, it moves forward to the next station and continues manufacturing.

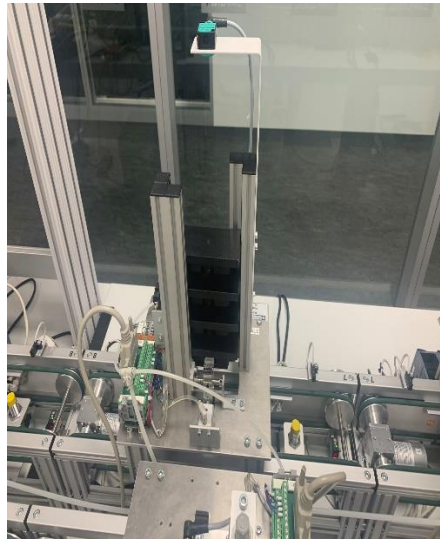


FIGURE 7. BLACK TOP (Xiaoyang Zhang 2022)

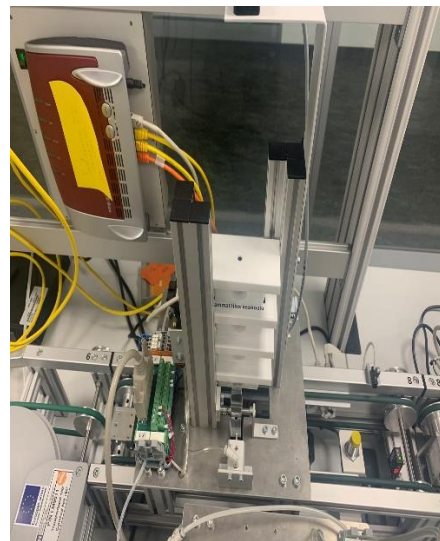


FIGURE 8. WHITE TOP (Xiaoyang Zhang 2022)

Figure 7 and Figure 8 show the structure of top manufacturing station. Their structure is similar to bottom ones. They are also managed by the clamp to control the drop of top parts. The reason why top manufacturing stations are installed after the bottom stations is because the products parts fall on the plate and the products are bottom-up manufactured. Therefore, installing top stations after bottom stations ensures smooth production of products.

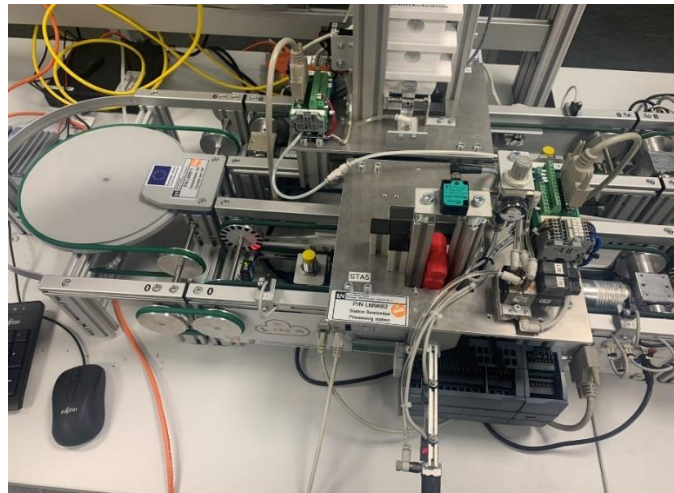


FIGURE 9. RED BOLT (Xiaoyang Zhang 2022)

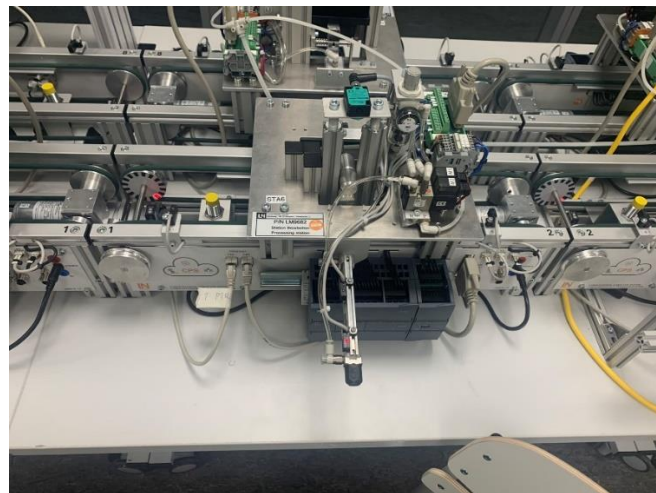


FIGURE 10. METAL BOLT (Xiaoyang Zhang 2022)

Figure 9 and Figure 10 show the structure of bolt manufacturing station. As the other four stations mentioned above, they are also powered by the gas tank. When the plate with the product reaches into this station, the bolt will fall and be at the same height as the piston. Then the piston will push the bolt part into the hole between the bottom part and top part. After that, the plate goes on to the next station.

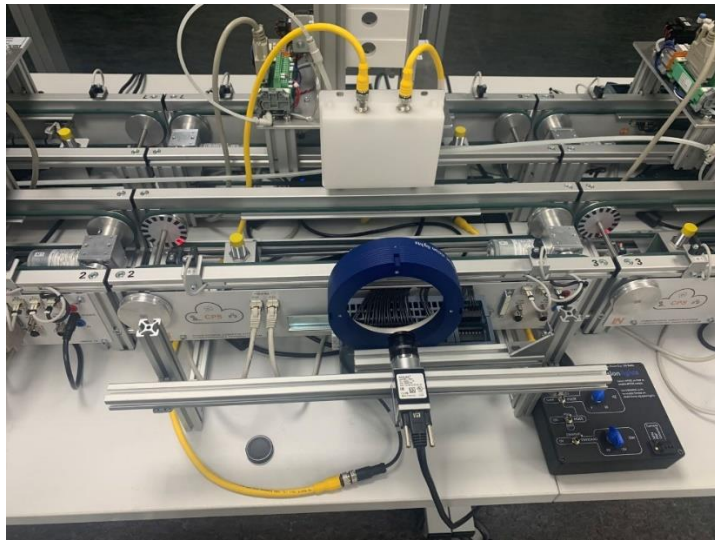


FIGURE 11. VISUAL CAMERA (Xiaoyang Zhang 2022)

After all parts of the product is assembled, it will come into the visual camera station. This station is used to verify the color of product passing through the camera while the lights in the blue circle are turned on by comparing the value collected from the product with the value in the database. The black box on the right side is the power supplier.

The user can choose whether to enable the camera function before manufacturing. If the user chooses to run without the camera, all the values that be sent to the camera will be set to None and pass this station. If the product cannot pass the station smoothly, the user can also press the button on the station to achieve manual plate positional reset himself.



FIGURE 12. MECHANICAL ARM (Xiaoyang Zhang 2022)

Figure 12 is the last station of one turn of manufacturing. As is shown in Figure 12, there is a suction cup located at the bottom of this robotic arm. When the plate with the manufactured product comes into this station, the arm will lift, suck the product up and then place the product on the left rail. The product slides along the rail to the end and finishes one round of product manufacturing.

If the user makes a new order, the Smart Factory will re-acquire new order's information and start a new round of manufacturing until it is running out of one of bottom, top or bolt part. If this situation happens, the user of the Smart Factory should refill the materials of the station so that the manufacturing will go on again.

2.2 The gist of how Smart Factories are connected

The idea of the project is to let the customers choose what kind of product they would like to have through the third-party server and make orders by running the Python program.

Figure 13 shows the main idea of the project.

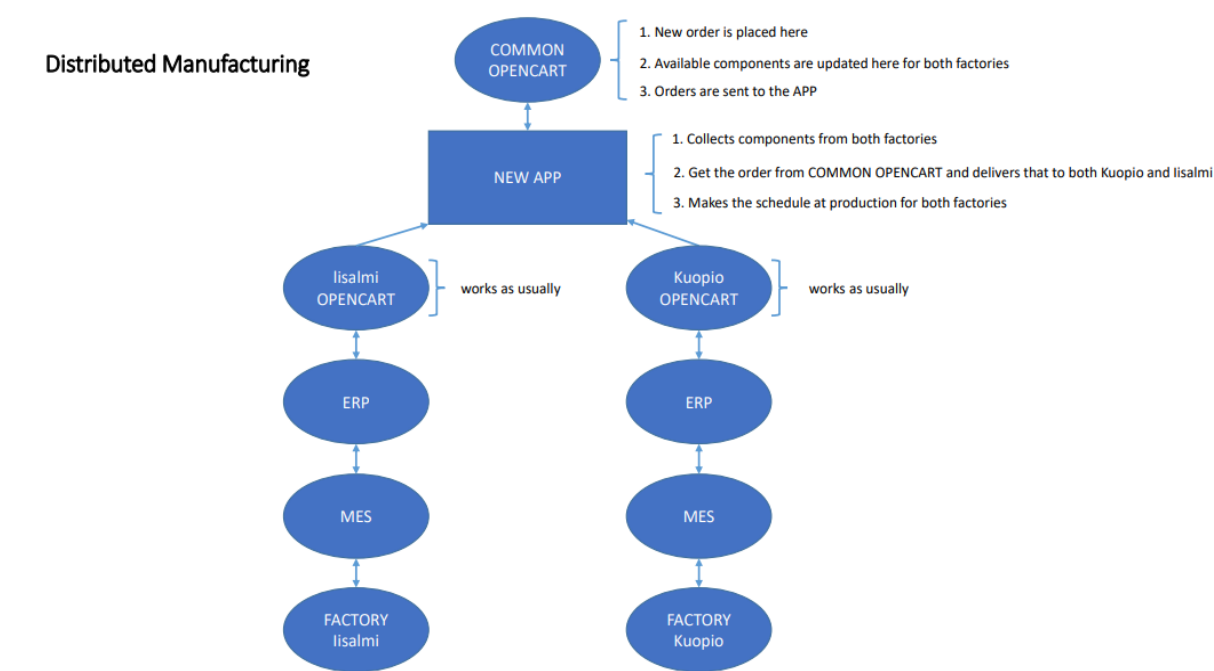


FIGURE 13. GIST OF SMART FACTORY (Xiaoyang Zhang 2022)

As Figure 13 shows, firstly the users of the Smart Factory make orders with COMMON OPENCART which is built by students attending the project. The method of building COMMON OPENCART will be mentioned in detail below.

After the orders are made properly from the COMMON OPENCART server, the customers are ready to send the orders. By running the Python script, it will extract the necessary data from the orders and deliver them to Kuopio and Iisalmi smart factory's ERP system. The connection between the COMMON OPENCART and Python script as well as the program to the smart factories' ERP system are based on the API keys. The way to generate the API keys for every OpenCart server is to get to the admin URL of the OpenCart server and find the navigation bar on the left side. Then get into System -> Users -> API and press "Generate" button to fetch the API keys. With the API keys, it is possible to establish the connection between the servers and accomplish the remote control through Python script.

Figure 14 is the screenshot from the COMMON OPENCART and it shows where exactly the place to get the API keys is.

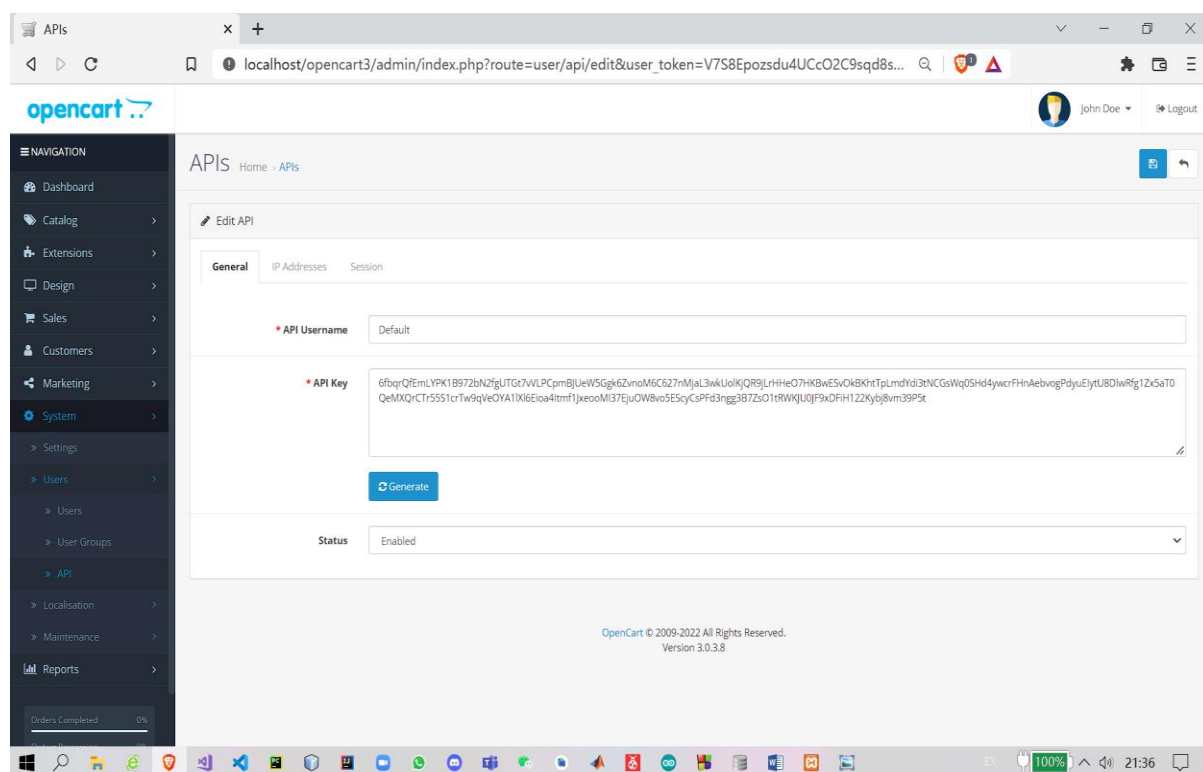


FIGURE 14. API KEYS (Xiaoyang Zhang 2022)

In addition, from Figure 14, it can be found out that the URL of COMMON OPENCART is set locally as "http://localhost/opencart3/admin". It means that currently the COMMON OPENCART holds the private local ip address and it can only be visible for the one who has installed the server configuration inside their device. Here are reasons why it is not recommended to make the ip into public. Firstly, the project is run as the educational proposal and do maintenance to the serve will be a long-term and time-consuming issue. Secondly, while opening the public network IP, it is usually necessary to rent a public domain name, upload the initial configuration, open network mapping as well as do the Intranet penetration for it. After these settings, the ip address is eventually opened to public. However, this will make the computer vulnerable to intrusion. In all, making a public ip is like broadcasting it to the public network users all over the world and they are welcomed to access the server at any time they want to. And meanwhile, the firewall or anti-virus software is not able to verify any malicious access or protect the device from computer virus attack since all access to the server is permitted. In other words, it is never a wise choice to be risky to make a public ip on a personal computer. Last but not the least, it is inevitable for the website builder to renew the domain name or server after they are expired. This is yet another financial expense and it would not stop until the server is no longer to be used.

Based on the above reasons, the default localhost is chosen to build the server of COMMON OPENCART. Later on the way to download and set the configuration of the server with it will be explained in detail.

Back to the Smart Factory, once the API keys are generated successfully, it is theoretically possible to pass one order from one OpenCart server to another. The following is the basic concept of the project: whether the factory will accept the order depends on the quantity of products ordered in the order. If the quantity of product is one, the order will be default sent to the Kuopio Smart Factory. If the

quantity of product is two, the order will be divided into 50% which means both Kuopio and Iisalmi Smart Factory will do the manufacturing which quantity is set as one. If the quantity is turned out to be bigger than two, the Python script will alarm and the order would not be sent to Smart Factory and executed. The biggest quantity value is set defaultly by the server and it is possible for the builder to change it into other integers. In this project, the default setting is used as the solution.

As the Figure 13 shows, the Python script is responsible for extracting and passing the necessary order information from COMMON OPENCART into the ERP system of the Smart Factory. After the ERP system has received the data, it will dispatch the data to the MES system. And MES system executes the manufacturing. Both Kuopio and Iisalmi's smart factory remains this series of processes.

2.3 Tosibox VPN

Figure 15 explains how the Kuopio and Iisalmi Smart Factory is linked with Python script by Tosibox VPN.

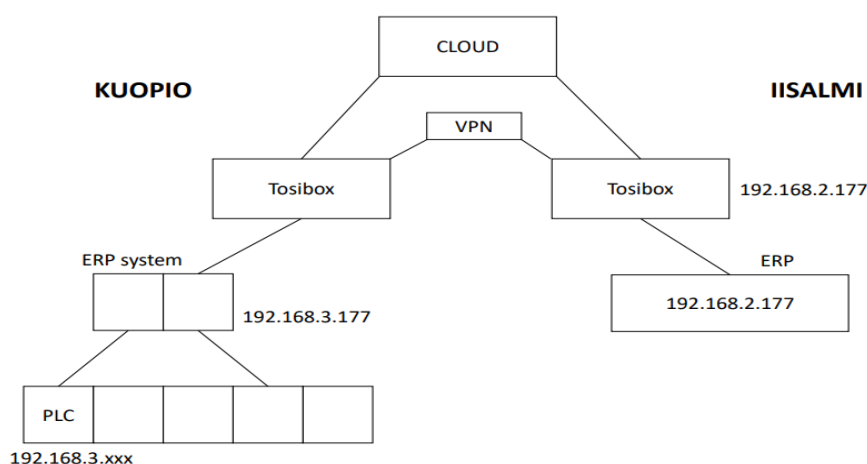


FIGURE 15. TOSIBOX VPN SETTING (Xiaoyang Zhang 2022)

As it is shown in Figure 15, both sides of the smart factory has PLC (Programmable Logic Control) to take charge of the ERP system. And both ERP systems are interfaced with Tosibox through VPN and the IP address of both factories are well-configured (Kuopio's smart factory IP is 192.168.3.177 and Iisalmi's is 192.168.2.177). Throughout VPN connection, the remote ip network communication between two Smart Factories in two different cities is accomplished.

2.4 Operating environment

Both Kuopio and Iisalmi's smart factory operating environments are installed in virtual machines. The virtual machine is Django framework which is Linux-based system. Therefore, the whole project is run in a dual-system environment.

While running the Smart Factory, it is always necessary for the user to turn on the virtual machine. The virtual machine is installed in the windows 10 system. After getting into the windows system, a software named as "ERP-lab" is able to be found on the desktop and it is the target software to engage the virtual machine.

After clicking the ERP-lab, the login page will be popped out in time. The following figure shows the UI interface of the login page:

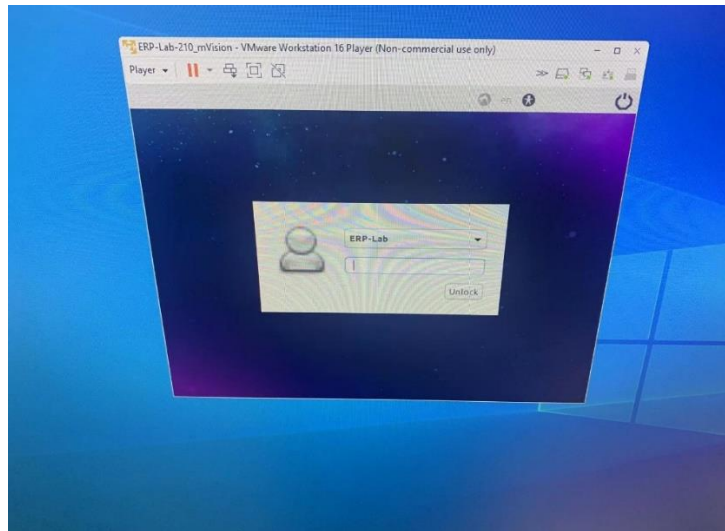


FIGURE 16. LOGIN PAGE OF VIRTUAL MACHINE (Xiaoyang Zhang 2022)

As is shown in Figure 16, the project is running the mVision project. And it requires password to be unlocked. Not only that, after inputting the password, it is also required to use encrypted key to mandate the permission to get into the Django system.



FIGURE 17. ENCRYPTED KEY (Xiaoyang Zhang 2022)

The encrypted key is the small blue USB chip which is shown in Figure 17. It is always stuck in the USB interface to let users have access to get into Django framework.

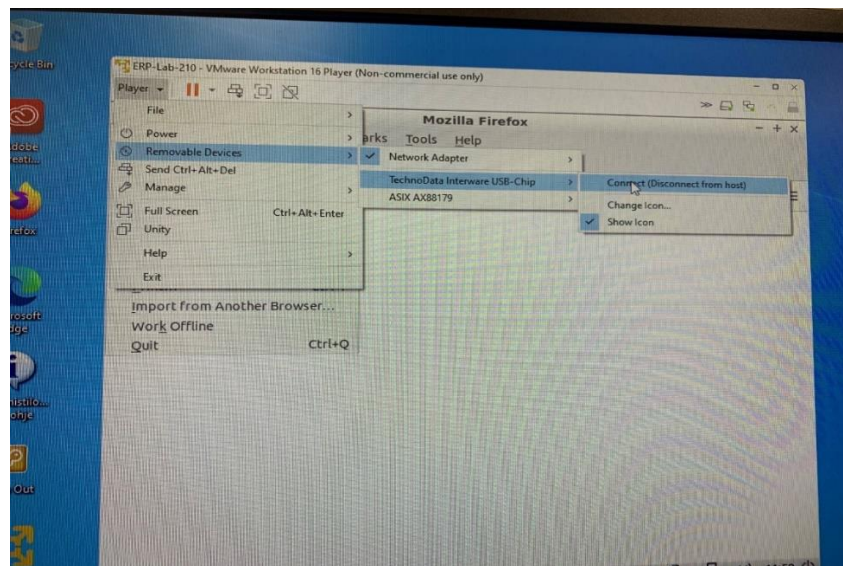


FIGURE 18. SCREENSHOT ON ACCESSING THE ENCRYPTED KEY (Xiaoyang Zhang 2022)

Figure 18 tells the exact way of accessing the encrypted key. The user needs to click the Player option on the left corner and then choose Removable Devices -> TechnoData Interware USB-chip -> Connect (Disconnect from host) in order to implement it

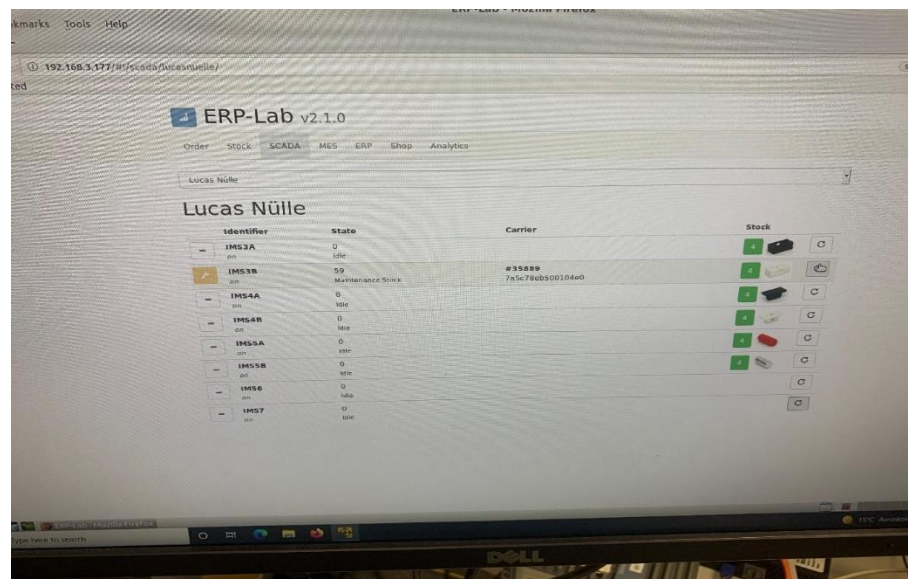


FIGURE 19. KUOPIO SMART FACTORY ERP SYSTEM (Xiaoyang Zhang 2022)

Figure 19 shows the ERP system webpage of OpenCart server from Kuopio Smart Factory. It will pop out after the user has successfully get into the Django powered web interface. As is shown in Figure 19, the ERP-lab webpage is connected together with the control panel (shown in Figure 20) and the user can easily check and adjust parameters of Smart Factories with either the ERP system or the control panel. Both the ERP system and the control panel need to be synchronized when running the Smart Factory.

Once they have lost contact with each other during the period of running the Smart Factory, the user should manually restart the whole system and repeat the steps mentioned above to re-manufacture the order once again.

2.5 Setting for API keys environment in ERP system

After the ERP system is visible, it is time to do settings for API keys which are necessary for the Python script. All servers have the same way to set an environment on API keys. In the following paragraph, an example on the COMMON OPENCART server will be set.

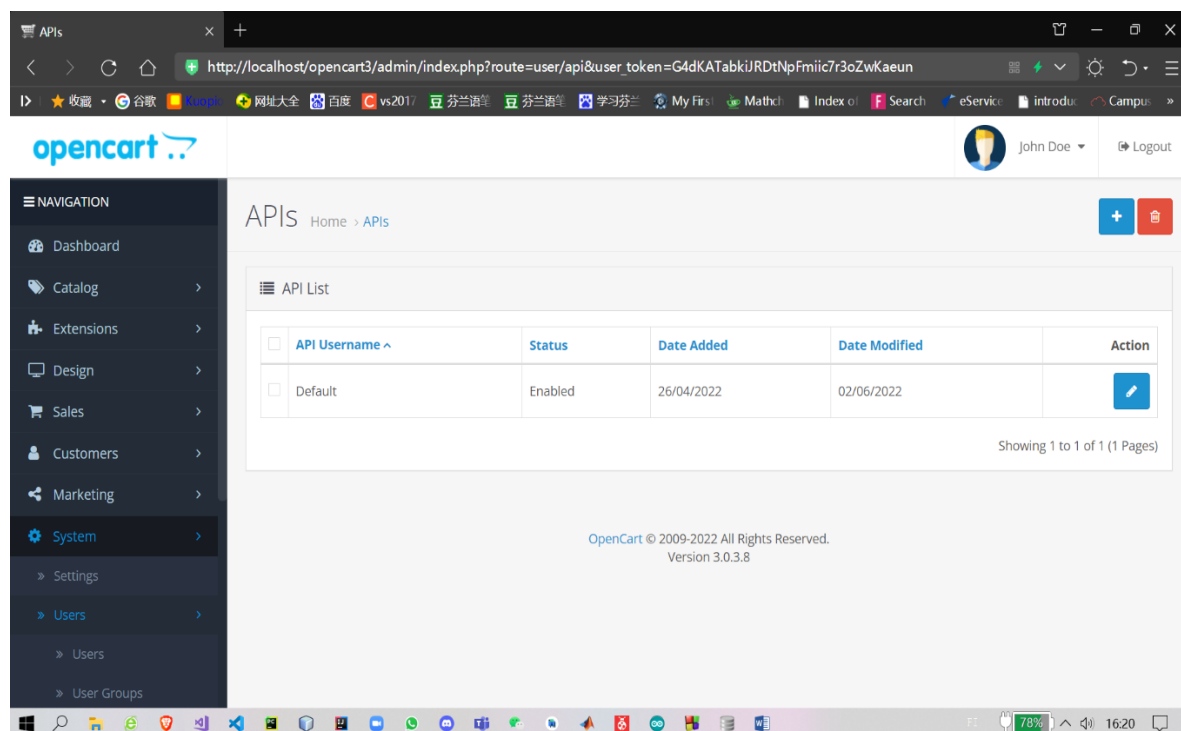


FIGURE 20. APIS (Xiaoyang Zhang 2022)

Figure 20 is a screenshot of where to set up the environment for API keys. It can be found in the path as System -> Users -> APIs. By clicking the blue add button on the right corner, the user can generate his first API named as "Default". Not only that, by clicking the "Default" API, a user can edit the configuration for it. Figures 21, 23, and 23 are showing steps of editing the API keys' settings.

Moreover, later on there will more modifications need to be done for the backend php files, and the process of programming will be elaborated when coming into that part. These steps are also important for establishing a connection between two and more servers.

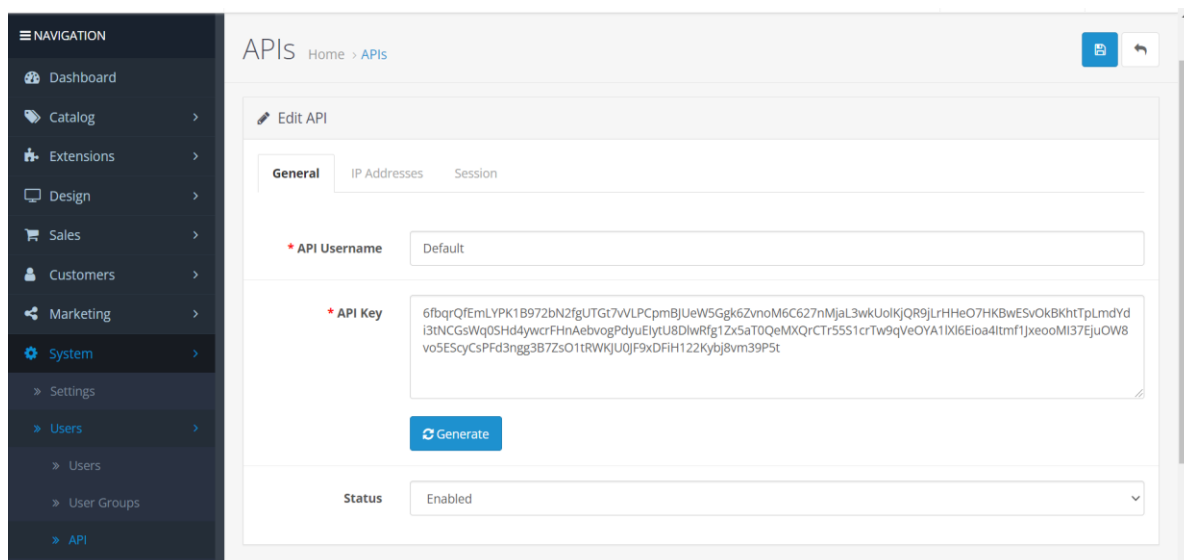


FIGURE 21: EDIT API – GENERAL

Figure 21 shows an environment where a user is able to change the API Username. When the “Generate” button is pressed, the API key is automatically generated and applied. If the “Generate” button is clicked once again, a new API key will appear and replace the old one at the same place. After renewing an API key, the old key will be made invalid immediately. In other words, a user is able to change the API key dynamically as he wants.

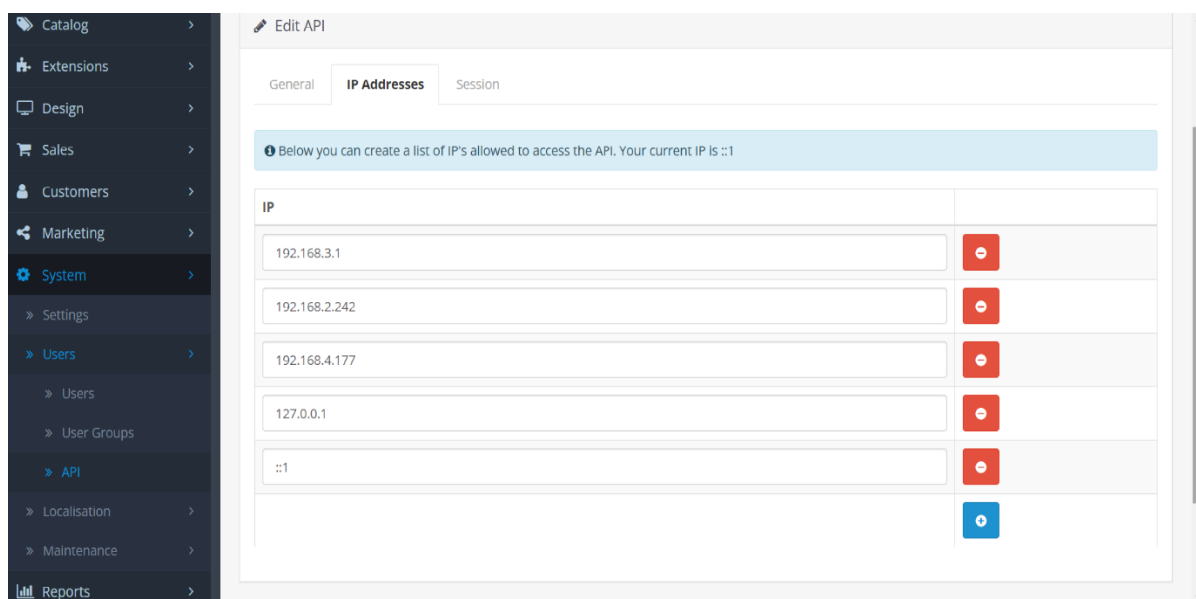


FIGURE 22: EDIT API – IP ADDRESSES (Xiaoyang Zhang 2022)

Figure 22 shows a place for the user to create a list of IPs that are allowed to access the API. Users can add IPs at will according to their needs and the list has no limits of length. Since the Python script is required to process and distribute orders between Kuopio's and Iisalmi's Smart Factories, all IPs of these three servers require to be appended to the list. In this project, all IPs that start with 192.168.3.xxx are the Kuopio Smart Factory's IPs, the ones start with 192.168.2.xxx are the Iisalmi's Smart Factory's IPs, and 127.0.0.1 as well as ::1 represents for the localhost IP which is used by the COMMON OPENCART server.

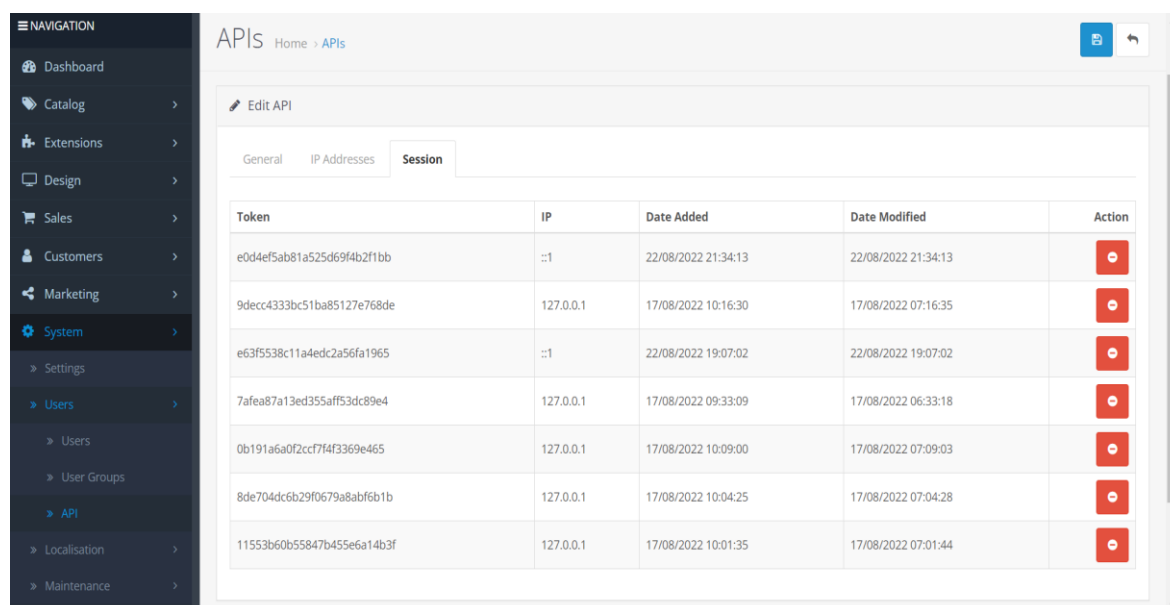


FIGURE 23: EDIT API – SESSION (Xiaoyang Zhang 2022)

Figure 23 shows the list of API tokens. These API tokens play very important role in getting a JSON response for the session between different servers. Without them, it is impossible for servers to make communications to each other. API tokens can be generated by Python's requests library, which will be explained later in this thesis.

Normally a generated API token will be valid for an hour. It is also possible to delete it manually by pressing a corresponding button. It is important to always have a valid API key to communicate with the server.

When all configurations are done, a user can save the status of the current API by pressing the save button on the right-up corner.

3 SOFTWARE FOR THE PROJECT

3.1 Postman for API test

Postman is an API platform software which is used to build and testify different APIs. And also, it helps to simplify the way of checking the API lifecycle by showing results of running an API in the terminal for the backend developer. Therefore, it is chosen to test the API for the project.

Postman is able to handle different kinds of HTTP methods. But in this project, only GET and POST methods are utilized. (Postman s.a.)

GET method is usually used for retrieving data from an API, and it should not change the state of a server. POST method is used to add new data to the server. (Deepshikha Singh, 2019) In order to make a reader understand GET and POST methods more intuitively, the demonstration of their usages in Postman will be explained in the following paragraphs.

However, it is still not sufficient to just know how to use Postman with GET and POST methods. The user also has to figure out which URLs he needs to test. Fortunately, OpenCart had published a

detailed tutorial on how to use its APIs to establish a connection between multiple OpenCart servers (Opencart s.a).

Since both the objective URLs are obtained and the way of testing API is confirmed, the user has to complete the API connection himself. An important thing is that all URLs are the key to complete the API connection, and any missing or wrong URL will cause the failure of establishing the API connection, which results in "Status: 404" response from Postman. If the connection is done correctly, Postman will give the feedback as "Status: 200 OK".

It is better to use Postman simultaneously with simulating the Python source code in the same order as an OpenCart API website (Opencart s.a.) shows since it will be much easier for the user to find out any mistakes he has made. The first thing first for the user to do is getting an API token. As it is mentioned above, an API token is used to get access to the JSON response and cookie files.

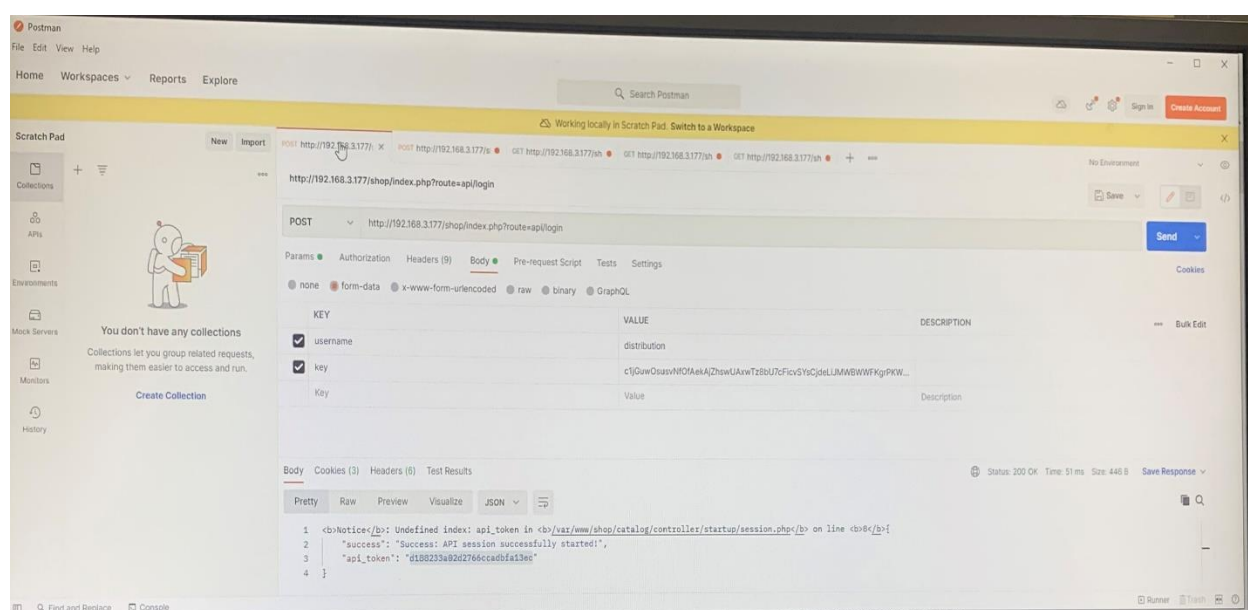


FIGURE 24: POSTMAN SCREENSHOT: FETECHING AN API TOKEN (Xiaoyang Zhang 2022)

Figure 24 shows how to use Postman to obtain an API token from the Smart Factory server. But before making the explanation of this step from API test, it is a must for the user to obtain a correct URL link for the API access (Opencart s.a.). The following Python code is used for obtaining a valid API token:

```
import requests
```

```
s = requests.Session()
```

```
username = 'Default'
```

```
key='L3MYyzlYMRL8gBcpCm6CdrVarFUXtPORZkJKP7vgaY8M8EIZWOr3EJxq'
```

```
# Actually, key is 256 character-long
```

```
s.post(
    'http://myopencart.example.com/index.php?route=api/login',
    data={'username':username, 'key':key}
).text

(Opencart s.a)
```

As it can be easily seen from the code, fetching an API token uses POST method since the ninth line of the code starts with "s.post". The reason is that a new individual token needs to be generated by a request. The tenth and eleventh lines of code illustrate what the user needs to know the URL, 'username' value and 'key' value. The URL is the address of the target link to the backend PHP file. Since the server is running on a local network, the URL should be a local IP instead of 'myopencart.example.com'. And the 'username' and 'key' values are set on the server manually. By clicking the Send button on the right side, the Postman will start working on sending the request to the server and the result can be seen in the terminal. Figure 24 shows a correct server's response.

The next step is to test an order capability of the server. For doing so, the user needs to send a request to "api/order/add" method of OpenCart API (Opencart s.a). A test request is shown in Figure 25. The terminal will pop out the following two sentences:

```
{
    "success": "Success: You have modified orders!";
    "order_id": 212
}
```

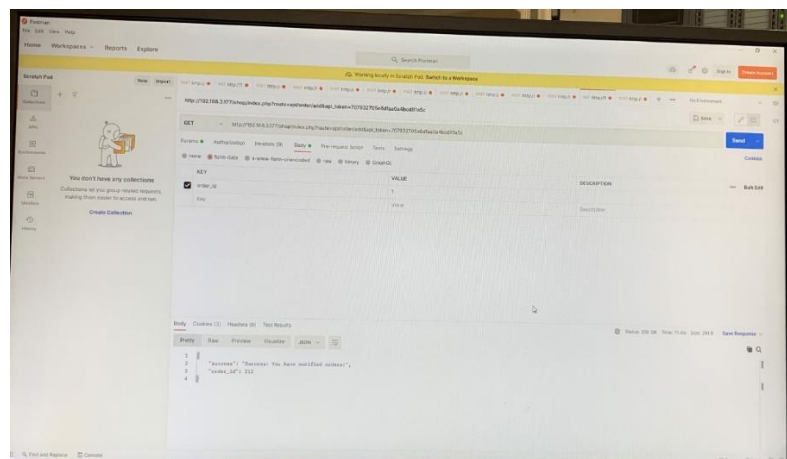


FIGURE 25: POSTMAN SCREENSHOT: API/ORDER/ADD (Xiaoyang Zhang 2022)

The server's response means that the user has made a new order whose ID is 212 and this order has been accepted by the Kuopio Smart Factory server. The order will be eventually sent to the ERP system and be executed in order by MES system. For example, if there is an order whose ID is 211, then the system will handle order 211 firstly and then order 212. This newly-built order can be checked in ERP system, in which all the orders are listed from the oldest to newest. Figure 26 shows an interface of an ERP system.

| Order ID | Customer | Status | Total | Date Added | Date Modified | Action |
|----------|--------------------|------------|--------|------------|---------------|--------|
| 212 | Firstname Lastname | Processing | 49.00K | 20/04/2022 | 20/04/2022 | + |
| 211 | Firstname Lastname | Processing | 27.00K | 20/04/2022 | 20/04/2022 | + |
| 210 | Firstname Lastname | Processing | 36.00K | 22/04/2022 | 22/04/2022 | + |
| 209 | Firstname Lastname | Processing | 27.00K | 20/04/2022 | 20/04/2022 | + |
| 208 | Firstname Lastname | Processing | 27.00K | 20/04/2022 | 20/04/2022 | + |
| 207 | Firstname Lastname | Processing | 27.00K | 20/04/2022 | 20/04/2022 | + |

FIGURE 26: ORDER LIST IN ERP SYSTEM (Xiaoyang Zhang 2022)

As long as the order has not been executed by the MES system, a user is able to delete it from the MES system to prevent this order's manufacturing. However, due to the fact that ERP systems and MES systems are not mutually updated, the order record in the MES system will still exist. Eventually the unwanted order will still be executed. So the best way of deleting an order is to manually remove it from the MES system. Figure 27 shows the user interface of the MES system.

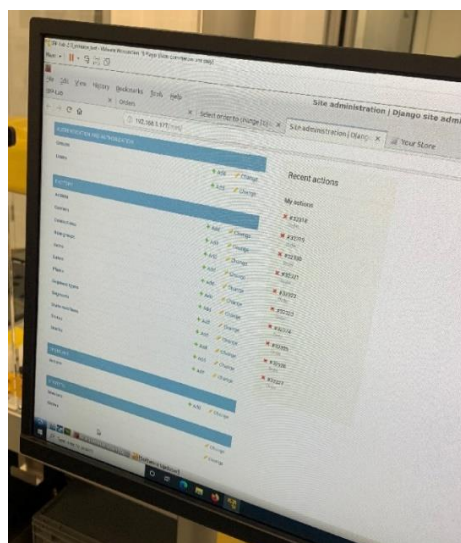


FIGURE 27: MES SYSTEM (Xiaoyang Zhang 2022)

In the MES's interface, there is one "Orders" option in the "STATISTIC" category which is located at the end of the webpage. And that is what is used to modify or delete orders from a MES system. By clicking the "order" option, the user will come into the order's webpage. Figure 28 shows a page in which the user could remove orders from the MES system. The user can choose the existing orders in the webpage and delete them by clicking an "Action" button. After the order is removed from this page, the product of this order would not be manufactured by the Smart Factory. Thus, the Smart Factory will skip the removed orders from MES system and keep manufacturing the rest of available orders until all of them are completed

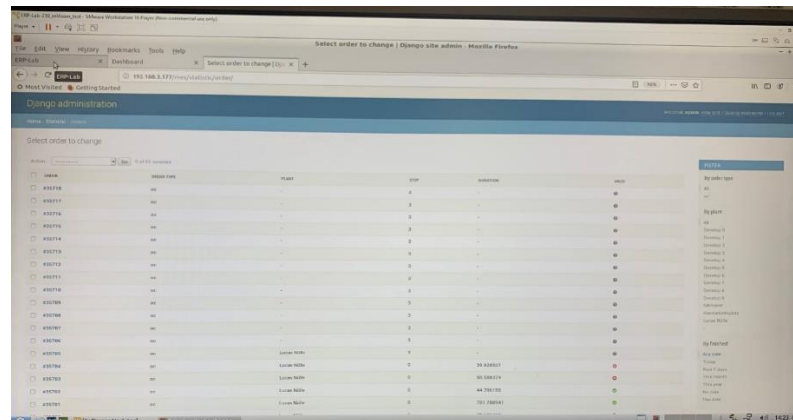


FIGURE 28: SELECT ORDER TO CHANGE (Xiaoyang Zhang 2022)

3.2 XAMPP package

XAMPP is an open-source cross-platform web server solution stack package which consists of Apache HTTP Server, MySQL database, FileZilla, Mercury, Tomcat as well as interpreters for scripts written in the PHP programming language. In this project, XAMPP package is used to build the COMMON OPEN-CART server.

Except XAMPP package, the OpenCart platform is also necessary to be downloaded. OpenCart is a website building platform which is based on MySQL as well as PHP programming language. It is possible for the OpenCart Platform user to customize his own online shop. Besides, it is extremely convenient to apply various kinds of extensions from OpenCart forum in order to improve the OpenCart webpage.

The first step of creating an OpenCart website is to download the OpenCart's source files from its official website (Thedmnyc s.a). The official OpenCart website supplies two different paths of downloading the OpenCart for the user. In this project, the locally hosted version is used to create an OpenCart website.



OR



Easy set-up by our hosting partner

[Click here to get started now](#)

- OpenCart is automatically setup for you
- Hosting optimized for OpenCart websites
- Trusted OpenCart [GOLD level partner](#)
- 99.9% server up time guarantee
- 24/7/365 excellent technical support

Download & host your own

v4.0.1.1 August 27, 2022. [Release notes](#)

- Download OpenCart 4.0.1.1 for free
- Customise your own hosting solution
- Install and configure your store
- Paid support from [approved partners](#)
- Or clone the OpenCart [GitHub repository](#)

Check out the [previous releases](#)

FIGURE 29: DOWNLOAD & HOST YOUR OWN (Thedmnyc s.a)

After the OpenCart source files have been installed, it is needed to download the XAMPP package for setting up an OpenCart server. The user can download the package from its official website as well. (Apachefriends s.a)

Download

XAMPP is an easy to install Apache distribution containing MariaDB, PHP, and Perl. Just download and start the installer. It's that easy.

XAMPP for Windows 7.4.29, 8.0.19 & 8.1.6

| Version | Checksum | Size |
|---------------------|----------|--------|
| 7.4.29 / PHP 7.4.29 | md5 sha1 | 159 Mb |
| 8.0.19 / PHP 8.0.19 | md5 sha1 | 161 Mb |
| 8.1.6 / PHP 8.1.6 | md5 sha1 | 164 Mb |

Requirements Add-ons More Downloads »

Windows XP or 2003 are not supported. You can download a compatible version of XAMPP for these platforms here.

XAMPP for Linux 7.4.29, 8.0.19 & 8.1.6

| Version | Checksum | Size |
|---------------------|----------|--------|
| 7.4.29 / PHP 7.4.29 | md5 sha1 | 163 Mb |
| 8.0.19 / PHP 8.0.19 | md5 sha1 | 162 Mb |
| 8.1.6 / PHP 8.1.6 | md5 sha1 | 162 Mb |

Requirements Add-ons More Downloads »

XAMPP for OS X 7.4.29, 8.0.19, 8.1.6, 7.4.29, 8.0.19 & 8.1.6

| Version | Checksum | Size |
|---------------------|----------|--------|
| 7.4.29 / PHP 7.4.29 | md5 sha1 | 163 Mb |
| 8.0.19 / PHP 8.0.19 | md5 sha1 | 162 Mb |

FIGURE 30: OFFICIAL XAMPP WEBSITE SCREENSHOT (Xiaoyang Zhang 2022)

Figure 30 shows different download options from an official XAMPP webpage. There are different sorts of installers depending on the operating system the users are using right now. In order not to miss any new features and bug fixes, the user needs to make sure to only download the latest version of XAMPP, no matter what OS he has got. Actually, this text is wrong. When installing OpenCart 3.0.3.8 with XAMPP package 8.1.2, this problem might come out firstly to the download page:

Unable to install Opencart

Problem:

SyntaxError: Unexpected token < in JSON at position 0 OK

Fatal error: Maximum execution time of 30 seconds exceeded in C:\xampp\htdocs\shop\upload\system\library\db\mysqli.php on line 18

This error is aroused by insufficient maximum execution time parameter in php.ini. To avoid this problem, the user can change this parameter into 300 seconds and then restart the installation once again (gigapro, 2018). In this project, the php.ini file shall be found under this path: "D:\xampp\htdocs\opencart3".

However, after this problem is solved, the installation still doesn't go fluently after restarting the downloading and OpenCart pre-installation page still complaining about some missing configuration files. Figure 30 shows correct settings for the OpenCart pre-installation page.

The screenshot shows the OpenCart 3.0.4 Pre-Installation configuration page. It is divided into several sections, each with a table of settings and their status.

1. Please configure your PHP settings to match requirements listed below.

| PHP Settings | Current Settings | Required Settings | Status |
|--------------------|-------------------|-------------------|--------|
| PHP Version | 7.0.0-1-00000-0-1 | 5.4+ | OK |
| Register Globals | Off | Off | OK |
| Magic Quotes GPC | Off | Off | OK |
| File Uploads | On | On | OK |
| Session Auto Start | Off | Off | OK |

2. Please make sure the PHP extensions listed below are installed.

| Extension Settings | Current Settings | Required Settings | Status |
|--------------------|------------------|-------------------|--------|
| Database | On | On | OK |
| GD | On | On | OK |
| cURL | On | On | OK |
| OpenSSL | On | On | OK |
| Zip | On | On | OK |

3. Please make sure you have set the correct permissions on the files list below.

| Files | Status |
|---|----------|
| /home/191647.douglasw@ps.com/xampp/public_html/config.php | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/admin/config.php | Writable |

4. Please make sure you have set the correct permissions on the directories list below.

| Directories | Status |
|--|----------|
| /home/191647.douglasw@ps.com/xampp/public_html/image/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/image/cache/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/image/cache/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/system/extension/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/system/extension/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/system/extension/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/system/extension/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/system/extension/ | Writable |
| /home/191647.douglasw@ps.com/xampp/public_html/system/extension/ | Writable |

At the bottom of the page, there are buttons for "BACK" and "CONTINUE".

FIGURE 30: SCREENSHOT OF SUCCESSFUL OPENCART SERVER CONFIGURATION (Hamza Zia, 2021)

By default, GD extension's status is wrong and both two config.php files are not writable. These problems are aroused by both XAMPP packages and OpenCart source files, so the user needs to modify some of them. First of all, the user needs to rename **config-dist.php** to **config.php** which can be found in the Upload directory. In this project, these two config-dist.php files are respectively from "D:\xampp\htdocs\opencart3" and "D:\xampp\htdocs\opencart3\admin". The user merely requires to rename them and one of the existing problems is solved.

2/4 Pre-Installation
Check your server is set-up correctly

1. Please configure your PHP settings to match requirements listed below.

| PHP Settings | Current Settings | Required Settings | Status |
|--------------------|------------------|-------------------|--------|
| PHP Version | 8.1.5 | 7.3+ | ✓ |
| Register Globals | Off | Off | ✓ |
| Magic Quotes GPC | Off | Off | ✓ |
| File Uploads | On | On | ✓ |
| Session Auto Start | Off | Off | ✓ |

2. Please make sure the PHP extensions listed below are installed.

| Extension Settings | Current Settings | Required Settings | Status |
|--------------------|------------------|-------------------|--------|
| Database | On | On | ✓ |
| GD | Off | On | ✗ |
| CURL | On | On | ✓ |
| OpenSSL | On | On | ✓ |
| ZIP | On | On | ✓ |

4. Please make sure you have set the correct permissions on the files list below.

| Files | Status |
|--|----------|
| C:\xampp\htdocs\opencart3\config.php | Writable |
| C:\xampp\htdocs\opencart3\admin\config.php | Writable |

FIGURE 31: AFTER THE CONFIG.PHP PROBLEM IS SOLVED (Xiaoyang Zhang 2022)

Figure 31 shows the status of pre-installation menu after the config.php files are renamed. And there is still one problem to be solved. The next step is going to be solving a problem of GD extension.

To solve the GD extension problem, the user has to firstly check if the "php_gd2.dll" file is existing in the following path: "C:\xampp\php\ext". Due to the fact that earlier OpenCart versions did not have this .dll file, the tutorial (Hamza Zia, 2021) has attached the link for downloading the GD extension inside it. After the file is checked to be at the right directory, it is needed for the user to edit the "php.ini" file which can be found in this path: "D:\xampp\php". The user needs to add "extension=gd2" manually under the "Dynamic Extensions" inside "php.ini" file. (Codeanddeploy, 2021)

If the modifications above take effect, the user is allowed to get into the next page. Figure 23 shows a configuration page for XAMPP installation. The user is able to enter MySQL database connection details from phpMyAdmin as well as the username, password and email address.

3/4 Configuration
Enter your database and administration details

1. Please enter your database connection details.

DB Driver:

* Hostname:

* Username:

Password:

* Database:

* Port:

Prefix:

2. Please enter a username and password for the administration.

* Username:

* Password:

* E-Mail:

Project Homepage | Documentation | Support Forums

FIGURE 32: OPENCART CONFIGURATION (Xiaoyang Zhang 2022)

Figure 33 shows dependency and correctness checking, which is the next step of installation. This webpage is used to check whether the server's settings are correct or not. The upgrade progress would not be interrupted if all settings of the server are correctly configured. After the upgrading is

done, the corresponding status will be shown at the bottom of the page. It refers to the fact that the localhost OpenCart server is preliminary done. And the user is able to get into the OpenCart server by pressing CONTINUE button once again.

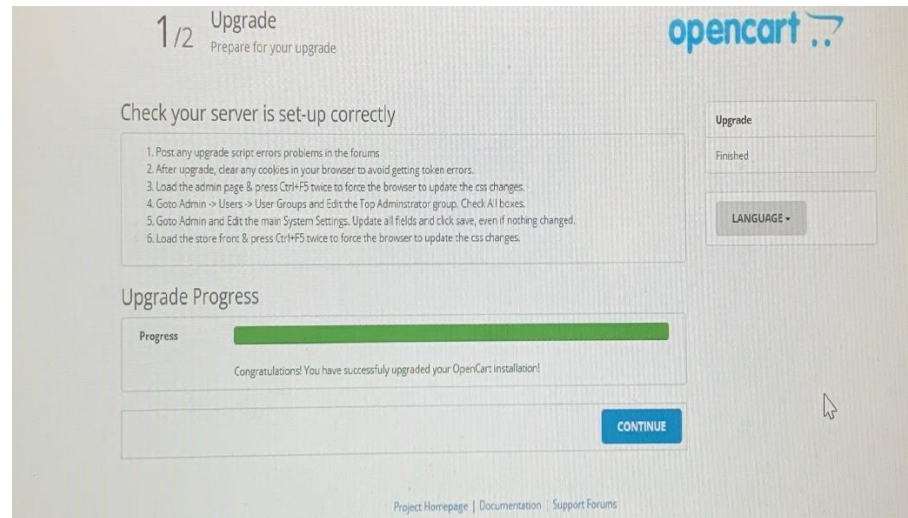


FIGURE 33: OPENCART UPGRADE (Xiaoyang Zhang 2022)

3.3 COMMON OPENCART

Now that the configuration has been set up, the user is permitted to get into the OpenCart website which is named as COMMON OPENCART previously. Figure 34 demonstrates COMMON OPENCART's default web interface. Figure 34 also contains a depiction of some error hints that are popped out from the top of the webpage. These errors hints are caused by the following two files: one was related to errors in the product.php file in the "D:\xampp\htdocs\opencart3\catalog\model\catalog" folder, the other one is from mysqli.php file from "D:\xampp\htdocs\opencart3\system\library\db" folder.

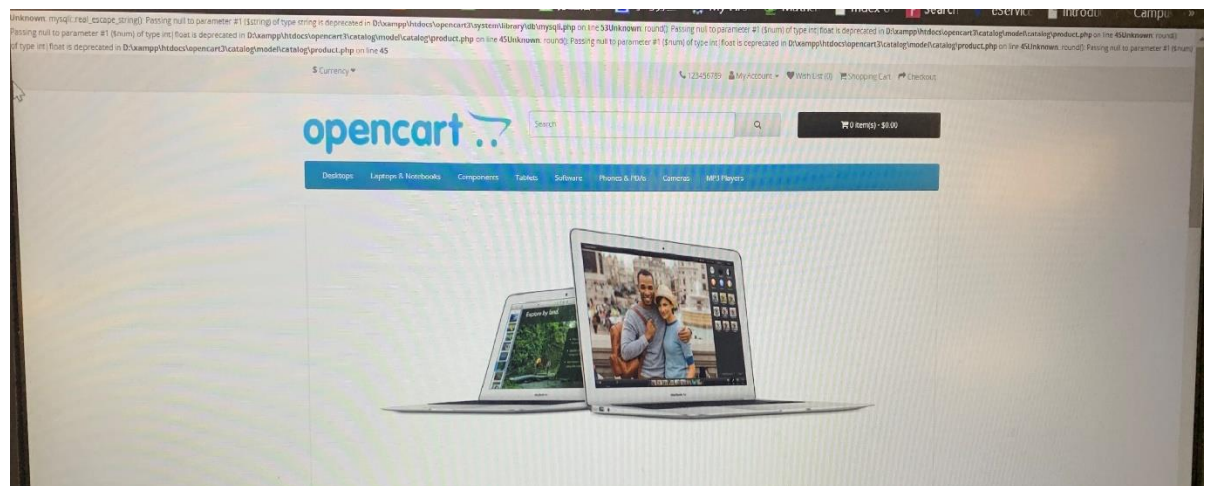


FIGURE 34: COMMON OPENCART'S FIRST LOOK (Xiaoyang Zhang 2022)

These errors are happening because of wrong PHP version (xxvirusxx 2012). In a nutshell, PHP 8.1.5 is used as XAMPP package at this time and it is not compatible with OpenCart 3.0.3.8. So the best solution is to downgrade the PHP version into PHP 7.4 (Raquel Crist, 2021). The way of upgrading XAMPP package is similar to the way of downgrading XAMPP package. Firstly, the user needs to get the needed PHP version. In this project, the user could download PHP 7.4.29 as the substitution and

the link of downloading the XAMPP's PHP 7.4.29 package can be seen in the official XAMPP package website (ApacheFriends s.a). After the user has downloaded the old version of XAMPP package, he needs to replace the PHP 7 version folder with PHP 8 version in the XAMPP folder which is located in the following directory: "D:\xampp". Figure 35 shows the result of replacing the newer PHP version to the older version from the personal computer device.

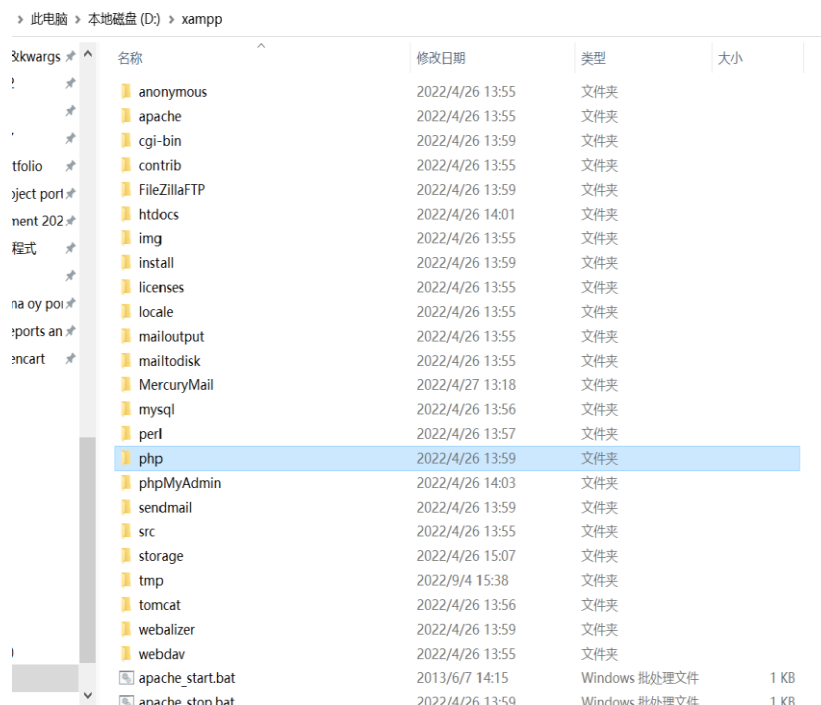


FIGURE 35: PHP 7.4 VERSION REPLACEMENT (Xiaoyang Zhang 2022)

The user also needs to configure the PHP 7.4 folder from the XAMPP Control Panel. However, it is likely to encounter the error "Access is denied due to User Account Control service in Windows operating system" while making the configuration from XAMPP control panel. It is needed to run the XAMPP as Administrator by Default so that the user will get permission for all the events and tasks of the XAMPP server (Fix XAMPP server error xampp-control.ini Access is denied 2020).

The next step is to open the XAMPP Control Panel and click the config button. Figure 36 shows how does the XAMPP control panel looks like:

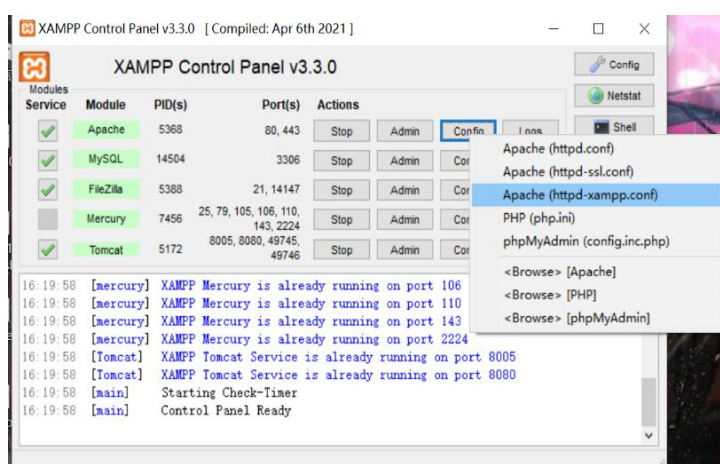


FIGURE 36: SCREENSHOT OF XAMPP CONTROL PANEL (Xiaoyang Zhang 2022)

The user needs to choose the option Apache (httpd-xampp.conf) and it will open the configuration file httpd-xampp.conf. Figure 37 shows the configuration file. Then the user is able to change the PHP version from 8 to 7.

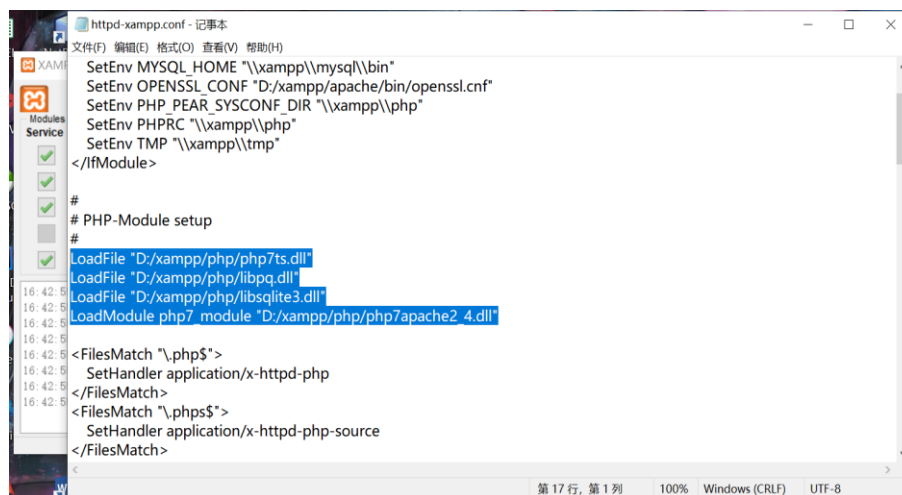


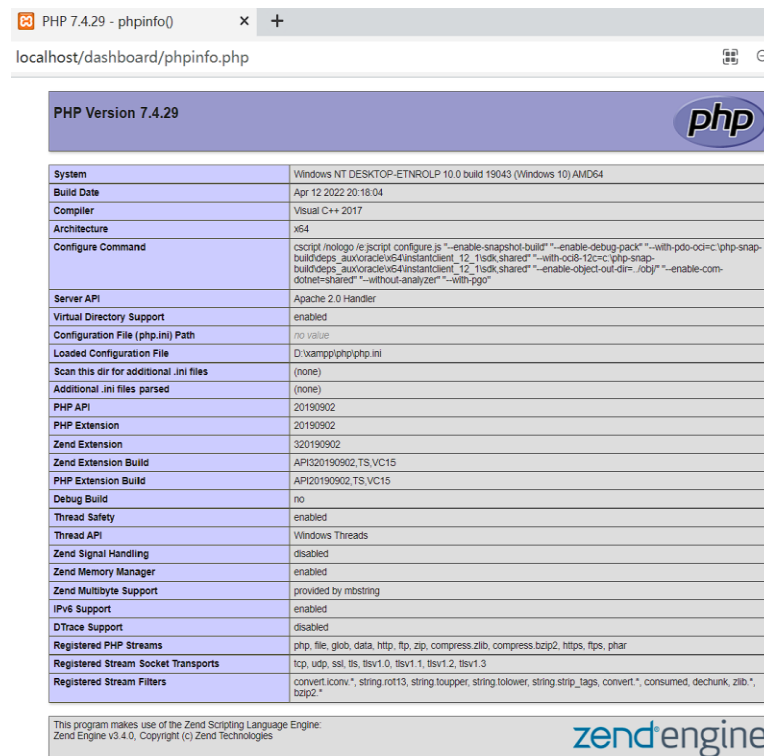
FIGURE 37: SCREENSHOT OF httpd-xampp.conf

Before exiting the httpd-xampp.conf, the last thing for the user to do is to check whether all the lines with php8_module have been changed to php7_module throughout the whole file. Figure 38 shows the most important line that needs to be changed. If all lines have been changed, the user may save the changes and start the Apache Web Server.



FIGURE 38: SCREENSHOT OF THE LINE THAT NEEDS TO BE CHANGED IN httpd-xampp.conf (Xiaoyang Zhang 2022)

Now the user can verify whether the PHP version was successfully downgraded by entering "http://localhost/dashboard/phpinfo.php" to his browser. Figure 39 shows a correct result of this operation. The next step is to set up the OpenCart website.



PHP 7.4.29 - phpinfo()

localhost/dashboard/phpinfo.php

| PHP Version 7.4.29 | |
|---|--|
| System | Windows NT DESKTOP-ETNROLP 10.0 build 19043 (Windows 10) AMD64 |
| Build Date | Apr 12 2022 20:18:04 |
| Compiler | Visual C++ 2017 |
| Architecture | x64 |
| Configure Command | cmd /c "cd /d %~dp0 & phpinfo --enable-snapshot-build --enable-debug-pack --with-pdo-oci=C:\php-snap-build\deps_aux\oracle64\instantclient_12_1\sdk\shared --with-oci8-12c=C:\php-snap-build\deps_aux\oracle64\instantclient_12_1\sdk\shared --enable-object-out-dir=.\obj --enable-com-dotnet=shared --without-analyzer --with-pgo" |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | enabled |
| Configuration File (php.ini) Path | no value |
| Loaded Configuration File | D:\xampp\php\php.ini |
| Scan this dir for additional .ini files | (none) |
| Additional .ini files parsed | (none) |
| PHP API | 20190902 |
| PHP Extension | 20190902 |
| Zend Extension | 320190902 |
| Zend Extension Build | API320190902.TS.VC15 |
| PHP Extension Build | API20190902.TS.VC15 |
| Debug Build | no |
| Thread Safety | enabled |
| Thread API | Windows Threads |
| Zend Signal Handling | disabled |
| Zend Memory Manager | enabled |
| Zend Multibyte Support | provided by mbstring |
| IPv6 Support | enabled |
| DTrace Support | disabled |
| Registered PHP Streams | php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar |
| Registered Stream Socket Transports | tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3 |
| Registered Stream Filters | convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.* |

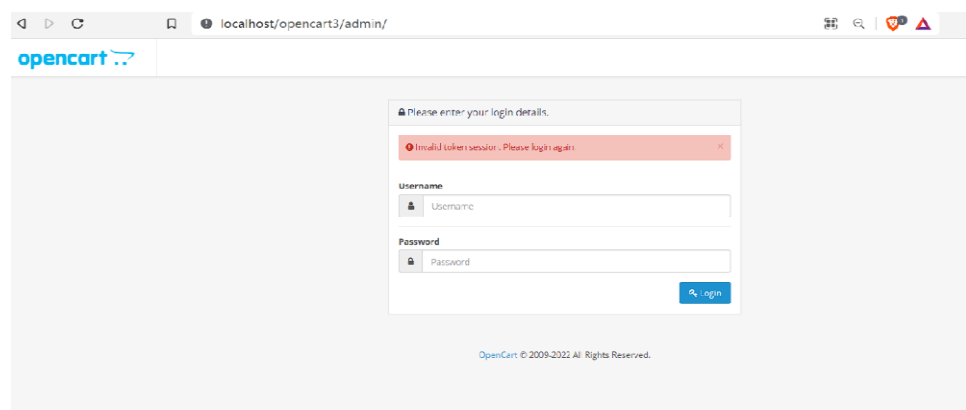
This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies

zend engine

FIGURE 39: SCREENSHOT OF VERIFYING XAMPP PHP VERSION (Xiaoyang Zhang 2022)

3.4 COMMON OPENCART website

The next step is to fill products into OpenCart webpage. In this project, the user is going to build the same UI as Smart Factory's OpenCart webpage since it will facilitate the transfer of data between COMMON OPENCART and Smart Factory's OpenCart. It is needed to change the product option in OpenCart webpage. The user has to get to the administration side of the store firstly. In this project, the URL for the localhost administration server webpage is "http://localhost/opencart3/admin/". Figure 40 shows the OpenCart's administration login page. The administration webpage can't be seen by the customers and it can only be visible for the web developers. (MarketinSG 2011)



localhost/opencart3/admin/

opencart

Please enter your login details.

Invalid token session - Please login again

Username

Username

Password

Password

Login

OpenCart © 2009-2022 All Rights Reserved.

FIGURE 40: COMMON OPENCART ADMIN LOGIN WEBPAGE (Xiaoyang Zhang 2022)

In the webpage, the user has to input the "Username" and "Password". Both of them had already been set previously. In this project, both "Username" and "Password" had been set defaultly as "admin". After entering "Username" and "Password", the user will formally reach into the admin webpage. The admin's UI page is shown in Figure 41.

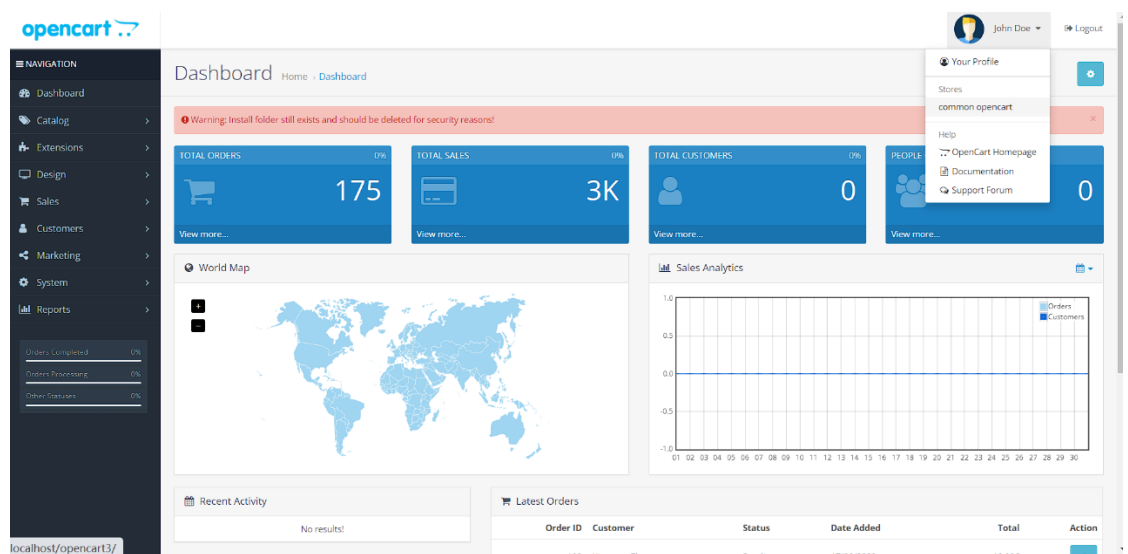


FIGURE 41: COMMON OPENCART ADMIN WEBPAGE (Xiaoyang Zhang 2022)

On the webpage it is possible to find the "Store" option from admin webpage. By clicking the option, the browser will jump into the customer's UI. Figure 42 shows a default store's webpage. That is the OpenCart webpage for customers to order the products from COMMON OPENCART. Since this project's COMMON OPENCART is built locally, the URL for turns out to be "http://localhost/opencart3/". And this URL is exactly the webpage that the user needs to modify for the customers.

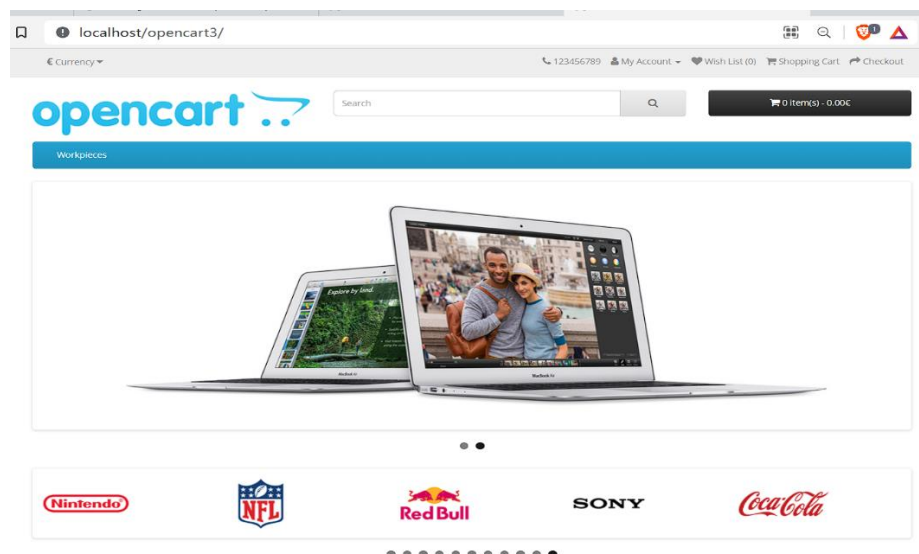


FIGURE 42: COMMON OPENCART WEBPAGE FOR CUSTOMERS (Xiaoyang Zhang 2022)

Now that the user has known where to find the OpenCart webpage for customers, he could manage to modify the webpage to change the product option in OpenCart webpage. The user can change the

product option under admin > catlog > options from admin webpage (MarketinSG 2011). Used options are shown in Figure 43.

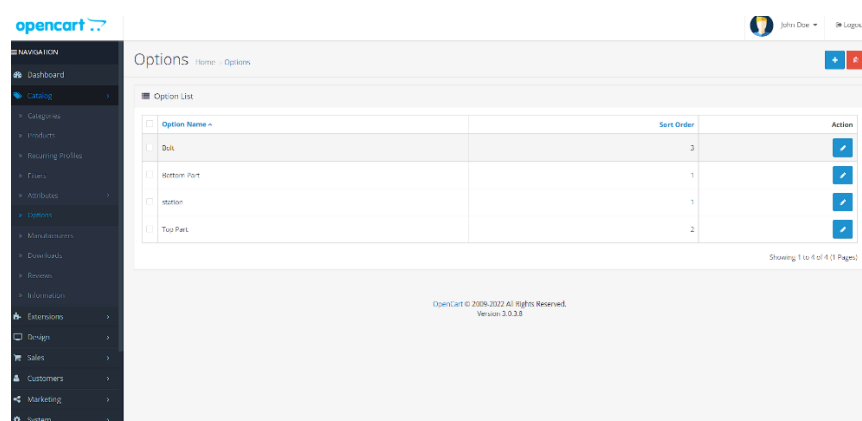


FIGURE 43: SCREENSHOT OF COMMON OPENCART'S PRODUCT OPTIONS CHANGE (Xiaoyang Zhang 2022)

However, merely changing COMMON OPENCART's product options is not enough to make it similar to Smart Factory's OpenCart UI webpage. Hence, more changes should be done. Next the user can try to add images for the COMMON OPENCART from admin webpage. The Image Manager can be used in the administration to upload the image files of products (OpenCart s.a). However, before uploading images to the COMMON OPENCART, the user needs to extract all the product images from Smart Factory's OpenCart server since both two servers share the same product manufacturing chain. For this project, all the images that will be added to COMMON OPENCART are all extracted from the Kuopio Smart Factory's server and then sorted in the "lucas nulle pics" folder.

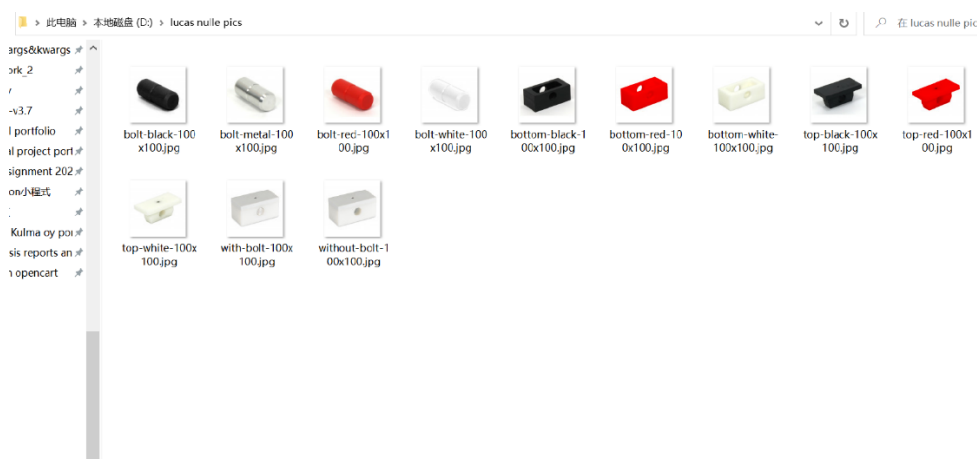


FIGURE 44: SCREENSHOT OF "lucas nulle pics" folder (Xiaoyang Zhang 2022)

In general, there are two places in the admin webpage of COMMON OPENCART where images need to be added. One is from Catalog > Products and the other one is from Catalog > Options. So hereby is one sample of uploading images from "lucas nulle pics" folder with image manager at Catalog > Products. In this sample, the user is going to add image to "product with bolt" option. Figure 45 shows a web UI of "Catalog > Products > Images" folder.

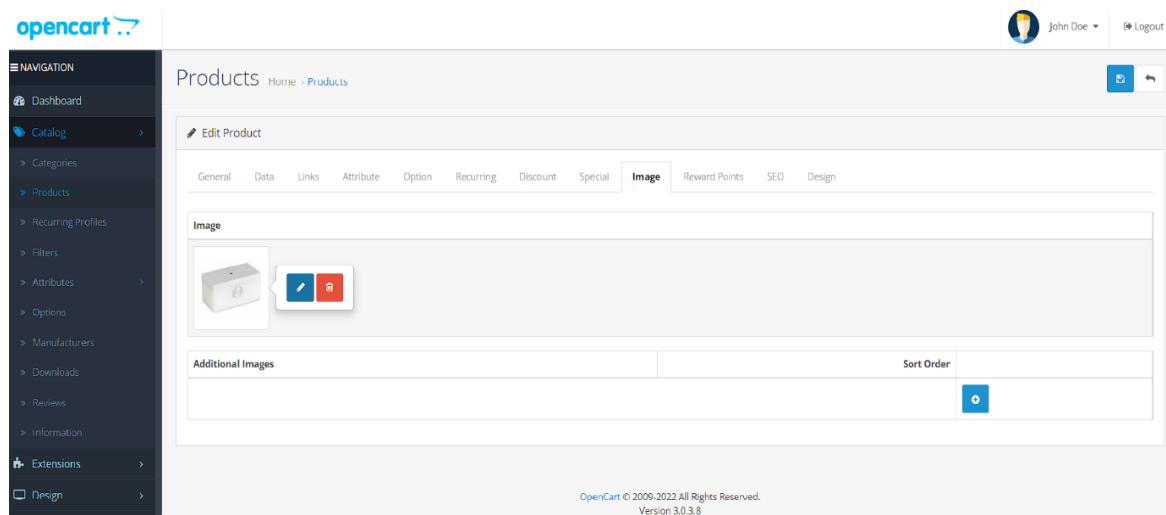


FIGURE 45: "EDIT IMAGE" BUTTON FROM CATALOG > PRODUCTS (Xiaoyang Zhang 2022)

By clicking the blue "Edit" button, the Image Manager will be popped out and the user is able to submit the folder with images by pressing the "Upload" button. Figure 46 shows a way of uploading images. After these steps the "lucas nulle pics" folder has been uploaded and the user is able to use all images inside it. To change images, the user needs to press delete button and re-upload new images by pressing the "refresh" button which is to the left side of "upload" button.

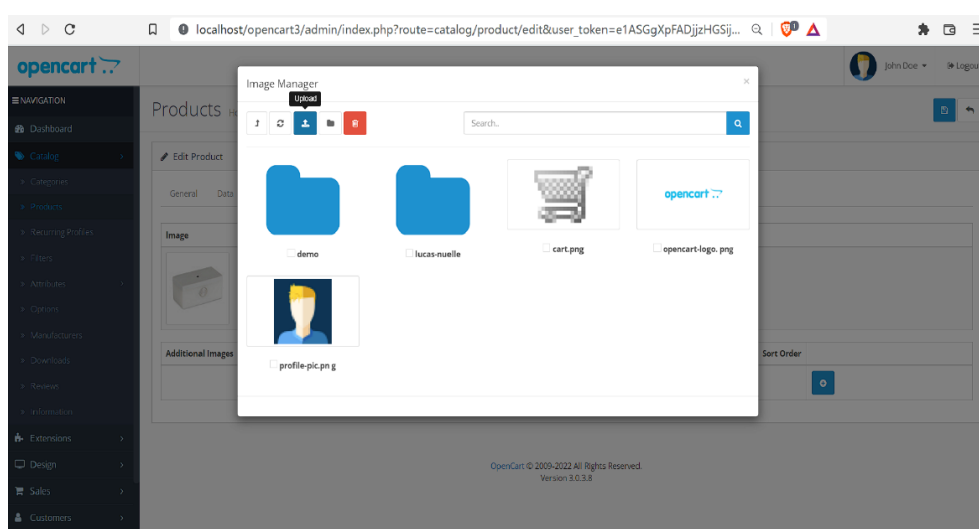


FIGURE 46: "upload" BUTTON FROM Image Manager (Xiaoyang Zhang 2022)

The next step is to set up the currency system of COMMON OPENCART. Due to the fact that the currency system of Kuopio Smart Factory is set to Euro, the one from COMMON OPENCART shall be set to Euro as well. US dollar currency is enabled by default from admin webpage and the user can go to System > Localization > Currencies where he will see the currencies available for use in COMMON OPENCART (Rupak Nepali, 2019). Figure 47 shows a currency UI. It can also be seen that Euro has been set as the default currency as well as it is set to a value of 1. That is, when the user has entered the product's price, the system will do price calculations against the background of Euro.

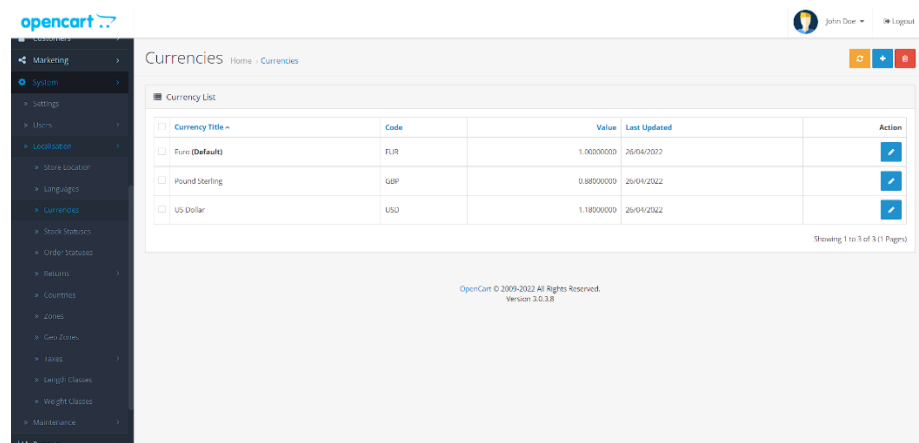


FIGURE 47: CURRENCY SYSTEM IN COMMON OPENCART (Xiaoyang Zhang 2022)

As it has been mentioned above, US dollar is enabled by default from admin webpage. But in this project, the default currency is required to be set as Euro. To change the default currency in COMMON OPENCART, the user needs to get to System > Settings firstly. After that, the Store List can be accessed (Rupak Nepali, 2019). Figure 48 shows a “Stores” UI. Then the user should click the blue “Edit” button on the right side.

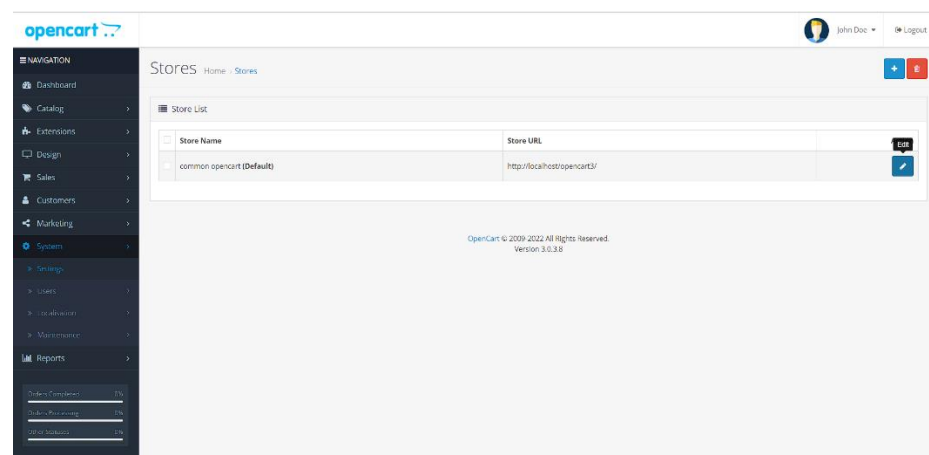


FIGURE 48: SCREENSHOT OF “System > Settings > Stores” FROM COMMON OPENCART (Xiaoyang Zhang 2022)

After the “Edit” button has been clicked, the user needs to click the local tab and then he will reach to the settings menu. From there the user can change his currency and make it as default. Figure 49 shows a default currency setting.

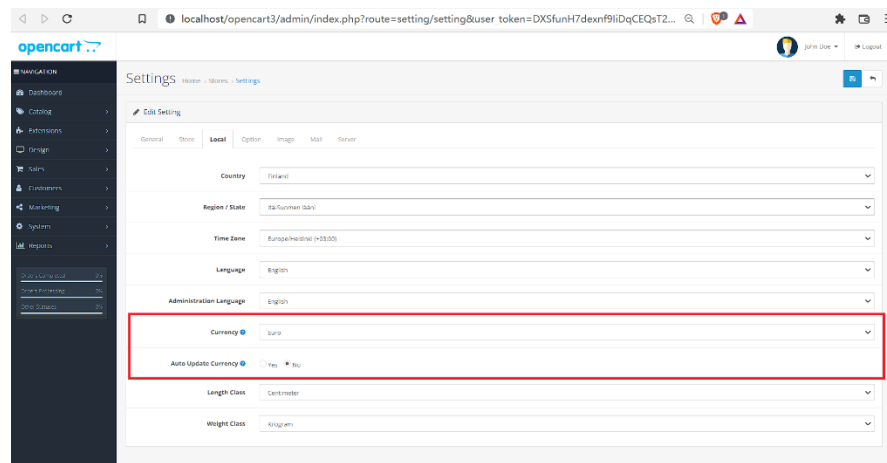


FIGURE 49: THE PLACE OF CHANGING THE DEFAULT CURRENCY (Xiaoyang Zhang 2022)

Now that all the preliminary setup for COMMON OPENCART server is done, it is needed to try ordering the product on COMMON OPENCART to test how reliable it is. However, an error will appear right after the order will be placed on the system from COMMON OPENCART. Figures 50 and 51 show the default problem.



FIGURE 50: FIRST PARAGRAPH OF THE ERROR MESSAGE (Xiaoyang Zhang 2022)

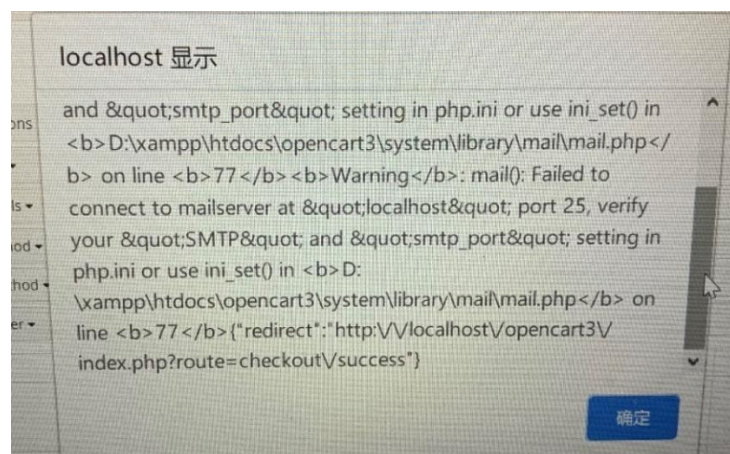


FIGURE 51: SECOND PARAGRAPH OF THE ERROR MESSAGE (Xiaoyang Zhang 2022)

This error is caused by STMP (Simple Mail Transfer Protocol) service which is not set up correctly in COMMON OPENCART. It is always necessary for the real user to contact the shop's hosting provider to configure PHP mail.

To avoid this problem, the user needs to login to COMMON OPENCART admin page and navigate to System > Settings > Edit > Mail (BENJITHREESTACK, 2014). Then the user has to open up the Mail setting tab. The next step for the user to do is to set the Mail Protocol to STMP. After that, the user should set the SMTP Host into "ssl://smtp.gmail.com". And it is very important for the user to include "ssl://" while setting the STMP Host. Otherwise, customers will receive errors when completing actions on COMMON OPENCART that use the mail feature. STMP Port should be "465" and STMP Timeout could be left set to the default of 5 (BENJITHREESTACK, 2014). The final step is choosing the Alert Mail. The "Orders" option has been chosen for COMMON OPENCART. Figure 52 shows correct mail settings page for COMMON OPENCART. After the above steps were completed, more Checkout tests from COMMON OPENCART had been implemented at that time and the orders had come out perfectly.

The screenshot shows the 'Mail' settings page in the COMMON OPENCART admin interface. The page has tabs for 'General', 'System', 'Users', 'Orders', 'Images', 'Mail', and 'Security'. The 'Mail' tab is active. Under the 'General' section, the 'Mail Engine' is set to 'Mail'. The 'SMTP Hostname' is 'ssl://smtp.gmail.com', 'SMTP Username' is 'admin@gmail.com', 'SMTP Password' is 'password', and 'SMTP Port' is '465'. The 'SMTP Timeout' is set to '5'. Under the 'Mail Alerts' section, the 'Alert Mail' is set to 'Orders'. There is also an 'Additional Alert Mail' field.

FIGURE 52: MAIL SETTINGS FOR COMMON OPENCART (Xiaoyang Zhang 2022)

The final configuration step of COMMON OPENCART is to make guest configuration. These options could also be accessed on COMMON OPENCART settings page.

The screenshot shows the 'Checkout' page in the COMMON OPENCART admin interface. The page has a 'Workspaces' header and a 'Shopping Cart' button. The 'Checkout' section is active. Under 'Step 1: Checkout Options', there are two sections: 'New Customer' and 'Returning Customer'. The 'New Customer' section has a 'Checkout Options' section with 'Register Account' and 'Guest Checkout' options. The 'Guest Checkout' option is selected. The 'Returning Customer' section has a 'Returning Customer' section with 'E-Mail', 'Password', and 'Forgot Password' options. The 'E-Mail' and 'Password' fields are empty. The 'Forgot Password' link is visible. Below the 'Guest Checkout' section, there is a 'Continue' button. Below the 'Returning Customer' section, there is a 'Login' button. Below the 'Guest Checkout' section, there is a list of steps: 'Step 2: Account & Billing Details', 'Step 3: Delivery Details', 'Step 4: Delivery Method', 'Step 5: Payment Method', and 'Step 6: Confirm Order'.

FIGURE 53: GUEST CHECKOUT OPTION IN STEP1 (Xiaoyang Zhang 2022)

Figure 53 shows the Step 1 of checkout. The Checkout Options in Step 1 is set to "Register Account" defaultly after COMMON OPENCART was built so every time after the order is sent to Checkout

webpage, the customers have to manually choose "Guest Checkout" option instead of "Register Account" option. To avoid this step during development, it is needed to change contents of webpage's behavior file. Figure 54 shows a correction that could be applied.

checked="checked" is correct ?!

I think, in catalog/view/theme/default/template/account/login.tpl, you would simply have to change:

Code: [Select all](#)

```
<input type="radio" name="account" value="guest" id="guest" />
```

to:

Code: [Select all](#)

```
<input type="radio" name="account" value="guest" id="guest" checked="checked" />
```

It may require slightly more if you change the order.

Free v1.4.9 Extensions: [Default Specials](#) | [Improved Search](#) | [Customer Activity Report](#) | [Customer Groups](#) | [Royal Mail With Handling](#) | [Improved Product Page](#) | [Random Products](#) | [Stock Report](#) | [All Products](#)

FIGURE 54: THE SOLUTION OF CHANGING DEFAULT TO GUEST CHECKOUT (mystifer 2010)

Before seeking for this file in XAMPP source files folder, there is one important thing that needs to be explained. The thing is that the material (mystifer 2010) is written for the users whose OpenCart servers are in version 2.X and their backend template files are .tpl ones. However, the project's OpenCart version is 3.0.3.8 in this project and its backend's template files are .twig files. This will not cause any problems since XAMPP backend files in different versions are roughly the same to each other. In this project, the login.twig (written as twig.tpl) is set under the following directory from XAMPP source files folder: "D:\xampp\htdocs\opencart3\catalog\view\theme\default\template\checkout". (mystifer 2010)

Below are the original contents of login.twig file from line 14 to line 21.

```
<div class="radio">

    <label> {% if account == 'guest' %}

        <input type="radio" name="account" value="guest" checked="checked" />

        {% else %}

        <input type="radio" name="account" value="guest" />

        {% endif %}

    {{ text_guest }}</label>

</div>
```

The user needs to add checked = "checked" after value = "guest" from line 18, so the adapted code works properly (mystifer 2010).

```
<div class="radio">

<label> {% if account == 'guest' %}

<input type="radio" name="account" value="guest" checked="checked" />

{% else %}

<input type="radio" name="account" value="guest" checked="checked" />

{% endif %}

{{ text_guest }}</label>

</div>
```

Then the user could save the adapted code into new login.twig file and make a backup file for it as well. After all these steps have been done correctly, the Checkout Options in Step1 will be set to "Guest Checkout" permanently. However, the modification of COMMON OPENCART checkout webpage is not completed at this moment. Another modification that requires to be done is from Step 5 of Checkout. Customers will encounter the choice of "I have read and agree to the Terms & Conditions" after they have reached to Step 5 and it is set as not ticked status defaultly.

Once again, customers have to check it every time. To solve this problem, the user should modify this file: "catalog/view/theme/default/template/checkout/payment_method.twig" (Rupak Nepali 2019). In this project, the payment_mthod.twig file can be seen through this directory: "D:\xampp\htdocs\opencart3\catalog\view\theme\default\template\checkout\payment_method.twig".

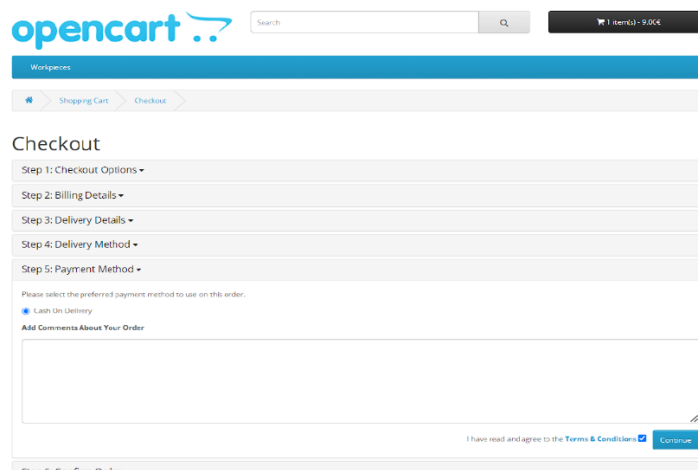


FIGURE 55: "Terms and Conditions" CHOICE

After finding out the `payment_method.twig` from the directory, the user needs to modify it. Specifically speaking, the user can see the following code from line 26 to line 36 once he has opened it:

```
<div class="buttons">

  <div class="pull-right">{{ text_agree }}

    {% if agree %}

      <input type="checkbox" name="agree" value="1" checked="checked" />

    {% else %}

      <input type="checkbox" name="agree" value="1" />

    {% endif %}

    &nbsp;

    <input type="button" value="{{ button_continue }}" id="button-payment-method" data-loading-
text="{{ text_loading }}" class="btn btn-primary" />

  </div>

</div>
```

And what the user has to add `checked = "checked"` after `value = "1"` from line 31 (Rupak Nepali 2019), so the adapted code should look like this:

```
<div class="buttons">

  <div class="pull-right">{{ text_agree }}

    {% if agree %}

      <input type="checkbox" name="agree" value="1" checked="checked" />

    {% else %}

      <input type="checkbox" name="agree" value="1" checked="checked"/>

    {% endif %}

    &nbsp;

    <input type="button" value="{{ button_continue }}" id="button-payment-method" data-loading-
text="{{ text_loading }}" class="btn btn-primary" />

  </div>
```

</div>

After the code has been adapted, the user just needs to save the changed file and keep a copied old file as a backup. The configuration of COMMON OPENCART is done.

3.5 Get API token with Python

As is mentioned above, the API token plays an important role on making communications between servers. Thus, a Python program is written with campus's host to fetch the API token from Kuopio Smart Factory's server as the first step of creating the application in Python and the code looks like this:

```
import requests

# Fetch CART content

username = 'distribution'

key='c1jGuwOsusvNfOfAekAjZhswUAxwTz8bU7cFicvSYsCjdeLiJMWBWWFKgrPKW0xUmv7ImGcmI-
TUCvZ1z42v3hCKsw1HNqym4POeWUPkHbWc37M843WC5HJX-
uSOI00By1PA2224MsFO65hv1gdvbfRsnGNmSVk0yfpAZbz4kBdDaJIH4F4SUnqHzOVTu-
gUSlsKgYVJytAbgeu7CDkMsuIiLNmGxbRWTa1h8B6rQa83s1kY4SYeOT1CQniJRpnU2F5'

# Establishing session for API user by key PARAMS

session = requests.Session()

r = session.post(

    'http://192.168.3.177/shop/index.php?route=api/login',

    data={'username':username, 'key':key}

).text

print(r)

# Create a variable for token in each session.post

data = r.split(",")

data = data[1].split(":")
```



```

data = data[1][:-1]

print(data)

token = data.replace("\\'", "'")

print(token)

```

As what can be seen through the code above, the Python Requests library is imported in this program. The Python Requests library is an HTTP library which is used for making HTTP requests simpler. Figure 55 shows the result of running this program: the terminal from VS code returns **Notice**: Undefined index: api_token in **/var/www/shop/catalog/controller/startup/session.php** on line **8** {**"success": "Success: API session successfully started!", "api_token": "e4b3bf8c18c322d2e948646a39"**}. It refers to the fact that the API token has been successfully achieved and it is "e4b3bf8c18c322d2e948646a39" for this time. From now the user is able to go on with development of Python program.

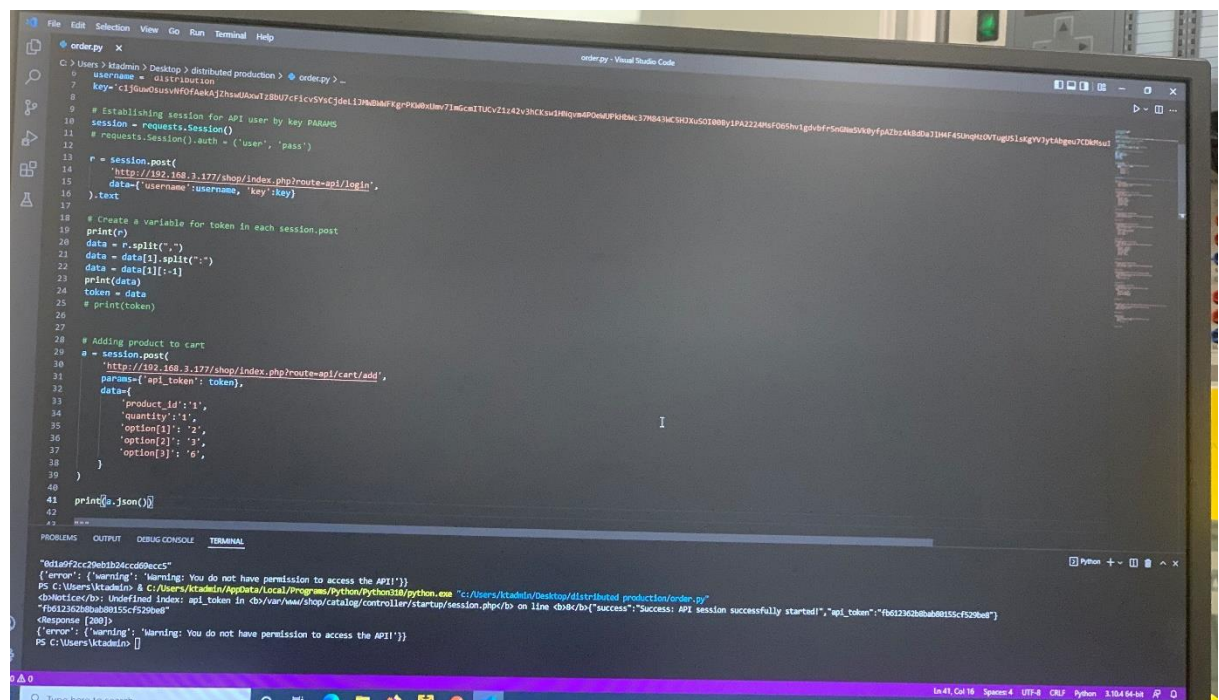


FIGURE 55: PYTHON PROGRAM ON FETECHING API TOKEN (Xiaoyang Zhang 2022)

3.6 Python program for logical test

The next step is to create a Python program with campus's host which can make specific orders and send them to the ERP system of Kuopio's Smart Factory. And after the ERP system has received the order, it should send the order information to the MES system, so the MES system will manufacture

corresponding products. A folder named as “distributed production” with two Python files “mainpage_UI.py” and “order.py” had been created with campus’s host for making this test. And the code of those two Python scripts are attached in the Appendix 1.

The code makes use of Tkinter library which is used to create the user interface for the current “distributed production” program. The user needs to run the project with “mainpage_UI.py” and a Tkinter window in which two buttons will be popped out. The contents of these two buttons are “black black red” as well as “white white metal” which represents for the program’s function that the Kuopio’s Smary Factory can only automatically manufacture product with “black bottom, black top, red bolt” or “white bottom, white top, metal top”. The “order.py” is built to handle requests that need to be sent to ERP system of Kuopio’s Smart Factory. Figure 55 shows the result of running the project from file “mainpage_UI.py” on the bottom.

As a result, the API token has been generated correctly but the error message follows after that and it is written as this: `{'error': {'Warning': 'Warning: You do not have permission to access the API!'}}`. This error is caused by two backend files called “api.php” and “login.php”. And these two PHP files are culprits of making API login error. The way of fixing this problem is shown in Figure 56.



FIGURE 56: SOLUTION FOR API LOGIN ERROR (tdtungit 2017)

Both PHP files’ codes are required to be changed. The change shall remain the same as what is written below.

1. api.php:

To deal with the function login() error inside api.php file, the user needs to find the api.php file from the backend file repository of Kuopio Smart Factory at “var/www/shop/catalog/model/account/api.php”. The code that requires changes is:

```

public function login($username, $key) {

    $query = $this->db->query("SELECT * FROM `". DB_PREFIX . "api`
WHERE `username` = '" . $this->db->escape($username) . "' `key` = '" . $this->db->es-
cape($key) . "' AND status = '1'");

    return $query->row;

}

```

Here is a properly patched code:

```

public function login($username, $key) {

    $query = $this->db->query("SELECT * FROM `". DB_PREFIX . "api`
WHERE `username` = '" . $this->db->escape($username) . "' AND `key` = '" . $this->db->es-
cape($key) . "' AND status = '1'");

    return $query->row;

}

```

2. **login.php:**

The login.php file is located in the repository of Kuopio Smart Factory at “var/www/shop/catalog/controller/api/login.php”. The code that needs to be changed:

```

$session = new Session($config->get('session_engine'), $registry);

```

The corrected code:

```

$session = new Session($this->config->get('session_engine'), $this->registry);

```

By saving the result, the user has done the changes for the Kuopio Smart Factory's backend files. In addition, the same changes also need to be done for Iisalmi's Smart Factory later on. Now that the PHP files were modified, the user can continue running the project. Figure 57 shows how does the user interface window created by Tkinter library look like if everything goes right.

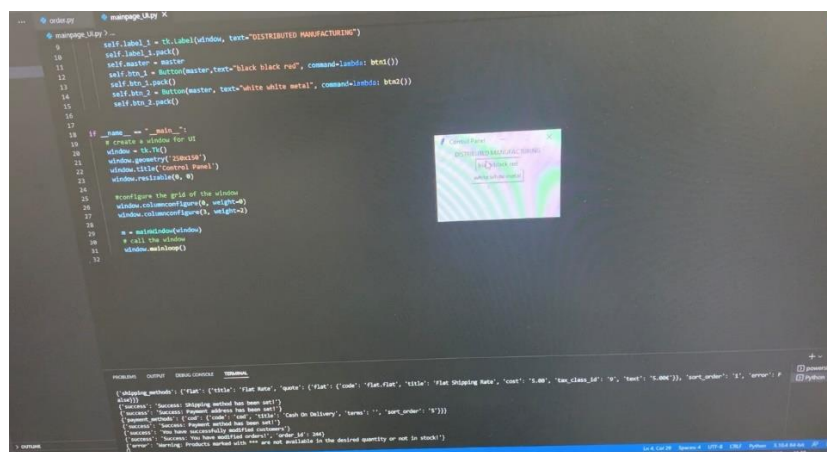


FIGURE 57: UI WINDOW (Xiaoyang Zhang 2022)

As it is shown in Figure 57, the "black black red" button has been chosen. Then the order will be formed by the Python program and be sent to the ERP system of Kuopio Smart Factory. After ERP system gets the order, it will publish the order information to the MES system. When the order information is received by MES system, the Smart Factory will start to manufacture the product. Figure 58 shows the result of the manufacturing.

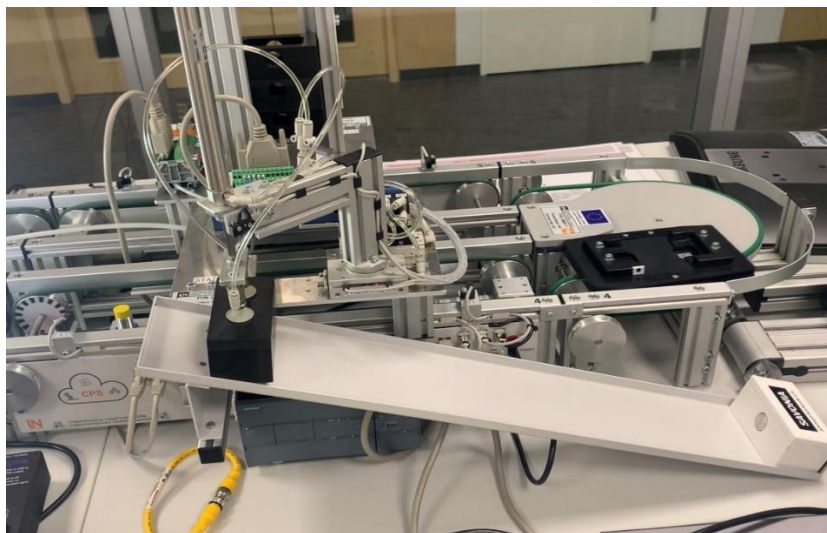


FIGURE 58: RESULT OF MANUFACTURING BY "distributed production" PROJECT (Xiaoyang Zhang 2022)

3.7 Extract order data from COMMON OPENCART databases

As what has been mentioned in chapter 2.2, customers need to make orders with COMMON OPENCART firstly. After that, the python program will send the orders from COMMON OPENCART to ERP systems of two Smart Factories by extracting the necessary order data. This means that the user shall

As it can be seen through FIGURE 60, the "oc_order" contains as much data as the following list shows: product_id, quantity, product_option_id_bottom_with_bolt_normalised, product_option_id_top_with_bolt_normalised, product_option_id_bolt_with_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, shipping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added. All the data above will be extracted from "oc_order" database using Python and then it will be stored in respective variables. As a result, these variables will be sent to the ERP system of Kuopio Smart Factory.

Figures 61 and 61 show the content of "oc_order_option" database as well as "oc_order_product" separately. The user needs to fetch "order_id" from "oc_order_option" database while "oc_order_product" database is responsible for "quantity" data from COMMON OPENCART.

[illegible]

FIGURE 61: SCREENSHOT OF "oc_order_option" DATABASE

FIGURE 62: SCREENSHOT OF "oc_order_product" DATABASE

It is needed to extract the substring between two markers in Python since the outputs of the data are in the form of a string type. Figure 63 shows a code snippet that will be used to process "b.json" output which is generated by fetching order information by API from "oc_order" database. The solution requires the use of Python re (regular expression) library to be imported.

regular expression

```
import re

re.search(r"(?<=AAA).*(?=ZZZ)", your_text).group(0)
```

The above as-is will fail with an `AttributeError` if there are no "AAA" and "ZZZ" in `your_text`

FIGURE 63: REGULAR EXPRESSION OF PYTHON re library (tzot 2011)

Additionally, a Python library BeautifulSoup is also imported in the program which is used for pulling data out of HTML and XML files. The data will be gathered into JSON (JavaScript Object Notation) using its internal libraries.

The code will utilize both libraries and Python's original functionality. The whole code solution could be found in Appendix 2. The code mainly contains two defined functions inside which are named as `GetData()` and `SendData()`. The `GetData()` function is used to get the order data from the COMMON OPENCART using requests library and all the orders are encapsulated into 61 or 62 variables, depending on whether the product ordered by customers from COMMON OPENCART is contains bolt or not. And with the help of Python BeautifulSoup module, all the variables will be contributed into one JSON file. In this project, it is named as `b.json` which is visible inside `GetData()` function. Unlike the responsibility of the `GetData()` function, the `SendData()` which locates inside the `GetData()`, is used for sending the processed order data from COMMON OPENCART into Kuopio Smart Factory server. It utilizes the re library to extract the JSON data made from `SendData()` function and send it into the OpenCart API. Below is a snippet of `SendData()` function.

```
#Bolt

def newOpt1ValBoltfull(norm):

    print("34/31? bolt red ", norm)

    norm = norm.strip()

    print(len(norm))

    if norm == "34":

        print("ret 1")

        return "1"

    elif norm == "31":

        print("ret 2")
```

```
    return "2"
```

```
#Bottom
```

```
def newOpt2ValBoltfull(norm):
```

```
    print("29/28? bottom white", norm)
```

```
    print(type(norm))
```

```
    norm = norm.strip()
```

```
    print(len(norm))
```

```
    if norm == "29":
```

```
        print("ret 4")
```

```
        return "4"
```

```
    elif norm == "28":
```

```
        print("ret 3")
```

```
        return "3"
```

```
#Top
```

```
def newOpt3ValBoltfull(norm):
```

```
    print("27/25? top black ", norm)
```

```
    print(len(norm))
```

```
    print(type(norm))
```

```
    norm = norm.strip()
```

```
    print(len(norm))
```

```
    if norm == "27":
```

```
        print("ret 6")
```

```
        return "6"
```

```
    elif norm == "25":
```

```
        print("ret 5")
```

```
        return "5"
```



```

# Top

def newOpt4ValBoltless(norm):

    #without bolt

    print(type(norm))

    norm = norm.strip()

    print("35/37 NORM: ", norm)

    if norm == "39":

        print("ret 7")

        return "9"

    elif norm == "40":

        print("ret 8")

        return "10"


# Bottom

def newOpt5ValBoltless(norm):

    norm = norm.strip()

    print(type(norm))

    print("39/40 NORM: ", norm)

    if norm == "35":

        # boltless black bottom

        print("ret 9")

        return "7"

    elif norm == "37":

        # boltless white bottom

        print("ret 10")

        return "8"

```

These specific functions are used to synchronize the label Elements of options between COMMON OPENCART server to Kuopio Smart Factory's server. For example: there is a product awaiting to be ordered from COMMON OPENCART which contains white top part. By inspecting the element of White

top option from the COMMON OPENCART, it could be seen that its internal name is "option[230]" and its value is "29". Figure 64 shows an "inspect element" menu of White top option on COMMON OPENCART server.

If the user check the same option from Kuopio Smart Factory's server, he will see different results. That is, the name of the White top from Kuopio Smart Factory turns out to be "option[2]" and value is "4". Figure 65 shows an "inspect element" menu of White top option on Kuopio Smart Factory's server. Because these values are not mutually equivalent, malfunctions will be caused while COMMON OPENCART is sending the order to Kuopio Smart Factory.

Hence, the functions that are mentioned in the above snippet are created to maintain the consistency of name and value in both server Elements. And after many tests been done with these functions, they are proved to be working correctly.

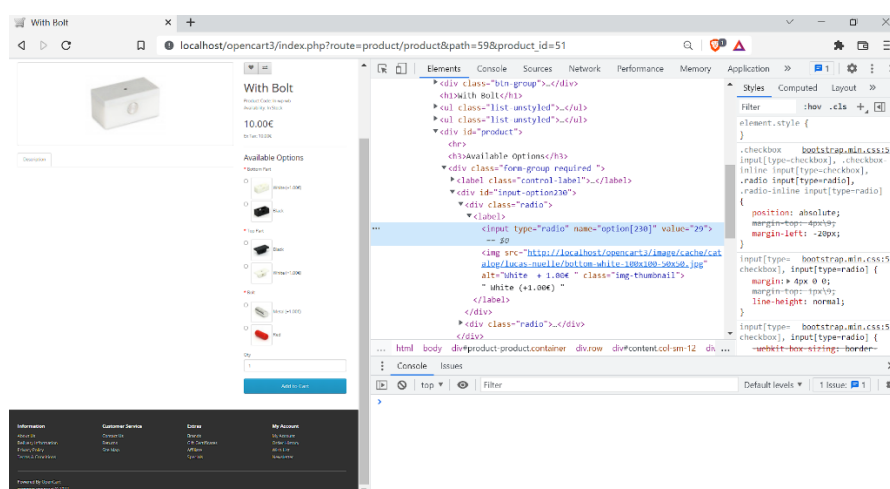


FIGURE 64: THE RESULT OF LABEL ELEMENTS FROM COMMON OPENCART (Xiaoyang Zhang 2022)

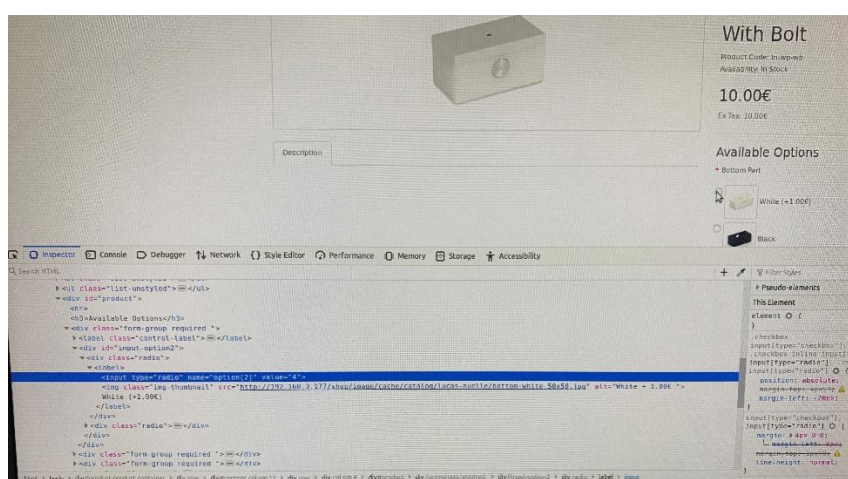


FIGURE 65: THE RESULT OF LABEL ELEMENTS FROM KUOPIO SMART FACTORY (Xiaoyang Zhang 2022)

An important thing is that OpenCart API functions and the order data will be sent to the ERP system of Kuopio Smart Factory with the help of an Ethernet cable. The reason why the connection between COMMON OPENCART server and Kuopio Smart Factory is done physically is because the COMMON OPENCART is set as localhost which means it's private and can only be visible for the developer. However, as what has been mentioned above, it's still possible to set up the COMMON OPENCART as public ip.

Figures 66 and 67 show both ends of the Ethernet cable. One side of the cable is connected to the personal computer which has COMMON OPENCART server inside. The other side of the Ethernet cable is connected to one of the Ethernet ports of IMS station from Kuopio Smart Factory.

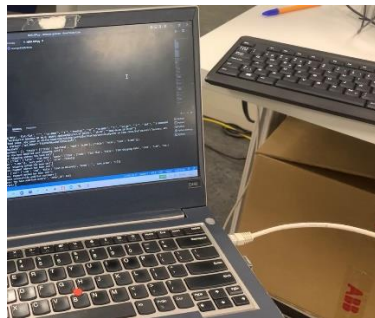


FIGURE 66: THE END OF THE ETHERNET CABLE THAT CONNECTS TO PERSONAL PC (Xiaoyang Zhang 2022)



FIGURE 67: THE END OF THE ETHERNET CABLE THAT CONNECTS TO IMS STATION (Xiaoyang Zhang 2022)

The next step is to test whether the program is working properly. A product order was made from COMMON OPENCART which wanted a product with black top, white bottom and metal bolt. By running the program, the order was sent from COMMON OPENCART server into Kuopio Smart Factory via Ethernet cable connection. And after the ERP system had received an order data, it would transfer the data to a MES system and Kuopio Smart Factory was about to begin the product manufacturing. Figure 68 shows a result of running a python script.

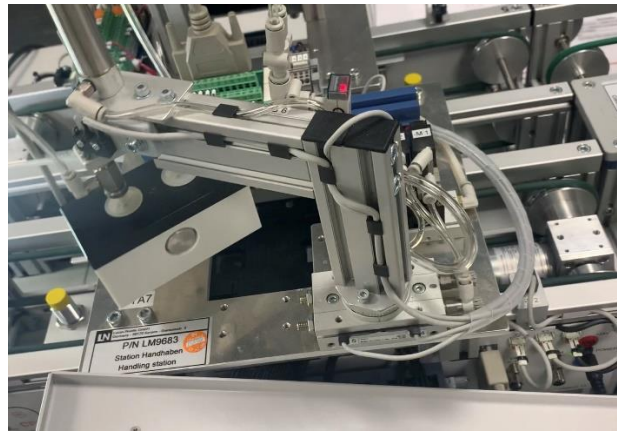


FIGURE 68: THE RESULT OF MANUFACTURING (Xiaoyang Zhang 2022)

3.8 Python script

The rules that are applied to the manufacturing are written below.

1. If the quantity of product is one, the order will be default sent to the Kuopio Smart Factory;
2. If the quantity of product is two, the order will be divided into 50% which means both Kuopio and Iisalmi Smart Factory will do the manufacturing which quantity is set as one;
3. If the quantity is turned out to be bigger than two, the Python script will alarm and the order won't be sent to Smart Factory and executed.

Since the code is working on a single machine, the next step is to generate a connection between the COMMON OPENCART server as well as the Iisalmi Smart Factory server. Due to the fact that Python is a very modern high-level programming language, it will solve the problem of memory release by itself. While the user is programming in C language, he is forced to add "return 0" at the end of the function to avoid stackoverflow problems. However, unlike the situation from C programming language, the user would not be bothered about the stackoverflow problem with Python. He can straightly do the Python programming and let the machine do the compiling and optimizing job itself. This feature really improves the user's experience of writing Python programs. However, once running a large project, the compiling speed of Python program is much slower than languages like C or C++. Eventually the computer's memory might be full, so the program would not function correctly. Hence, the final version of Python scripts in this project is being separated into two files: "NEW_APP.py" and "connection to Iisalmi.py". Among these two files, "NEW_APP.py" is used to handle the connection between the COMMON OPENCART server and Kuopio Smart Factory while "connection to Iisalmi.py" is used for building the connection between the COMMON OPENCART server and Iisalmi Smart Factory. As it has been mentioned from chapter 2.3, the two Smart Factories from different cities have been remotely connected together by VPN service, so the connections between all servers are completed. The complete code of final Python scripts will be stored in Appendix 3.

The python script is mainly focusing on establishing a connection between two Smart Factories. Also, it can be observed from "connection to Iisalmi.py" that there is an additional while loop `quantity_for_all == 2` added inside the program compared to "NEW_APP.py". That is because of the way

of how two Smart Factories are working together: the while condition is used to deal with the situation when an order with quantity that equals to 2 is fulfilled from COMMON OPENCART and being sent in half to ERP systems of smart factories from both places. Both two Python scripts had proven to work successfully.

4 CONCLUSION

This thesis described the basic smart factory design process which is relevant to Industry 4.0. After more investigation is made for Industry 4.0 within Internet of Things, cloud manufacturing, cyber-physical systems and so on, a conclusion can be drawn: the Smart Factory is a colossal project which is based on all the key constitutive technologies mentioned above together. With all these technologies connected, the Industry 4.0 can be realized.

Moreover, the complexity of a modern manufacturing process has already gone beyond the extent to be manually observed or controlled. Hence, an increasing number of researchers are dedicating themselves to improve the performance of all kinds of Smart Factories. As a result, the essential skills of building the Smart Factory's system like data collection, analysis or even AI for depatching the orders are getting better and better every day. Since more and more valuable knowledge from Smart Factory has been excavated, it is a pleasure for one to get an opportunity to come into contact with this project since the importance of manufacturing process will never be overstated.

REFERENCES

Nepali Rupak 2019. How to pull products JSON through API in Opencart?. Internet publication. Webocreation. <https://webocreation.com/pull-products-json-through-api-opencart/>. Accessed 10.05.2022.

Postman 2022. Building requests. Internet publication. Postman. <https://learning.postman.com/docs/sending-requests/requests/>. Accessed 23.04.2022.

Singh Deepshikha 2019. Manual API Testing using Postman for Beginners. Internet publication. Srijan. <https://www.srijan.net/resources/blog/manual-api-testing-using-postman>. Accessed 23.04.2022.

Opencart 2016. API. Internet publication. OpenCart. <https://docs.opencart.com/en-gb/system/users/api/>. Accessed 26.04.2022.

Thedmnyc 2020. How to Create OpenCart Website?. Internet publication. DIGITAL MEDIA. <https://thedmnyc.com/how-to-create-website-with-opencart/>. Accessed 22.04.2022.

Apachefriends 2022. XAMPP for Windows 7.4.29, 8.0.19 & 8.1.6. Internet publication. Apache Friends. <https://www.apachefriends.org/download.html>. Accessed 25.04.2022.

Gigapro 2018. Installation. Internet publication. OpenCart forum. <https://forum.opencart.com/viewtopic.php?t=201850>. Accessed 25.04.2022.

Zia Hamza 2021. Use OpenCart To Power Your Ecommerce Store. Internet publication. Cloudways. <https://www.cloudways.com/blog/setup-opencart-ecommerce-store/>. Accessed 25.04.2022.

Codeanddeploy 2021. How to Install PHP GD Extension on Windows WAMP and XAMPP Server?. Internet publication. Codeanddeploy. <https://codeanddeploy.com/blog/php/how-to-install-php-gd-extension-on-windows-wamp-and-xampp-server>. Accessed 22.04.2022.

Xxvirusxx 2012. ERROR with mysqli.php file on line 53 in \system\library\db\. Forum by OpenCart Community. <https://forum.opencart.com/viewtopic.php?f=199&t=227216>. Accessed 25.04.2022.

Crist Raquel 2021. How to upgrade Xampp from php 7 to 8 in windows 10. Internet publication. We-code101. <https://wecode101.com/upgrade-xampp-from-php-7-to-8-windows-10>. Accessed 26.04.2022.

Free Courses 2020. Fix XAMPP server error xampp-control.ini Access is denied. Video. Director Free Courses. Youtube video service, published 09.04.2020. <https://www.youtube.com/watch?v=jh1q0w-mHKU>. Accessed 26.04.2022.

MarketinSG 2011. Change Product Option "Option Value" Text. Forum by OpenCart Community. <https://forum.opencart.com/viewtopic.php?t=113157>. Accessed 26.04.2022.

OpenCart 2016. Image manager. Internet publication. OpenCart. <http://docs.opencart.com/en-gb/administration/image-manager/>. Accessed 26.04.2022.

Nepali Rupak 2019. Currencies management in the OpenCart 3. Internet publication. Webocreation. <https://webocreation.com/currencies-management-in-the-opencart-3/>. Accessed 26.04.2022.

BENJITHREESTACK 2014. How to Use Google's SMTP Service with OpenCart. Internet publication. Medium. <https://medium.com/@BENJITHREESTACK/how-to-use-googles-smtp-service-with-opencart-944b9c5c76ea>. Accessed 26.04.2022.

Mystifer 2010. [SOLVED] account/login - change default to guest checkout. Forum by OpenCart Community. <https://forum.opencart.com/viewtopic.php?t=19829>. Accessed 27.04.2022.

Nepali Rupak 2019. Opencart - How to select the 'Terms' by default on the checkout page. Answer by Stackoverflow. <https://stackoverflow.com/questions/56910207/opencart-how-to-select-the-terms-by-default-on-the-checkout-page>. Accessed 29.04.2022.

Tdhungit 2017. API Login Error - OpenCart 3.0.2.0. Forum by OpenCart Community. <https://forum.opencart.com/viewtopic.php?t=186063>. Accessed 10.05.2022.

Tzot 2011. How to extract the substring between two markers?. Answer by Stackoverflow.
<https://stackoverflow.com/questions/4666973/how-to-extract-the-substring-between-two-markers>.
Accessed 25.05.2022.

APPENDIX 1: THE CODE OF PYTHON SCRIPT FOR API CONNECTION TEST

1. mainpage_UI.py:

```

from multiprocessing.spawn import import_main_path

import tkinter as tk

from tkinter import LEFT, RIGHT, TOP, Label, Tk, Toplevel, ttk, Button

from order import btn1, btn2


class mainWindow(object):

    def __init__(self, master):

        self.label_1 = tk.Label(window, text="DISTRIBUTED MANUFACTURING")

        self.label_1.pack()

        self.master = master

        self.btn_1 = Button(master, text="black black red", command=lambda: btn1())

        self.btn_1.pack()

        self.btn_2 = Button(master, text="white white metal", command=lambda: btn2())

        self.btn_2.pack()


if __name__ == "__main__":

    # create a window for UI

    window = tk.Tk()

    window.geometry('250x150')

    window.title('Control Panel')

    window.resizable(0, 0)

    #configure the grid of the window

```

```
window.columnconfigure(0, weight=0)
```

```
window.columnconfigure(3, weight=2)
```

```
m = mainWindow(window)
```

```
# call the window
```

```
window.mainloop()
```

2. order.py:

```
from posixpath import split
```

```
from turtle import update
```

```
import requests
```

```
import time
```

```
# black black red
```

```
def btn1():
```

```
    o1 = 2
```

```
    o2 = 3
```

```
    o3 = 5
```

```
    Post(o1,o2,o3)
```

```
# white white metal
```

```
def btn2():
```

```
    o1 = 1
```

```
    o2 = 4
```

```
    o3 = 6
```

```
    Post(o1,o2,o3)
```

```

def Post(o1,o2,o3):

    # Fetch CART content

    username = 'distribution'

    key='c1jGuwOsusvNfOfAekAjZhswUAxwTz8bU7cFicvSYsCjdeLiJMWBWWFKgrPKW0xUmv7ImGcmI-
TUCvZ1z42v3hCKsw1HNqvm4POeWUPkHbWc37M843WC5HJX-
uSOI00By1PA2224MsFO65hv1gdvbfSrSnGNmSVk0yfpAZbz4kBdDaJIH4F4SUnqHzOVTu-
gUSIsKgYVJytAbgeu7CDkMsuIiLNmGxbRWTa1h8B6rQa83s1kY4SYeOT1CQniJRpnU2F5'

    # Establishing session for API user by key PARAMS

    session = requests.Session()

    # requests.Session().auth = ('user', 'pass')

    r = session.post(

        'http://192.168.3.177/shop/index.php?route=api/login',

        data={'username':username, 'key':key}

    ).text

    # Create a variable for token in each session.post

    print(r)

    data = r.split(",")

    data = data[1].split(":")

    data = data[1][:-1]

    print(data)

    token = data.replace("\\\"", "")

    time.sleep(1) # Sleep for 1 second

    # print(token)

```

```
# Get cart content

cart = session.get(

    'http://192.168.3.177/shop/index.php?route=api/cart/products',

    params={'api_token':token},

    #data={'product_id': '1'}

)

print(cart.json())

# Adding product to cart

a = session.post(

    'http://192.168.3.177/shop/index.php?route=api/cart/add',

    params={'api_token': token},

    data={

        'product_id':'1',

        'quantity':'1',

        'option[1]': o1 ,# 1 or 2

        'option[2]': o2,# 3 or 4

        'option[3]': o3,# 5 or 6

    }

)

print(a.json())

# Set shipping address for current session

b = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/address',
```

```
params={'api_token':token},

data={

    'firstname':'Hello',

    'lastname':'World',

    'address_1':'Musterstr. 1',

    'city':'Musterstadt',

    'country_id':'81',

    'zone_id':'1263',

    'postcode': '01234',

}

)

print(b.json())

# Returning available shipping methods

c = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/methods',

    params={'api_token':token},

)

print(c.json())

# Set shipping method for current session

d = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/method',

    params={'api_token':token},

    data={
```

```

        'shipping_method': 'flat.flat',

        'title': 'Flat Shipping Rate',

        'cost': '5.00',

        'tax_class_id': '9',

        'text': '5.00€',

    }

)

print(d.json())

# Set payment address for this session

e = session.post(

    'http://192.168.3.177/shop/index.php?route=api/payment/address',

    params={'api_token': token},

    data={

        'firstname': 'Firstname',

        'lastname': 'Lastname',

        'address_1': 'Musterstr. 1',

        'city': 'Musterstadt',

        'country_id': '81',

        'zone_id': '1263',

        'postcode': '01234',

    }

)

print(e.json())

# Returning available payment methods

```



```
f = session.post(
    'http://192.168.3.177/shop/index.php?route=api/payment/methods',
    params={'api_token': token},
)

print(f.json())

# Setting payment method of available in api/payment/methods
g = session.post(
    'http://192.168.3.177/shop/index.php?route=api/payment/method',
    params={'api_token': token},
    data={
        'payment_method': 'cod'
    }
)

print(g.json())

# Set customer for current session
h = session.post(
    'http://192.168.3.177/shop/index.php?route=api/customer',
    params={'api_token': token},
    data={
        'firstname': 'Firstname',
        'lastname': 'Lastname',
        'email': 'muster@email.de',
        'telephone': '0123456789'
    }
)
```

```
print(h.json())
```

```
# New order by cart content and payment/delivery information has been set by current session
```

```
i = session.post(
    'http://192.168.3.177/shop/index.php?route=api/order/add',
    params={'api_token':token},
    data={'cart_id': '7'}
)
```

```
print(i.json())
```

```
# Return order id
```

```
j = session.get(
    'http://192.168.3.177/shop/index.php?route=api/order/add',
    params={'api_token':token},
    data= {
        'order_id': '1',
    }
)
```

```
print(j.json())
```

APPENDIX 2: THE CODE OF PYTHON SCRIPT FOR KUOPIO SMART FACTORY

```

import requests

from bs4 import BeautifulSoup

import re

import time


# print the separate line

def print_line():

    print("-" * 50)


# For new OpenCart

def GetData():

    # Establishing session for API user by key PARAMS

    session = requests.Session()


    # Fetch CART content

    username = 'Default'

    key= '6fbqrQfEm-
LYPK1B972bN2fgUTGt7vLPCpmBJUeW5Ggk6ZvnoM6C627nMjaL3wkUolKjQR9jLrH-
HeO7HKBwESvOkBKhtTpLmdYdi3tNCGsWq0SHd4ywcrFHnAebvogPdy-
uEIytU8DlwRfg1Zx5aT0QeMXQrCTr55S1crTw9qVe-
OYA1lXl6Eioa4Itmf1JxeooMI37EjuOW8vo5EScyCsPFd3ngg3B7ZsO1tRWKJU0JF9xDfiH122Kybj8vm39
P5t'

    a = session.post(

        'http://127.0.0.1/opencart3/index.php?route=api/login',

        data={'username':username, 'key':key}

    ).text

```

```

print(a)

# print("here")

# Create a variable for token in each session.post

data = a.split(",")

data = data[1]

data = data.replace("}", "")

data = data.replace("\'", "'")

token = data[10::]

print_line()

print(token)

# Print out the latest order and send it if it hasn't been sent before

# Link for extract html data

def getdata(url):

    r = requests.get(url)

    return r.text

# Providing the url for reading

htmldata1 = getdata('http://localhost/phpmyadmin/index.php?route=/sql&db=opencart3&table=oc_order&pos=0')

# Parse the html file

soup1 = BeautifulSoup(htmldata1, 'html.parser')

# Create an empty container

successOrder = "

```

```
for successOrder in soup1.find('td', class_='text-right data grid_edit click2 not_null nowrap'):
```

```
    order_id = successOrder.get_text()
```

```
    print_line()
```

```
    print(order_id)
```

```
# Fetch order info by API
```

```
b = session.get(
```

```
    'http://127.0.0.1/opencart3/index.php?route=api/order/info',
```

```
    params={'api_token':token, 'order_id':order_id},
```

```
    data={
```

```
    }
```

```
)
```

```
print_line()
```

```
print(b)
```

```
print(b.json())
```

```
# Extracting all the needed info from b.json() dynamically
```

```
text = str(b.json())
```

```
t1 = re.search("'firstname': (.+?)," , text)
```

```
if t1:
```

```
    first_name = t1.group(1)
```

```
    firstname = first_name.strip('\"')"
```

```
    print_line()
```

```
    print(firstname)
```

```
t2 = re.search("'lastname': (.+?)," , text)
```

```
if t2:
```

```
last_name = t2.group(1)

lastname = last_name.strip('')

print_line()

print(lastname)
```

```
t3 = re.search("email': (.+?)", text)

if t3:

    Email = t3.group(1)

    email = Email.strip('')

    print_line()

    print(email)
```

```
t4 = re.search("telephone': (.+?)", text)

if t4:

    Telephone = t4.group(1)

    telephone = Telephone.strip('')

    print_line()

    print(telephone)
```

```
t5 = re.search("custom_field': (.+?)", text)

if t5:

    custom_field = t5.group(1)

    print_line()

    print(custom_field)
```

```
t6 = re.search("payment_firstname': (.+?)", text)

if t6:

    paymentFirstname = t6.group(1)

    payment_firstname = paymentFirstname.strip('')
```

```
print_line()

print(payment_firstname)

t7 = re.search("payment_lastname': (.+?)," , text)

if t7:

    paymentLastname = t7.group(1)

    payment_lastname = paymentLastname.strip("''")

    print_line()

    print(payment_lastname)

t8 = re.search("payment_company': (.+?)," , text)

if t8:

    payment_company = t8.group(1)

    print_line()

    print(payment_company)

t9 = re.search("payment_address_1': (.+?)," , text)

if t9:

    paymentAddress_1 = t9.group(1)

    payment_address_1 = paymentAddress_1.strip("''")

    print_line()

    print(payment_address_1)

t10 = re.search("payment_address_2': (.+?)," , text)

if t10:

    payment_address_2 = t10.group(1)

    print_line()

    print(payment_address_2)
```



```
t11 = re.search("'payment_postcode': (.+?)," , text)
```

```
if t11:
```

```
    paymentPostcode = t11.group(1)
```

```
    payment_postcode = paymentPostcode.strip('\"')"
```

```
    print_line()
```

```
    print(payment_postcode)
```

```
t12 = re.search("'payment_city': (.+?)," , text)
```

```
if t12:
```

```
    paymentCity = t12.group(1)
```

```
    payment_city = paymentCity.strip('\"')"
```

```
    print_line()
```

```
    print(payment_city)
```

```
t13 = re.search("'payment_zone_id': (.+?)," , text)
```

```
if t13:
```

```
    paymentZone_id = t13.group(1)
```

```
    payment_zone_id = paymentZone_id.strip('\"')"
```

```
    print_line()
```

```
    print(payment_zone_id)
```

```
t14 = re.search("'payment_zone': (.+?)," , text)
```

```
if t14:
```

```
    payment_zone = t14.group(1)
```

```
    print_line()
```

```
    print(payment_zone)
```

```
t15 = re.search("'payment_zone_code': (.+?)," , text)
```

```
if t15:
```

```
payment_zone_code = t15.group(1)

print_line()

print(payment_zone_code)
```

```
t16 = re.search("'payment_country_id': (.+?)", text)

if t16:

    paymentCountry_id = t16.group(1)

    payment_country_id = paymentCountry_id.strip("''")

    print_line()

    print(payment_country_id)
```

```
t17 = re.search("'payment_country': (.+?)", text)

if t17:

    payment_country = t17.group(1)

    print_line()

    print(payment_country)
```

```
t18 = re.search("'payment_iso_code_2': (.+?)", text)

if t18:

    payment_iso_code_2 = t18.group(1)

    print_line()

    print(payment_iso_code_2)
```

```
t19 = re.search("'payment_iso_code_3': (.+?)", text)

if t19:

    payment_iso_code_3 = t19.group(1)

    print_line()

    print(payment_iso_code_3)
```

```
t20 = re.search("payment_address_format': (.+?)", text)
```

```
if t20:
```

```
    payment_address_format = t20.group(1)
```

```
    print_line()
```

```
    print(payment_address_format)
```

```
t21 = re.search("payment_custom_field': (.+?)", text)
```

```
if t21:
```

```
    payment_custom_field = t21.group(1)
```

```
    print_line()
```

```
    print(payment_custom_field)
```

```
t22 = re.search("payment_method': (.+?)", text)
```

```
if t22:
```

```
    payment_method = t22.group(1)
```

```
    print_line()
```

```
    print(payment_method)
```

```
t23 = re.search("payment_code': (.+?)", text)
```

```
if t23:
```

```
    paymentCode = t23.group(1)
```

```
    payment_code = paymentCode.strip('\"')"
```

```
    print_line()
```

```
    print(payment_code)
```

```
t24 = re.search("shipping_firstname': (.+?)", text)
```

```
if t24:
```

```
    shippingFirstname = t24.group(1)
```

```
    shipping_firstname = shippingFirstname.strip('\"')"
```

```
print_line()

print(shipping_firstname)

t25 = re.search("'shipping_lastname': (.+?)," , text)

if t25:

    shippingLastname = t25.group(1)

    shipping_lastname = shippingLastname.strip('""')

    print_line()

    print(shipping_lastname)

t26 = re.search("'shipping_company': (.+?)," , text)

if t26:

    shipping_company = t26.group(1)

    print_line()

    print(shipping_company)

t27 = re.search("'shipping_address_1': (.+?)," , text)

if t27:

    shippingAddress_1 = t27.group(1)

    shipping_address_1 = shippingAddress_1.strip('""')

    print_line()

    print(shipping_address_1)

t28 = re.search("'shipping_address_2': (.+?)," , text)

if t28:

    shipping_address_2 = t28.group(1)

    print_line()

    print(shipping_address_2)
```

```
t29 = re.search("'shipping_postcode': (.+?)," , text)
```

```
if t29:
```

```
    shippingPostcode = t29.group(1)
```

```
    shipping_postcode = shippingPostcode.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_postcode)
```

```
t30 = re.search("'shipping_city': (.+?)," , text)
```

```
if t30:
```

```
    shippingCity = t30.group(1)
```

```
    shipping_city = shippingCity.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_city)
```

```
t31 = re.search("'shipping_zone_id': (.+?)," , text)
```

```
if t31:
```

```
    shippingZone_id = t31.group(1)
```

```
    shipping_zone_id = shippingZone_id.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_zone_id)
```

```
t32 = re.search("'shipping_zone': (.+?)," , text)
```

```
if t32:
```

```
    shipping_zone = t32.group(1)
```

```
    print_line()
```

```
    print(shipping_zone)
```

```
t33 = re.search("'shipping_zone_code': (.+?)," , text)
```

```
if t33:
```

```
shipping_zone_code = t33.group(1)

print_line()

print(shipping_zone_code)
```

```
t34 = re.search("'shipping_country_id': (.*?)", text)

if t34:

    shippingCountry_id = t34.group(1)

    shipping_country_id = shippingCountry_id.strip("''")

    print_line()

    print(shipping_country_id)
```

```
t35 = re.search("'shipping_country': (.*?)", text)

if t35:

    shipping_country = t35.group(1)

    print_line()

    print(shipping_country)
```

```
t36 = re.search("'shipping_iso_code_2': (.*?)", text)

if t36:

    shipping_iso_code_2 = t36.group(1)

    print_line()

    print(shipping_iso_code_2)
```

```
t37 = re.search("'shipping_iso_code_3': (.*?)", text)

if t37:

    shipping_iso_code_3 = t37.group(1)

    print_line()

    print(shipping_iso_code_3)
```

```
t38 = re.search("'shipping_address_format': (.+?)", text)
```

```
if t38:
```

```
    shipping_address_format = t38.group(1)
```

```
    print_line()
```

```
    print(shipping_address_format)
```

```
t39 = re.search("'shipping_custom_field': (.+?)", text)
```

```
if t39:
```

```
    shipping_custom_field = t39.group(1)
```

```
    print_line()
```

```
    print(shipping_custom_field)
```

```
t40 = re.search("'shipping_method': (.+?)", text)
```

```
if t40:
```

```
    shippingMethod = t40.group(1)
```

```
    shipping_method = shippingMethod.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_method)
```

```
t41 = re.search("'shipping_code': (.+?)", text)
```

```
if t41:
```

```
    shippingCode = t41.group(1)
```

```
    shipping_code = shippingCode.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_code)
```

```
t42 = re.search("'comment': (.+?)", text)
```

```
if t42:
```

```
    comment = t42.group(1)
```

```
print_line()

print(comment)
```

```
t43 = re.search('"total': (.+?),", text)
```

```
if t43:
```

```
    total = t43.group(1)
```

```
    print_line()
```

```
    print(total)
```

```
t44 = re.search('"order_status_id': (.+?),", text)
```

```
if t44:
```

```
    order_status_id = t44.group(1)
```

```
    print_line()
```

```
    print(order_status_id)
```

```
t45 = re.search('"order_status': (.+?),", text)
```

```
if t45:
```

```
    order_status = t45.group(1)
```

```
    print_line()
```

```
    print(order_status)
```

```
t46 = re.search('"affiliate_id': (.+?),", text)
```

```
if t46:
```

```
    affiliate_id = t46.group(1)
```

```
    print_line()
```

```
    print(affiliate_id)
```

```
t47 = re.search('"commission': (.+?),", text)
```

```
if t47:
```



```
commission = t47.group(1)
```

```
print_line()
```

```
print(commission)
```

```
t48 = re.search("'language_id': (.+?)", text)
```

```
if t48:
```

```
    language_id = t48.group(1)
```

```
    print_line()
```

```
    print(language_id)
```

```
t49 = re.search("'language_code': (.+?)", text)
```

```
if t49:
```

```
    language_code = t49.group(1)
```

```
    print_line()
```

```
    print(language_code)
```

```
t50 = re.search("'currency_id': (.+?)", text)
```

```
if t50:
```

```
    currency_id = t50.group(1)
```

```
    print_line()
```

```
    print(currency_id)
```

```
t51 = re.search("'currency_code': (.+?)", text)
```

```
if t51:
```

```
    currency_code = t51.group(1)
```

```
    print_line()
```

```
    print(currency_code)
```

```
t52 = re.search("'currency_value': (.+?)", text)
```

```
if t52:

    currency_value = t52.group(1)

    print_line()

    print(currency_value)

t53 = re.search("'ip': (.+?)", text)

if t53:

    ip = t53.group(1)

    print_line()

    print(ip)

t54 = re.search("'forwarded_ip': (.+?)", text)

if t54:

    forwarded_ip = t54.group(1)

    print_line()

    print(forwarded_ip)

t55 = re.search("'user_agent': (.+?)", text)

if t55:

    user_agent = t55.group(1)

    print_line()

    print(user_agent)

t56 = re.search("'accept_language': (.+?)", text)

if t56:

    accept_language = t56.group(1)

    print_line()

    print(accept_language)
```

```

t57 = re.search("'date_added': (.+?)," , text)

if t57:

    date_added = t57.group(1)

    print_line()

    print(date_added)

# Fetch product_id and quantity by API (check oc_order_product)

htmldata2 =
getdata('http://localhost/phpmyadmin/index.php?route=/sql&db=opencart3&table=oc_order_produ
ct&pos=0')

# Parse the html file

soup2 = BeautifulSoup(htmldata2, 'html.parser')

# Row selection for extracting the data I need from OpenCart database

productID = str(soup2.find_all('td', class_='text-right data grid_edit click2 not_null nowrap'))

'''

print_line()

print(productID)

'''

# Remove the duplicate HTML script tags

product_ID2 = productID.replace('<td class="text-right data grid_edit click2 not_null nowrap"
data-decimals="0" data-type="int">', '')

product_ID3 = product_ID2.replace('</td>', '')

print_line()

print('The product ID3 is: ' + product_ID3)

# Extract product_id and quantity from phpmyadmin

# product_id = product_ID3[8:11]

print_line()

product_ID4 = "

```

```

product_ID4 = product_ID3.split(',')[2]

product_id = product_ID4

print(product_id)

print('product_id is' + product_id)

# quantity = product_ID3[12:14]

print_line()

product_ID5 = "

product_ID5 = product_ID3.split(',')[3]

quantity = product_ID5

print(quantity)

print('quantity is' + quantity)


# Fetch option[] by API (check oc_order_option)

htmldata3 =
getdata('http://localhost/phpmyadmin/index.php?route=/sql&db=opencart3&table=oc_order_option
&pos=0')

# Parse the html file

soup3 = BeautifulSoup(htmldata3, 'html.parser')

# Row selection for extracting the data I need from OpenCart database

Option = str(soup3.find_all('td', class_='text-right data grid_edit click2 not_null nowrap'))

'''

print_line()

print(Option)

'''

# Remove the duplicate HTML script tags

option1 = Option.replace('<td class="text-right data grid_edit click2 not_null nowrap" data-deci-
mals="0" data-type="int">', '')

option2 = option1.replace('</td>', '')

print(option2)

```

```

print_line()

print('The option2 is' + option2)

# Extract all option[] from phpmyadmin

order_product_id_1 = option2.split(',')[1]

print('order_product_id_1 is: ' + order_product_id_1)

order_product_id_2 = option2.split(',')[11]

print('order_product_id_2 is: ' + order_product_id_2)

#Bolt

def newOpt1ValBoltfull(norm):

    print("34/31? bolt red ", norm)

    norm = norm.strip()

    print(len(norm))

    if norm == "34":

        print("ret 1")

        return "1"

    elif norm == "31":

        print("ret 2")

        return "2"

#Bottom

def newOpt2ValBoltfull(norm):

    print("29/28? bottom white", norm)

    print(type(norm))

    norm = norm.strip()

    print(len(norm))

    if norm == "29":

        print("ret 4")

```

```

        return "4"

elif norm == "28":

    print("ret 3")

    return "3"

#Top

def newOpt3ValBoltfull(norm):

    print("27/25? top black ", norm)

    print(len(norm))

    print(type(norm))

    norm = norm.strip()

    print(len(norm))

    if norm == "27":

        print("ret 6")

        return "6"

    elif norm == "25":

        print("ret 5")

        return "5"

# Top

def newOpt4ValBoltless(norm):

    #without bolt

    print(type(norm))

    norm = norm.strip()

    print("35/37 NORM: ", norm)

    if norm == "39":

        print("ret 7")

        return "9"

    elif norm == "40":

```

```
print("ret 8")
```

```
return "10"
```

```
# Bottom
```

```
def newOpt5ValBoltless(norm):
```

```
    norm = norm.strip()
```

```
    print(type(norm))
```

```
    print("39/40 NORM: ", norm)
```

```
    if norm == "35":
```

```
        # boltless black bottom
```

```
        print("ret 9")
```

```
        return "7"
```

```
    elif norm == "37":
```

```
        # boltless white bottom
```

```
        print("ret 10")
```

```
        return "8"
```

```
# With bolt
```

```
if order_product_id_1 == order_product_id_2:
```

```
    # Option for bolt
```

```
    product_option_id_bolt_with_bolt = option2.split(',')[4]
```

```
    print_line()
```

```
    print("34?", product_option_id_bolt_with_bolt)
```

```
    product_option_id_bolt_with_bolt_normalised = newOpt1ValBoltfull(product_option_id_bolt_with_bolt)
```

```
    print(product_option_id_bolt_with_bolt_normalised)
```

```
    #product_option_id_bolt_with_bolt_normalised = 2
```

```
# Option for top
```

```

product_option_id_top_with_bolt = option2.split(',')[9]

print_line()

print(product_option_id_top_with_bolt)

product_option_id_top_with_bolt_normalised = newOpt3ValBoltfull(product_option_id_top_with_bolt)

print(product_option_id_top_with_bolt_normalised)

#product_option_id_top_with_bolt_normalised = 6

# Option for bottom

product_option_id_bottom_with_bolt = option2.split(',')[14]

print_line()

print(product_option_id_bottom_with_bolt)

product_option_id_bottom_with_bolt_normalised = newOpt2ValBoltfull(product_option_id_bottom_with_bolt)

print(product_option_id_bottom_with_bolt_normalised)

#product_option_id_bottom_with_bolt_normalised = 3

#63, boltfull

vals = (product_id, quantity, product_option_id_bottom_with_bolt_normalised, product_option_id_top_with_bolt_normalised, product_option_id_bolt_with_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, shipping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added)

SendData(vals)

```



```

# Without bolt

else:

    # Option for top

    product_option_id_top_without_bolt = option2.split(',')[4]

    print_line()

    print(product_option_id_top_without_bolt)

    product_option_id_top_without_bolt_normalised = newOpt4ValBoltless(product_option_id_top_without_bolt)

    print(product_option_id_top_without_bolt_normalised)

    # Option for bottom

    product_option_id_bottom_without_bolt = option2.split(',')[9]

    print_line()

    print(product_option_id_bottom_without_bolt)

    product_option_id_bottom_without_bolt_normalised = newOpt5ValBoltless(product_option_id_bottom_without_bolt)

    print(product_option_id_bottom_without_bolt_normalised)

#62, boltless

vals = (product_id, quantity, product_option_id_bottom_without_bolt_normalised, product_option_id_top_without_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, shipping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added)

SendData(vals)

```

```
return vals
```

```
# For old OpenCart
```

```
def SendData(values):
```

```
    print(values)
```

```
    #Boltless
```

```
    if len(values) == 62:
```

```
        pid = 2
```

```
        bottom = values[2]
```

```
        top = values[3]
```

```
        shipping_firstname = values[28]
```

```
        shipping_lastname = values[29]
```

```
        shipping_postcode = values[33]
```

```
        shipping_address_1 = values[31]
```

```
        shipping_city = values[34]
```

```
        shipping_country_id = values[38]
```

```
        shipping_zone_id = values[35]
```

```
        payment_firstname = values[9]
```

```
        payment_lastname = values[10]
```

```
        payment_address_1 = values[12]
```

```
        payment_city = values[15]
```

```
        payment_country_id = values[19]
```

```
        payment_zone_id = values[16]
```

```
        payment_postcode = values[14]
```

```
        customer_firstname = values[4]
```

```
        customer_lastname = values[5]
```

```
        customer_email = values[6]
```

```
        customer_telephone = values[7]
```

```
payment_code = values[27]  
  
shipping_code = values[45]  
  
shipping_method = values[44]
```

```
#Boltfull
```

```
elif len(values) == 63:
```

```
    pid = 1  
  
    bottom = values[2]  
  
    top = values[3]  
  
    bolt = values[4]  
  
    shipping_firstname = values[29]  
    shipping_lastname = values[30]  
    shipping_postcode = values[34]  
    shipping_address_1 = values[32]  
    shipping_city = values[35]  
    shipping_country_id = values[39]  
    shipping_zone_id = values[36]  
  
    payment_firstname = values[10]  
    payment_lastname = values[11]  
    payment_address_1 = values[13]  
    payment_city = values[16]  
    payment_country_id = values[20]  
    payment_zone_id = values[17]  
    payment_postcode = values[15]  
  
    customer_firstname = values[5]  
    customer_lastname = values[6]  
    customer_email = values[7]  
    customer_telephone = values[8]
```

```

payment_code = values[28]

shipping_code = values[46]

shipping_method = values[45]

```

```

"""

```

If quantity is 1, then it will go to Kuopio smart factory defaultly.

If quantity is 2, the order will be done separately to both factories.

Quantity over 2 or 0 is not considered since both smart factories Quantity amount is set to tackle the order which quantity is not higher than 2

```

"""

```

```

# Fetch CART content

```

```

username = 'distribution'

```

```

key='c1jGuwOsusvNfOfAekAjZhswUAxwTz8bU7cFicvSYsCjdeLiJMWBWWFKgrPKW0xUmv7ImGcmI-
TUCvZ1z42v3hCKsw1HNqym4POeWUPkHbWc37M843WC5HJX-
uSOI00By1PA2224MsFO65hv1gdvbfrrSnGNmSVk0yfpAZbz4kBdDaJIH4F4SUnqHzOVTu-
gUSIsKgYVJytAbgeu7CDkMsuIiLNmGxbRWTa1h8B6rQa83s1kY4SYeOT1CQniJRpnU2F5'

```

```

# Establishing session for API user by key PARAMS

```

```

session = requests.Session()

```

```

REQUEST = session.post(

    'http://192.168.3.177/shop/index.php?route=api/login',

    data={'username':username, 'key':key}

).text

```

```

# Create a variable for token in each session.post

```

```

print(REQUEST)

```

```

Data = REQUEST.split(",")

```

```

Data = Data[1].split(":")

Data = Data[1][: -1]

print(Data)

Token = Data.replace("\\"", "")

time.sleep(1) # Sleep for 1 second

# print(token)

# Get cart content

cart = session.get(

    'http://192.168.3.177/shop/index.php?route=api/cart/products',

    params={'api_token':Token},

)

print(cart.json())

if pid == 1:

    # Adding product to cart

    A = session.post(

        'http://192.168.3.177/shop/index.php?route=api/cart/add',

        params={'api_token': Token},

        data={

            'product_id': pid,

            'quantity': values[1],

            'option[1]': str(bolt),

            'option[2]': str(bottom),

            'option[3]': str(top),

        }

    )

else:

```

```
A = session.post(
    'http://192.168.3.177/shop/index.php?route=api/cart/add',
    params={'api_token': Token},
    data={
        'product_id': pid,
        'quantity': values[1],
        'option[4]': str(bottom),
        'option[5]': str(top),
    }
)
```

```
print(A.json())
```

```
# Set shipping address for current session
```

```
B = session.post(
    'http://192.168.3.177/shop/index.php?route=api/shipping/address',
    params={'api_token':Token},
    data={
        'firstname':str(shipping_firstname),
        'lastname':str(shipping_lastname),
        'address_1':str(shipping_address_1),
        'city':str(shipping_city),
        'country_id':str(shipping_country_id),
        'zone_id':str(shipping_zone_id),
        'postcode':str(shipping_postcode),
    }
)
```

```
print(B.json())
```

```
# Returning available shipping methods

C = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/methods',

    params={'api_token':Token},

)

print(C.json())

# Set shipping method for current session

D = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/method',

    params={'api_token':Token},

    data={

        'shipping_method':str(shipping_code),

        'title':str(shipping_method),

    }

)

print(D.json())

# Set payment address for this session

E = session.post(

    'http://192.168.3.177/shop/index.php?route=api/payment/address',

    params={'api_token':Token},

    data={

        'firstname':str(payment_firstname),
```

```

        'lastname':str(payment_lastname),

        'address_1':str(payment_address_1),

        'city':str(payment_city),

        'country_id':str(payment_country_id),

        'zone_id':str(payment_zone_id),

        'postcode':str(payment_postcode),

    }

)

print(E.json())

# Returning available payment methods

F = session.post(

    'http://192.168.3.177/shop/index.php?route=api/payment/methods',

    params={'api_token': Token},

)

print(F.json())

# Setting payment method of available in api/payment/methods

G = session.post(

    'http://192.168.3.177/shop/index.php?route=api/payment/method',

    params={'api_token':Token},

    data={

        'payment_method': str(payment_code),

    }

)

```



```
print(G.json())
```

```
# Set customer for current session
```

```
H = session.post(
    'http://192.168.3.177/shop/index.php?route=api/customer',
    params={'api_token':Token},
    data={
        'firstname': str(customer_firstname),
        'lastname':str(customer_lastname),
        'email':str(customer_email),
        'telephone':str(customer_telephone)
    }
)
```

```
print(H.json())
```

```
# New order by cart content and payment/delivery information has been set by current session
```

```
I = session.post(
    'http://192.168.3.177/shop/index.php?route=api/order/add',
    params={'api_token':Token},
    # data={'cart_id':'7'}
)
```

```
print(I.json())
```

```
GetData()
```

APPENDIX 3: THE CODE OF PYTHON SCRIPTS FOR BOTH CITIES' SMART FACTORIES

1. NEW_APP.py

```

import requests

from bs4 import BeautifulSoup

import re

import time


# print the separate line

def print_line():

    print("-" * 50)


# For new OpenCart

def GetData():

    # Establishing session for API user by key PARAMS

    session = requests.Session()


    # Fetch CART content

    username = 'Default'

    key= '6fbqrQfEm-
LYPK1B972bN2fgUTGt7vVLPCpmBJUeW5Ggk6ZvnoM6C627nMjaL3wkUoIkjQR9jLrH-
HeO7HKBwESvOkBKhtTpLmdYdi3tNCGsWq0SHd4ywcrFHnAebvogPdy-
uEIytU8DlwRfg1Zx5aT0QeMXQrCTr55S1crTw9qVe-
OYA1lXl6Eioa4Itmf1JxeooMI37EjuOW8vo5EScyCsPFd3ngg3B7ZsO1tRWKJU0JF9xDfIH122Kybj8vm39
P5t'


    a = session.post(

        'http://127.0.0.1/opencart3/index.php?route=api/login',

        data={'username':username, 'key':key}

```

```
).text
```

```
print(a)
```

```
# print("here")
```

```
# Create a variable for token in each session.post
```

```
data = a.split(",")
```

```
data = data[1]
```

```
data = data.replace("}", "")
```

```
data = data.replace("\\"", "")
```

```
token = data[10::]
```

```
print_line()
```

```
print(token)
```

```
# Print out the latest order and send it if it hasn't been sent before
```

```
# Link for extract html data
```

```
def getdata(url):
```

```
    r = requests.get(url)
```

```
    return r.text
```

```
# Providing the url for reading
```

```
htmldata1 = getdata('http://localhost/phpmyadmin/index.php?route=/sql&db=openkart3&table=oc_order&pos=0')
```

```
# Parse the html file
```

```
soup1 = BeautifulSoup(htmldata1, 'html.parser')
```

```
# Create an empty container
```

```
successOrder = "
```

```

for successOrder in soup1.find('td', class_='text-right data grid_edit click2 not_null nowrap'):

    order_id = successOrder.get_text()

    print_line()

    print(order_id)

# Fetch order info by API

b = session.get(

    'http://127.0.0.1/opencart3/index.php?route=api/order/info',

    params={'api_token':token, 'order_id':order_id},

    data={

    }

)

print_line()

print(b)

print(b.json())

# Extracting all the needed info from b.json() dynamically

text = str(b.json())

t1 = re.search("'firstname': (.+?)", text)

if t1:

    first_name = t1.group(1)

    firstname = first_name.strip('""')

    print_line()

    print(firstname)

```

```
t2 = re.search("lastname': (.+?)," , text)

if t2:

    last_name = t2.group(1)

    lastname = last_name.strip('""')

    print_line()

    print(lastname)


t3 = re.search("email': (.+?)," , text)

if t3:

    Email = t3.group(1)

    email = Email.strip('""')

    print_line()

    print(email)


t4 = re.search("telephone': (.+?)," , text)

if t4:

    Telephone = t4.group(1)

    telephone = Telephone.strip('""')

    print_line()

    print(telephone)


t5 = re.search("custom_field': (.+?)," , text)

if t5:

    custom_field = t5.group(1)

    print_line()

    print(custom_field)


t6 = re.search("payment_firstname': (.+?)," , text)

if t6:
```

```
paymentFirstname = t6.group(1)

payment_firstname = paymentFirstname.strip('')

print_line()

print(payment_firstname)


t7 = re.search("payment_lastname': (.+?)," , text)

if t7:

    paymentLastname = t7.group(1)

    payment_lastname = paymentLastname.strip('')

    print_line()

    print(payment_lastname)


t8 = re.search("payment_company': (.+?)," , text)

if t8:

    payment_company = t8.group(1)

    print_line()

    print(payment_company)


t9 = re.search("payment_address_1': (.+?)," , text)

if t9:

    paymentAddress_1 = t9.group(1)

    payment_address_1 = paymentAddress_1.strip('')

    print_line()

    print(payment_address_1)


t10 = re.search("payment_address_2': (.+?)," , text)

if t10:

    payment_address_2 = t10.group(1)

    print_line()
```

```
print(payment_address_2)
```

```
t11 = re.search("payment_postcode': (.+?)," , text)
```

```
if t11:
```

```
    paymentPostcode = t11.group(1)
```

```
    payment_postcode = paymentPostcode.strip('\"')"
```

```
    print_line()
```

```
    print(payment_postcode)
```

```
t12 = re.search("payment_city': (.+?)," , text)
```

```
if t12:
```

```
    paymentCity = t12.group(1)
```

```
    payment_city = paymentCity.strip('\"')"
```

```
    print_line()
```

```
    print(payment_city)
```

```
t13 = re.search("payment_zone_id': (.+?)," , text)
```

```
if t13:
```

```
    paymentZone_id = t13.group(1)
```

```
    payment_zone_id = paymentZone_id.strip('\"')"
```

```
    print_line()
```

```
    print(payment_zone_id)
```

```
t14 = re.search("payment_zone': (.+?)," , text)
```

```
if t14:
```

```
    payment_zone = t14.group(1)
```

```
    print_line()
```

```
    print(payment_zone)
```

```
t15 = re.search("payment_zone_code': (.+?)," , text)

if t15:

    payment_zone_code = t15.group(1)

    print_line()

    print(payment_zone_code)


t16 = re.search("payment_country_id': (.+?)," , text)

if t16:

    paymentCountry_id = t16.group(1)

    payment_country_id = paymentCountry_id.strip("''")

    print_line()

    print(payment_country_id)


t17 = re.search("payment_country': (.+?)," , text)

if t17:

    payment_country = t17.group(1)

    print_line()

    print(payment_country)


t18 = re.search("payment_iso_code_2': (.+?)," , text)

if t18:

    payment_iso_code_2 = t18.group(1)

    print_line()

    print(payment_iso_code_2)


t19 = re.search("payment_iso_code_3': (.+?)," , text)

if t19:

    payment_iso_code_3 = t19.group(1)

    print_line()
```



```
print(payment_iso_code_3)
```

```
t20 = re.search("payment_address_format': (.+?)," , text)
```

```
if t20:
```

```
    payment_address_format = t20.group(1)
```

```
    print_line()
```

```
    print(payment_address_format)
```

```
t21 = re.search("payment_custom_field': (.+?)," , text)
```

```
if t21:
```

```
    payment_custom_field = t21.group(1)
```

```
    print_line()
```

```
    print(payment_custom_field)
```

```
t22 = re.search("payment_method': (.+?)," , text)
```

```
if t22:
```

```
    payment_method = t22.group(1)
```

```
    print_line()
```

```
    print(payment_method)
```

```
t23 = re.search("payment_code': (.+?)," , text)
```

```
if t23:
```

```
    paymentCode = t23.group(1)
```

```
    payment_code = paymentCode.strip('\"')"
```

```
    print_line()
```

```
    print(payment_code)
```

```
t24 = re.search("shipping_firstname': (.+?)," , text)
```

```
if t24:
```

```
shippingFirstname = t24.group(1)

shipping_firstname = shippingFirstname.strip("""")

print_line()

print(shipping_firstname)
```

```
t25 = re.search("'shipping_lastname': (.+?)," , text)

if t25:

    shippingLastname = t25.group(1)

    shipping_lastname = shippingLastname.strip("""")

    print_line()

    print(shipping_lastname)
```

```
t26 = re.search("'shipping_company': (.+?)," , text)

if t26:

    shipping_company = t26.group(1)

    print_line()

    print(shipping_company)
```

```
t27 = re.search("'shipping_address_1': (.+?)," , text)

if t27:

    shippingAddress_1 = t27.group(1)

    shipping_address_1 = shippingAddress_1.strip("""")

    print_line()

    print(shipping_address_1)
```

```
t28 = re.search("'shipping_address_2': (.+?)," , text)

if t28:

    shipping_address_2 = t28.group(1)

    print_line()
```

```
print(shipping_address_2)
```

```
t29 = re.search("'shipping_postcode': (.+?)," , text)
```

```
if t29:
```

```
    shippingPostcode = t29.group(1)
```

```
    shipping_postcode = shippingPostcode.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_postcode)
```

```
t30 = re.search("'shipping_city': (.+?)," , text)
```

```
if t30:
```

```
    shippingCity = t30.group(1)
```

```
    shipping_city = shippingCity.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_city)
```

```
t31 = re.search("'shipping_zone_id': (.+?)," , text)
```

```
if t31:
```

```
    shippingZone_id = t31.group(1)
```

```
    shipping_zone_id = shippingZone_id.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_zone_id)
```

```
t32 = re.search("'shipping_zone': (.+?)," , text)
```

```
if t32:
```

```
    shipping_zone = t32.group(1)
```

```
    print_line()
```

```
    print(shipping_zone)
```

```
t33 = re.search("'shipping_zone_code': (.+?)", text)
```

```
if t33:
```

```
    shipping_zone_code = t33.group(1)
```

```
    print_line()
```

```
    print(shipping_zone_code)
```

```
t34 = re.search("'shipping_country_id': (.+?)", text)
```

```
if t34:
```

```
    shippingCountry_id = t34.group(1)
```

```
    shipping_country_id = shippingCountry_id.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_country_id)
```

```
t35 = re.search("'shipping_country': (.+?)", text)
```

```
if t35:
```

```
    shipping_country = t35.group(1)
```

```
    print_line()
```

```
    print(shipping_country)
```

```
t36 = re.search("'shipping_iso_code_2': (.+?)", text)
```

```
if t36:
```

```
    shipping_iso_code_2 = t36.group(1)
```

```
    print_line()
```

```
    print(shipping_iso_code_2)
```

```
t37 = re.search("'shipping_iso_code_3': (.+?)", text)
```

```
if t37:
```

```
    shipping_iso_code_3 = t37.group(1)
```

```
    print_line()
```

```
print(shipping_iso_code_3)
```

```
t38 = re.search("'shipping_address_format': (.+?)," , text)
```

```
if t38:
```

```
    shipping_address_format = t38.group(1)
```

```
    print_line()
```

```
    print(shipping_address_format)
```

```
t39 = re.search("'shipping_custom_field': (.+?)," , text)
```

```
if t39:
```

```
    shipping_custom_field = t39.group(1)
```

```
    print_line()
```

```
    print(shipping_custom_field)
```

```
t40 = re.search("'shipping_method': (.+?)," , text)
```

```
if t40:
```

```
    shippingMethod = t40.group(1)
```

```
    shipping_method = shippingMethod.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_method)
```

```
t41 = re.search("'shipping_code': (.+?)," , text)
```

```
if t41:
```

```
    shippingCode = t41.group(1)
```

```
    shipping_code = shippingCode.strip('\"')"
```

```
    print_line()
```

```
    print(shipping_code)
```

```
t42 = re.search("'comment': (.+?)," , text)
```

```
if t42:

    comment = t42.group(1)

    print_line()

    print(comment)

t43 = re.search("'total': (.+?)", text)

if t43:

    total = t43.group(1)

    print_line()

    print(total)

t44 = re.search("'order_status_id': (.+?)", text)

if t44:

    order_status_id = t44.group(1)

    print_line()

    print(order_status_id)

t45 = re.search("'order_status': (.+?)", text)

if t45:

    order_status = t45.group(1)

    print_line()

    print(order_status)

t46 = re.search("'affiliate_id': (.+?)", text)

if t46:

    affiliate_id = t46.group(1)

    print_line()

    print(affiliate_id)
```

```
t47 = re.search('"commission': (.+?),", text)
```

```
if t47:
```

```
    commission = t47.group(1)
```

```
    print_line()
```

```
    print(commission)
```

```
t48 = re.search('"language_id': (.+?),", text)
```

```
if t48:
```

```
    language_id = t48.group(1)
```

```
    print_line()
```

```
    print(language_id)
```

```
t49 = re.search('"language_code': (.+?),", text)
```

```
if t49:
```

```
    language_code = t49.group(1)
```

```
    print_line()
```

```
    print(language_code)
```

```
t50 = re.search('"currency_id': (.+?),", text)
```

```
if t50:
```

```
    currency_id = t50.group(1)
```

```
    print_line()
```

```
    print(currency_id)
```

```
t51 = re.search('"currency_code': (.+?),", text)
```

```
if t51:
```

```
    currency_code = t51.group(1)
```

```
    print_line()
```

```
    print(currency_code)
```

```
t52 = re.search('"currency_value': (.+?),", text)
```

```
if t52:
```

```
    currency_value = t52.group(1)
```

```
    print_line()
```

```
    print(currency_value)
```

```
t53 = re.search('"ip': (.+?),", text)
```

```
if t53:
```

```
    ip = t53.group(1)
```

```
    print_line()
```

```
    print(ip)
```

```
t54 = re.search('"forwarded_ip': (.+?),", text)
```

```
if t54:
```

```
    forwarded_ip = t54.group(1)
```

```
    print_line()
```

```
    print(forwarded_ip)
```

```
t55 = re.search('"user_agent': (.+?),", text)
```

```
if t55:
```

```
    user_agent = t55.group(1)
```

```
    print_line()
```

```
    print(user_agent)
```

```
t56 = re.search('"accept_language': (.+?),", text)
```

```
if t56:
```

```
    accept_language = t56.group(1)
```

```
    print_line()
```



```

print(accept_language)

t57 = re.search('"date_added": (.+?)', text)

if t57:

    date_added = t57.group(1)

    print_line()

    print(date_added)

# Fetch product_id and quantity by API (check oc_order_product)

htmldata2 =
getdata('http://localhost/phpmyadmin/index.php?route=/sql&db=opencart3&table=oc_order_produ
ct&pos=0')

# Parse the html file

soup2 = BeautifulSoup(htmldata2, 'html.parser')

# Row selection for extracting the data I need from OpenCart database

productID = str(soup2.find_all('td', class_='text-right data grid_edit click2 not_null nowrap'))

'''

print_line()

print(productID)

'''

# Remove the duplicate HTML script tags

product_ID2 = productID.replace('<td class="text-right data grid_edit click2 not_null nowrap"
data-decimals="0" data-type="int">', '')

product_ID3 = product_ID2.replace('</td>', '')

print_line()

print('The product ID3 is: ' + product_ID3)

# Extract product_id and quantity from phpmyadmin

# product_id = product_ID3[8:11]

```

```

print_line()

product_ID4 = ""

product_ID4 = product_ID3.split(',')[2]

product_id = product_ID4

print(product_id)

print('product_id is' + product_id)

# quantity = product_ID3[12:14]

print_line()

product_ID5 = ""

product_ID5 = product_ID3.split(',')[3]

product_ID5 = product_ID5[1]

quantity = product_ID5

print(quantity)

print('quantity is' + quantity)

'''

# Fetch option[] by API (check oc_order_option)

htmldata3 =
getdata('http://localhost/phpmyadmin/index.php?route=/sql&db=opencart3&table=oc_order_option
&pos=0')

# Parse the html file

soup3 = BeautifulSoup(htmldata3, 'html.parser')

# Row selection for extracting the data I need from OpenCart database

Option = str(soup3.find_all('td', class_='text-right data grid_edit click2 not_null nowrap'))

'''

print_line()

print(Option)

'''

# Remove the duplicate HTML script tags

```

```
option1 = Option.replace('<td class="text-right data_grid_edit click2 not_null nowrap" data-deci-
mals="0" data-type="int">', " ")
```

```
option2 = option1.replace('</td>', " ")
```

```
print(option2)
```

```
print_line()
```

```
print('The option2 is' + option2)
```

```
# Extract all option[] from phpmysqladmin
```

```
# order_product_id_1 = option2[9:12]
```

```
# order_product_id_2 = option2[49:52]
```

```
order_product_id_1 = option2.split(',')[1]
```

```
print('order_product_id_1 is: ' + order_product_id_1)
```

```
order_product_id_2 = option2.split(',')[11]
```

```
print('order_product_id_2 is: ' + order_product_id_2)
```

```
#Bolt
```

```
def newOpt1ValBoltfull(norm):
```

```
    print("34/31? bolt red ", norm)
```

```
    norm = norm.strip()
```

```
    print(len(norm))
```

```
    if norm == "34":
```

```
        print("ret 1")
```

```
        return "1"
```

```
    elif norm == "31":
```

```
        print("ret 2")
```

```
        return "2"
```

```
#Bottom
```

```
def newOpt2ValBoltfull(norm):
```

```

print("29/28? bottom white", norm)

print(type(norm))

norm = norm.strip()

print(len(norm))

if norm == "29":

    print("ret 4")

    return "4"

elif norm == "28":

    print("ret 3")

    return "3"

```

#Top

```

def newOpt3ValBoltfull(norm):

    print("27/25? top black ", norm)

    print(len(norm))

    print(type(norm))

    norm = norm.strip()

    print(len(norm))

    if norm == "27":

        print("ret 6")

        return "6"

    elif norm == "25":

        print("ret 5")

        return "5"

```

Top

```

def newOpt4ValBoltless(norm):

    #without bolt

    print(type(norm))

```

```

norm = norm.strip()

print("35/37 NORM: ", norm)

if norm == "39":

    print("ret 7")

    return "9"

elif norm == "40":

    print("ret 8")

    return "10"

```

```

# Bottom

```

```

def newOpt5ValBoltless(norm):

    norm = norm.strip()

    print(type(norm))

    print("39/40 NORM: ", norm)

    if norm == "35":

        # boltless black bottom

        print("ret 9")

        return "7"

    elif norm == "37":

        # boltless white bottom

        print("ret 10")

        return "8"

```

```

# With bolt

```

```

if order_product_id_1 == order_product_id_2:

    # Option for bolt

    product_option_id_bolt_with_bolt = option2.split(',')[4]

    print_line()

    print("34?", product_option_id_bolt_with_bolt)

```

```

    product_option_id_bolt_with_bolt_normalised = newOpt1ValBoltfull(product_option_id_bolt_with_bolt)

    print(product_option_id_bolt_with_bolt_normalised)

    #product_option_id_bolt_with_bolt_normalised = 2

    # Option for top

    product_option_id_top_with_bolt = option2.split(',')[9]

    print_line()

    print(product_option_id_top_with_bolt)

    product_option_id_top_with_bolt_normalised = newOpt3ValBoltfull(product_option_id_top_with_bolt)

    print(product_option_id_top_with_bolt_normalised)

    #product_option_id_top_with_bolt_normalised = 6

    # Option for bottom

    product_option_id_bottom_with_bolt = option2.split(',')[14]

    print_line()

    print(product_option_id_bottom_with_bolt)

    product_option_id_bottom_with_bolt_normalised = newOpt2ValBoltfull(product_option_id_bottom_with_bolt)

    print(product_option_id_bottom_with_bolt_normalised)

    #product_option_id_bottom_with_bolt_normalised = 3

    #63, boltfull

    #SendData((product_id, quantity, product_option_id_bottom_with_bolt_normalised, product_option_id_top_with_bolt_normalised, product_option_id_bolt_with_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, ship-

```

```
ping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added))
```

```
vals = (product_id, quantity, product_option_id_bottom_with_bolt_normalised, product_option_id_top_with_bolt_normalised, product_option_id_bolt_with_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, shipping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added)
```

```
SendData(vals)
```

```
# Without bolt
```

```
else:
```

```
# Option for top
```

```
product_option_id_top_without_bolt = option2.split(',')[4]
```

```
print_line()
```

```
print(product_option_id_top_without_bolt)
```

```
product_option_id_top_without_bolt_normalised = newOpt4ValBoltless(product_option_id_top_without_bolt)
```

```
print(product_option_id_top_without_bolt_normalised)
```

```
# Option for bottom
```

```
product_option_id_bottom_without_bolt = option2.split(',')[9]
```

```
print_line()
```

```

    print(product_option_id_bottom_without_bolt)

    product_option_id_bottom_without_bolt_normalised = newOpt5ValBoltless(product_option_id_bottom_without_bolt)

    print(product_option_id_bottom_without_bolt_normalised)

#62, boltless

#SendData((product_id, quantity, product_option_id_bottom_without_bolt_normalised, product_option_id_top_without_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, shipping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added))

    vals = (product_id, quantity, product_option_id_bottom_without_bolt_normalised, product_option_id_top_without_bolt_normalised, firstname, lastname, email, telephone, custom_field, payment_firstname, payment_lastname, payment_company, payment_address_1, payment_address_2, payment_postcode, payment_city, payment_zone_id, payment_zone, payment_zone_code, payment_country_id, payment_country, payment_country, payment_iso_code_2, payment_iso_code_3, payment_address_format, payment_custom_field, payment_method, payment_code, shipping_firstname, shipping_lastname, shipping_company, shipping_address_1, shipping_address_2, shipping_postcode, shipping_city, shipping_zone_id, shipping_zone, shipping_zone_code, shipping_country_id, shipping_country, shipping_iso_code_2, shipping_iso_code_3, shipping_address_format, shipping_custom_field, shipping_method, shipping_code, comment, total, order_status_id, order_status, affiliate_id, commission, language_id, language_code, currency_id, currency_code, currency_value, ip, forwarded_ip, user_agent, accept_language, date_added)

    SendData(vals)

    return vals

# For old OpenCart

```



```
def SendData(values):  
  
    print(values)  
  
    #Boltless  
  
    if len(values) == 62:  
  
        pid = 2  
  
        quantity_for_Kuopio = values[1]  
  
        bottom = values[2]  
  
        top = values[3]  
  
        shipping_firstname = values[28]  
        shipping_lastname = values[29]  
        shipping_postcode = values[33]  
        shipping_address_1 = values[31]  
        shipping_city = values[34]  
        shipping_country_id = values[38]  
        shipping_zone_id = values[35]  
        payment_firstname = values[9]  
        payment_lastname = values[10]  
        payment_address_1 = values[12]  
        payment_city = values[15]  
        payment_country_id = values[19]  
        payment_zone_id = values[16]  
        payment_postcode = values[14]  
        customer_firstname = values[4]  
        customer_lastname = values[5]  
        customer_email = values[6]  
        customer_telephone = values[7]  
        payment_code = values[27]  
        shipping_code = values[45]  
        shipping_method = values[44]
```

```
#Boltfull

elif len(values) == 63:

    pid = 1

    quantity_for_Kuopio = values[1]

    bottom = values[2]

    top = values[3]

    bolt = values[4]

    shipping_firstname = values[29]

    shipping_lastname = values[30]

    shipping_postcode = values[34]

    shipping_address_1 = values[32]

    shipping_city = values[35]

    shipping_country_id = values[39]

    shipping_zone_id = values[36]

    payment_firstname = values[10]

    payment_lastname = values[11]

    payment_address_1 = values[13]

    payment_city = values[16]

    payment_country_id = values[20]

    payment_zone_id = values[17]

    payment_postcode = values[15]

    customer_firstname = values[5]

    customer_lastname = values[6]

    customer_email = values[7]

    customer_telephone = values[8]

    payment_code = values[28]

    shipping_code = values[46]
```

```
shipping_method = values[45]
```

```
"""
```

If quantity is 1, then it will go to Kuopio smart factory defaultly.

If quantity is 2, the order will be done separately to both factories.

Quantity over 2 or 0 is not considered since both smart factories Quantity amount is set to tackle the order which quantity is not higher than 2

```
"""
```

```
# Fetch CART content
```

```
username = 'distribution'
```

```
key='c1jGuwOsusvNfOfAekAjZhswUAxwTz8bU7cFicvSYsCjdeLiJMWBWWFKgrPKW0xUmv7ImGcmI-
TUCvZ1z42v3hCKsw1HNqvm4POeWUPkHbWc37M843WC5HJX-
uSOI00By1PA2224MsFO65hv1gdvbfSrSnGNmSVk0yfpAZbz4kBdDaJIH4F4SUnqHzOVTu-
gUSIsKgYVJytAbgeu7CDkMsuIiLNmGxbRWTa1h8B6rQa83s1kY4SYeOT1CQniJRpnU2F5'
```

```
# Establishing session for API user by key PARAMS
```

```
session = requests.Session()
```

```
REQUEST = session.post(
    'http://192.168.3.177/shop/index.php?route=api/login',
    data={'username':username, 'key':key}
).text
```

```
# Create a variable for token in each session.post
```

```
print(REQUEST)
```

```
Data = REQUEST.split(",")
```

```
Data = Data[1].split(":")
```

```
Data = Data[1][:1]
```

```

print(Data)

Token = Data.replace("\'", "'")

time.sleep(1) # Sleep for 1 second

# print(token)

# Get cart content

cart = session.get(

    'http://192.168.3.177/shop/index.php?route=api/cart/products',

    params={'api_token':Token},

)

print(cart.json())

if pid == 1 and quantity_for_Kuopio == '1':

    # Adding product to cart

    A = session.post(

        'http://192.168.3.177/shop/index.php?route=api/cart/add',

        params={'api_token': Token},

        data={

            'product_id': pid,

            'quantity': str(quantity_for_Kuopio),

            'option[1]': str(bolt),

            'option[2]': str(bottom),

            'option[3]': str(top),

        }

    )

elif pid == 1 and quantity_for_Kuopio == '2':

    A = session.post(

```

```

        'http://192.168.3.177/shop/index.php?route=api/cart/add',

        params={'api_token': Token},

        data={

            'product_id': pid,

            'quantity': '1',

            'option[4]': str(bottom),

            'option[5]': str(top),

        }

    )

elif pid == 2 and quantity_for_Kuopio == '1':

    A = session.post(

        'http://192.168.3.177/shop/index.php?route=api/cart/add',

        params={'api_token': Token},

        data={

            'product_id': pid,

            'quantity': str(quantity_for_Kuopio),

            'option[4]': str(bottom),

            'option[5]': str(top),

        }

    )

else:

    A = session.post(

        'http://192.168.3.177/shop/index.php?route=api/cart/add',

        params={'api_token': Token},

        data={

            'product_id': pid,

            'quantity': '1',

```

```
        'option[4]': str(bottom),

        'option[5]': str(top),

    }

)

print(A.json())

# Set shipping address for current session

B = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/address',

    params={'api_token':Token},

    data={

        'firstname':str(shipping_firstname),

        'lastname':str(shipping_lastname),

        'address_1':str(shipping_address_1),

        'city':str(shipping_city),

        'country_id':str(shipping_country_id),

        'zone_id':str(shipping_zone_id),

        'postcode':str(shipping_postcode),

    }

)

print(B.json())

# Returning available shipping methods

C = session.post(

    'http://192.168.3.177/shop/index.php?route=api/shipping/methods',

    params={'api_token':Token},
```

```
)
```

```
print(C.json())
```

```
# Set shipping method for current session
```

```
D = session.post(
```

```
    'http://192.168.3.177/shop/index.php?route=api/shipping/method',
```

```
    params={'api_token':Token},
```

```
    data={
```

```
        'shipping_method':str(shipping_code),
```

```
        'title':str(shipping_method),
```

```
        # 'cost': '5.00',
```

```
        # 'tax_class_id': '9',
```

```
        # 'text': '5.00€',
```

```
    }
```

```
)
```

```
print(D.json())
```

```
# Set payment address for this session
```

```
E = session.post(
```

```
    'http://192.168.3.177/shop/index.php?route=api/payment/address',
```

```
    params={'api_token':Token},
```

```
    data={
```

```
        'firstname':str(payment_firstname),
```

```
        'lastname':str(payment_lastname),
```

```
        'address_1':str(payment_address_1),
```

```
        'city':str(payment_city),
```

```

        'country_id':str(payment_country_id),

        'zone_id':str(payment_zone_id),

        'postcode':str(payment_postcode),

    }

)

print(E.json())

# Returning available payment methods

F = session.post(

    'http://192.168.3.177/shop/index.php?route=api/payment/methods',

    params={'api_token': Token},

)

print(F.json())

# Setting payment method of available in api/payment/methods

G = session.post(

    'http://192.168.3.177/shop/index.php?route=api/payment/method',

    params={'api_token':Token},

    data={

        'payment_method': str(payment_code),

    }

)

print(G.json())

# Set customer for current session

```



```

H = session.post(
    'http://192.168.3.177/shop/index.php?route=api/customer',
    params={'api_token':Token},
    data={
        'firstname': str(customer_firstname),
        'lastname':str(customer_lastname),
        'email':str(customer_email),
        'telephone':str(customer_telephone)
    }
)

print(H.json())

```

New order by cart content and payment/delivery information has been set by current session

```

I = session.post(
    'http://192.168.3.177/shop/index.php?route=api/order/add',
    params={'api_token':Token},
    # data={'cart_id':'7'}
)

print(I.json())

```

2. connection to Iisalmi.py

```
import requests
```

```

import time

from NEW_APP import *

# Fetch all the data from new opencart
values = GetData()

# quantity for both two smart factories detection
quantity_for_all = values[1]

print('The value of quantity for all is:' ,quantity_for_all)

#Boltless

if len(values) == 62:

    pid_for_Iisalmi = 2

    bottom_for_Iisalmi = values[2]

    top_for_Iisalmi = values[3]

    shipping_firstname_for_Iisalmi = values[28]

    shipping_lastname_for_Iisalmi = values[29]

    shipping_postcode_for_Iisalmi = values[33]

    shipping_address_1_for_Iisalmi = values[31]

    shipping_city_for_Iisalmi = values[34]

    shipping_country_id_for_Iisalmi = values[38]

    shipping_zone_id_for_Iisalmi = values[35]

    payment_firstname_for_Iisalmi = values[9]

    payment_lastname_for_Iisalmi = values[10]

    payment_address_1_for_Iisalmi = values[12]

    payment_city_for_Iisalmi = values[15]

    payment_country_id_for_Iisalmi = values[19]

    payment_zone_id_for_Iisalmi = values[16]

```

```

payment_postcode_for_Iisalmi = values[14]

customer_firstname_for_Iisalmi = values[4]

customer_lastname_for_Iisalmi = values[5]

customer_email_for_Iisalmi = values[6]

customer_telephone_for_Iisalmi = values[7]

payment_code_for_Iisalmi = values[27]

shipping_code_for_Iisalmi = values[45]

shipping_method_for_Iisalmi = values[44]

```

```
#Boltfull
```

```
elif len(values) == 63:
```

```

pid_for_Iisalmi = 1

bottom_for_Iisalmi = values[2]

top_for_Iisalmi = values[3]

bolt_for_Iisalmi = values[4]

shipping_firstname_for_Iisalmi = values[29]

shipping_lastname_for_Iisalmi = values[30]

shipping_postcode_for_Iisalmi = values[34]

shipping_address_1_for_Iisalmi = values[32]

shipping_city_for_Iisalmi = values[35]

shipping_country_id_for_Iisalmi = values[39]

shipping_zone_id_for_Iisalmi = values[36]

payment_firstname_for_Iisalmi = values[10]

payment_lastname_for_Iisalmi = values[11]

payment_address_1_for_Iisalmi = values[13]

payment_city_for_Iisalmi = values[16]

payment_country_id_for_Iisalmi = values[20]

payment_zone_id_for_Iisalmi = values[17]

```

```
payment_postcode_for_Iisalmi = values[15]
```

```
customer_firstname_for_Iisalmi = values[5]
```

```
customer_lastname_for_Iisalmi = values[6]
```

```
customer_email_for_Iisalmi = values[7]
```

```
customer_telephone_for_Iisalmi = values[8]
```

```
payment_code_for_Iisalmi = values[28]
```

```
shipping_code_for_Iisalmi = values[46]
```

```
shipping_method_for_Iisalmi = values[45]
```

```
while quantity_for_all == 2:
```

```
    # Fetch CART content
```

```
    username_2 = 'Iisalmi'
```

```
    key_2 = 'dsPbb35fg7LW8bR0JzO5AK8kBcv5Wz1zvqQkyrUPkStAwYf70BbLz-
```

```
KoG3sOdUOmE4v3YODac4brX8Dsui8Zj1ZCJUoFDWKJ1rR8702bgp383ueFDj3Q3RSWtd42mag-  
FDXjKm31vyemDLgmpQgjRHGCkeaaFLTHZK8YeZrvZrSqvoEf1Rx5jVHL8I6mSoA1WaLzvLkKwJgDuX-  
fOx275oxpzVJLSCTOk4EUmNrsxq7042T162TsNgYz3QgMu2zseT2'
```

```
    # Establishing session for API user by key PARAMS
```

```
    session_2 = requests.Session()
```

```
    req = session_2.post(
```

```
        'http://192.168.2.177/shop/index.php?route=api/login',
```

```
        data={'username':username_2, 'key':key_2}
```

```
    ).text
```

```
    print(req)
```

```
    # Create a variable for token in each session.post
```

```
    data_2 = req.split(",")
```

```

data_2 = data_2[1].split(":")

data_2 = data_2[1][:-1]

print(data_2)

token_2 = data_2.replace("\\\"", "")

time.sleep(1) # Sleep for 1 second

print(token_2)


# Get cart content

cart_2 = session_2.get(

    'http://192.168.2.177/shop/index.php?route=api/cart/products',

    params={'api_token':token_2},

    #data={'product_id': '1'}

)


print(cart_2.json())


# Adding product to cart

a_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/cart/add',

    params={'api_token': token_2},

    # With bolt

    data={

        'product_id': pid_for_Iisalmi,

        'quantity': '1',

        # Bolt

        'option[1]': str(bolt_for_Iisalmi),

        # Bottom Part

```

```

        'option[2]': str(bottom_for_Iisalmi),

        # Top Part

        'option[3]': str(top_for_Iisalmi),

    }

)

print(a_2.json())

# Set shipping address for current session

b_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/shipping/address',

    params={'api_token':token_2},

    data={

        'firstname':str(shipping_firstname_for_Iisalmi),

        'lastname':str(shipping_lastname_for_Iisalmi),

        'address_1':str(shipping_address_1_for_Iisalmi),

        'city':str(shipping_city_for_Iisalmi),

        'country_id':str(shipping_country_id_for_Iisalmi),

        'zone_id':str(shipping_zone_id_for_Iisalmi),

        'postcode': str(shipping_postcode_for_Iisalmi),

    }

)

print(b_2.json())

# Returning available shipping methods

c_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/shipping/methods',

    params={'api_token':token_2},

```

```

)

print(c_2.json())

# Set shipping method for current session

d_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/shipping/method',

    params={'api_token':token_2},

    data={

        'shipping_method': str(shipping_code_for_Iisalmi),

        'title': str(shipping_method_for_Iisalmi),

        # 'cost': '5.00',

        # 'tax_class_id': '9',

        # 'text': '5.00€',

    }

)

print(d_2.json())

# Set payment address for this session

e_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/payment/address',

    params={'api_token':token_2},

    data={

        'firstname': str(payment_firstname_for_Iisalmi),

        'lastname': str(payment_lastname_for_Iisalmi),

        'address_1': str(payment_address_1_for_Iisalmi),

        'city': str(payment_city_for_Iisalmi),

        'country_id': str(payment_country_id_for_Iisalmi),

```

```

        'zone_id': str(payment_zone_id_for_Iisalmi),

        'postcode': str(payment_postcode_for_Iisalmi),

    }

)

print(e_2.json())

# Returning available payment methods

f_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/payment/methods',

    params={'api_token': token_2},

)

print(f_2.json())

# Setting payment method of available in api/payment/methods

g_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/payment/method',

    params={'api_token': token_2},

    data={

        'payment_method': str(payment_code_for_Iisalmi)

    }

)

print(g_2.json())

# Set customer for current session

h_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/customer',

```



```
params={'api_token':token_2},

data={

    'firstname': str(customer_firstname_for_Iisalmi),

    'lastname': str(customer_lastname_for_Iisalmi),

    'email': str(customer_email_for_Iisalmi),

    'telephone': str(customer_telephone_for_Iisalmi)

}

)

print(h_2.json())

# New order by cart content and payment/delivery information has been set by current session

i_2 = session_2.post(

    'http://192.168.2.177/shop/index.php?route=api/order/add',

    params={'api_token':token_2},

    # data={'cart_id': '7'}

)

print(i_2.json())
```