Aleksandr Romanov

Building an automation system for customer acquisition processes

Bachelor's thesis

Information Technology

2017

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Aleksandr Romanov	Bachelor of Engineering, Information Technology	May 2017

Title

Building an automation system for customer acquisition processes

56 pages
3 pages of appendices

Commissioned by

Caterva GmbH

Supervisor

Timo Mynttinen

Abstract

CRM software has been supporting companies in their internal customer acquisition processes for decades. However, in the field of renewable energy storage, which involves the handling of a larger amount of customer related information, traditional CRM systems become less effective and the need for more efficient software is required

The objective of the thesis was to implement the initial state of the automation system for customer acquisition processes, required by the case company in order to remove the future complexity in managing information manually.

The methods used for designing and implementing the system were influenced by a set of predefined software requirements and the existing state of internal CRM software. The thesis described the solutions, including the design of communication mechanism between the external CRM system and an internal one, performing new data creation and updates, utilizing the RESTful APIs provided by Sugar CRM system and finally the development and testing of the automation workflow.

The outcome of the development was a functioning part of the automation system which was performing the required tasks, supporting internal employees. Since the project was focused on the initial important functionality, the potential improvements and extensions were discussed in the thesis.

Keywords

Software Engineering, Sugar CRM, PHP, MySQL, REST, API, OAuth, Linux

CONTENTS

1 INTRODUCTION

Nowadays, software engineering is a one of the most rapidly developing fields. Software products are present in every single aspect of our lives, and most importantly, they serve as a complementary part of our work, making people's and businesses' tasks easier, empowering them to create even greater products and solutions.

Caterva GmbH, a company which provided this thesis opportunity, is an innovative startup, tackling the problems of renewable energy storage. Since the company is relatively young, it is important to address the key problems which might potentially constrain the company's future growth and development. One of such problems is the efficient management of customer acquisition processes. When the company just entered the market, the number of customers was relatively low, which initially made it possible to handle the processes manually. However, the most important startup metric is the ability to grow efficiently and to scale to adjacent markets quickly, which eventually makes a manual process extremely difficult to handle.

The main practical aim of this project is to provide a step by step overview of the core design and engineering decisions, needed to implement the set of software requirements, in particular, a working automation system for customer acquisition processes, which employees can benefit from. From the theoretical perspective, the thesis explains important business concepts such as customer relationship management and customer acquisition. It also gives a general overview of software tools and technologies utilized throughout the development, like the MySQL database, PHP programming language, RESTful APIs and the Sugar CRM software. However, the core idea of the thesis is not to go deeply into every single technology, but to show how multiple software tools can be integrated together for achieving a set goal. Important background, theory, introduction to technologies and systems used are introduced in Chapter 2. Chapter 3 describes project requirements, explains the old process details and the need for a new system design.

In general, the implementation part is divided into three key chapters, which altogether comprise the system's functionality. Chapter 4 focuses on the development of the mechanism, required for the data manipulation and the interaction with an external CRM, which was responsible for communication with new potential customers and provided the information in the form of CSV and zip archives. Chapter 5 concentrates on explaining how to interact with Sugar CRM, by using RESTful API which enables developers to programmatically manipulate information, in this case creating and updating entries and linking necessary documents. In Chapter 6 the implementation of the automation workflow mechanism will be explained, which lets users easily start an acquisition process, by following a sequence of automatically created tasks within Sugar CRM. The chapter also demonstrates how the code can be run automatically with the use of Linux cron job utility which is important for setting up an automated process. Finally, there is a conclusion chapter, where I will summarize my work, describe system's limitations and the possible ways for future improvements and functionality extensions.

2 TECHNOLOGIES

This chapter introduces the reader to the main theoretical concepts, tools, and technologies used throughout the development. It contains the description of what the Customer Relationship Management and Customer Acquisition Processes mean from a business perspective and then moves on explaining the technology stack behind the chosen CRM system in the case company. It will address all the necessary components which the system provides, in order to start the implementation.

2.1 Customer relationship management

According to SalesForce (2016) one of the leading companies on the CRM software market, Customer Relationship Management is a set of practices, strategies, and technologies which help companies in managing the relationships and interactions with their current and potential customers, which leads to improved communication, higher profits and sales growth.

Every company establishes its unique relationships with the customers, which makes the CRM strategies and technologies very specific and customized to the company's needs.

2.1.1 Customer acquisition processes

According to NGDATA website (2016) customer acquisition process is a process which involves the persuasion of a consumer to purchase the company's product. The process usually includes the cycle between making the consumer aware of the available technology (the potential customer is usually called a lead) to converting him into the customer.

However, Caterva GmbH defined customer acquisition processes as a more complex structure, which included the communication with the leads as well as a set of required tasks needed to be done before the lead could be converted into the customer. A more detailed overview will be given in the third chapter (Requirements).

2.1.2 CRM software and types of CRM systems

CRM software is designed to address the challenges in storing and analyzing customer related information to enable businesses to work more efficiently. The number of features, provided by the CRM systems, varies between different implementations and goals which they try to accomplish. According to Wikipedia website (2016a), CRM systems can be divided into three categories introduced below:

- Operational refers to the CRM system which mainly focuses on automating the sales and marketing processes, by providing a convenient user interface for employees to track client related information, past sales and marketing efforts.
- Analytical refers to the CRM system which focuses on analyzing and manipulating customer gathered information in order to better understand clients’ preferences, buying habits and eventually adjust marketing and sales strategies, based on the studied data.
- Collaborative refers to the CRM system which is used to incorporate external stakeholders, like vendors and suppliers into the company’s processes and share the internal customer related data between each other.

All businesses need customers. However, to succeed the company has to build a reliable relationship with them. For this reason, it is required to have customer relationship data, which is stored, automatically updated and always available for the employees. This information comes from different communication channels, such as emails, chats, calls, meetings and social media. By keeping customer related data centralized, it becomes easier to learn about the customers and to successfully address their needs.

2.1.3 CRM technology market

CRM software market is very mature. The list below shows the most popular CRM software providers.

- Microsoft Dynamics CRM
- Oracle

- Salesforce
- SAP AG
- Sugar CRM

The list includes the major technology players, like Microsoft, Oracle, SAP, and Salesforce, that offer their own implementations of CRM software. The company which purchases the license can have several options for managing the installed system. The options are demonstrated in a list below according to TechTarget (2016a).

- On-premises CRM is the solution which gives the company a full control over the system in administration, maintenance and security of the system. The company installs the software on its own servers and controls customer related information which gives an opportunity to extend the system's functionality, fulfilling more complex CRM demands.

- Cloud-based CRM is the solution which provides companies with the same CRM tools as on-premises CRM software. However, the data is stored on remote servers, maintained by the software provider. Therefore, many companies are concerned about the security, since they have no physical control over the data and hardware. In addition, extending the functionality becomes more complicated. A great example of a cloud-based CRM provider is Salesforce.

In the following chapters, I will focus on the capabilities and technology details of the SugarCRM system, which was originally selected by Caterva GmbH as a primary on-premises CRM solution.

2.2 Sugar CRM software stack

SugarCRM is a software company, based in Cupertino, California. The company's main business is a CRM software solution which is also known as SugarCRM (Wikipedia website 2016b)

Figure 1. SugarCRM logo

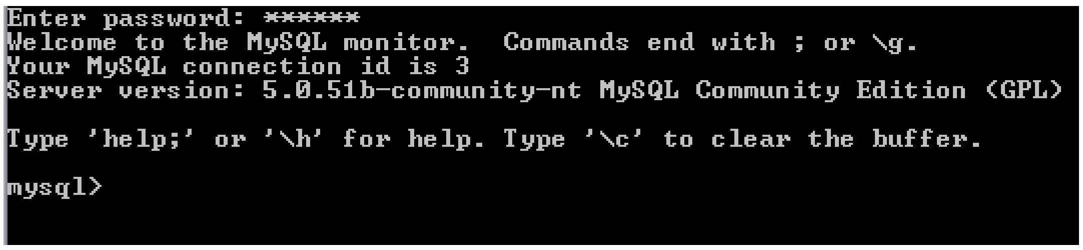
SugarCRM is a full software stack which was originally based on a popular set of technologies, known as a LAMP, which stands for Linux, Apache, MySQL and PHP. However, nowadays, in order to deploy SugarCRM, there is no need in using Apache as a primary server or MySQL as a database. For instance, some company might use Microsoft Internet Information Server instead of Apache, or an Oracle database as an alternative to MySQL. However, PHP still remains the main programming language for the SugarCRM environment. (Wikipedia website 2016c)

Caterva GmbH utilized Linux Debian distribution with a preinstalled Apache web server, PHP interpreter and a MySQL database for the SugarCRM deployment.

2.2.1 PHP, Apache web server and MySQL database

The Apache HTTP web server is an open source software which implements the functionality, needed to serve web content on the Internet. It is believed to be the most widely used web server solution nowadays, powering more than 100 million websites. (Wikipedia website 2016i)

MySQL is an open source database management system for creating and working with SQL databases. Since the project is open for contributions, it makes the software very popular among top IT companies. MySQL provides a command line interface for administrating the databases by using the SQL language. However, for more convenient work, there is a possibility to install a graphical program, known as MySQL Workbench, which provides the graphical user interface for managing the data. (Wikipedia website 2016e)



```
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.0.51b-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Figure 2. MySQL command line interface

According to Wikipedia (2016d) PHP is a programming language which has been in the industry of web development for more than 21 years. It was specifically designed for the web in order to address the challenges in making the web more dynamic. Even though the language is primarily used for building web applications, it can also be used as a general purpose programming language.

In order to execute a PHP source code, it is required to install a PHP interpreter, which makes the language interpreted, meaning that the source code is interpreted while the program is running, without needing to compile it. Linux OS has a PHP interpreter installed initially.

2.2.2 HTTP protocol

HTTP (Hypertext Transfer Protocol) is a fundamental protocol of the web. It defines the rules for communication between the client and server computers. Every time a user requests a web page in his client application, like a web browser, he utilizes HTTP. Based on Wikipedia (2016f) the HTTP implementation defines a set of HTTP methods, also known as HTTP verbs, which describe what action should be performed on the requested information. The list most popular HTTP methods, which were frequently used during the project development, is given below.

- GET returns the requested data from the server
- POST indicates that the server should accept a piece of data sent from the client (for example web form)
- PUT the message may be used similarly to POST method, however, it can also perform updates on data entries on the server
- DELETE tells the server to remove the resource permanently

Originally, the HTTP protocol was designed to let computers on the network request static content like HTML documents and images from the web server software. Web server needed to read the HTTP request and determine which content the client requested and then the server could look up for a file in a directory which was set for the server. In the end, the server prepared the HTTP response either containing the data or a notification if the file was not found. Since then HTTP has evolved into a more robust technology, allowing clients to request not only HTML pages and static images, but also specific data formats like JSON or XML documents, which made it possible to develop web APIs.

2.2.3 REST architecture

REST stands for Representational State Transfer and it is a software architecture approach. A software system which is based on the REST architecture is usually referred to as a RESTful application. Based on Service Architecture website (2016) a software system can be characterized as RESTful if:

- the state of the system is divided into distributed resources
- each resource can be accessed, using a set of command, like HTTP methods.

Nowadays, the RESTful architecture principal has become popular in designing web services, because RESTful systems work flawlessly with the HTTP protocol. The REST approach is a number one choice, when it comes to designing application programming interfaces, especially for web applications. SugarCRM was built with the RESTful architecture in mind, which makes it possible to interact with the system programmatically, using HTTP verbs.

2.2.4 RESTful API

According to TechTarget (2016b) API (Application Programming Interface) is a software interface that has a form of computer program, allowing multiple software systems to interact with each other programmatically. APIs are widely used in the field of software engineering. Whether developer needs to interact with operating system, hardware component or another web application, an appropriate API is used

When a software engineer designs web services, he wants other developers to access the data, using an application programming interface. By using the HTTP protocol to retrieve the information from a web service a developer makes an API RESTful, hence RESTful API. (TechTarget website 2016c)

Nowadays, top IT companies like Facebook, Twitter or Pinterest have their own web services, allowing other developers to interact with the company's data by using RESTful APIs.

2.2.5 OAuth protocol

The web services are designed to provide developers with unique information to work with. For example, Facebook might provide a web service for third-party developers, allowing them to interact with Facebook's user data, like pictures, friends or meetings. These services empower third party applications, by letting them process the information and present new experiences on top of the existing user base. To make it happen Facebook built RESTful APIs.

However, there is a security concern about this workflow. What if the user does not want to grant an access to his personal data to a third party application? How to make sure that the application, requesting user's details from Facebook is actually what it claims to be and not a malicious piece of software, trying to steal private information? To address the problems explained above, OAuth protocol was developed.

OAuth protocol provides a secure mechanism to enable third-party applications to request personal data on behalf of a user. In addition, the protocol guarantees that the developer registered his application on service provider's servers, by giving his app id. When an application has been successfully registered, a service provider gives to a client a secret key, which will be used for authenticating the app, requesting user's data.

The mechanism consists of multiple steps to make sure all parties confirmed their identity. First, it is important to understand the notions, involved into the process: (IETF tools website, 2012)

- Client is a third-party application, requesting the user-specific data from a web service provider.
- Resource Owner is a user, who has his data stored on web service provider's data center (for example a Twitter user)
- Authorization server is a server, belonging to web service provider, designed specifically for authenticating a third party application.
- Resource server is a server, belonging to web service provider, storing user's data (for example Facebook's servers)
- Access Token is a piece of information (similar to a digital key) which is granted by the Authorization server. The token is used subsequently by a client, in order to access data on Resource server.

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |        Authorization Grant &  +---------------+
|        |--(C)--- Client Credentials -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 3. OAuth workflow

A diagram above (Figure 3) describes a workflow visually. First, a client (application) asks a resource owner (user) if it can access the information stored on resource servers (for example a Facebook server). The important thing to note is that user does not need to give its username and password to a third party app. The only thing step (A) does is asking if it could access a certain piece of information, which will be limited by an API provider. In step (B) a user either accepts or declines the request. If a user accepts the request then an application has to prove to an authorization server, that it is indeed an app, which was registered to use an API. In this step app id and secret key come into play, which was given by a web service provider after the app was registered (C). If a secret key is wrong, then access token would not be granted, resulting in a cancellation of information request. On the other hand, if an app has successfully authenticated itself, by giving the correct credentials, an Access token is sent to a client (D) which makes it possible for an application to request the data from Resource servers (E, F).

In the following chapters (SugarCRM API) I will show how the protocol was utilized, when working with SugarCRM RESTful APIs, in order to securely communicate with the system.

2.3 Linux operating system and a cron job utility

Linux is an open source operating system kernel software, which has gained a large popularity among developer’s community, bringing new software products and companies on the market. Because of the fact that a kernel is an open source, everyone can contribute to the development and create his own specific version of it.

Linux comes in multiple versions, which are known as Linux distributions. Different distributions are based on Linux kernel and designed for specific purposes, for example, Ubuntu distribution is a popular desktop OS, Android is a major mobile OS and OpenWrt, the embedded OS, primarily used for network hardware devices. In addition, Linux powers most servers on the Internet, as well as supercomputers, according to Wikipedia (2016g).

In order to deploy a SugarCRM system at Caterva GmbH, Linux Debian distribution was used. The company has two Linux servers, which are designed to host development and testing systems separately.

Development sandbox is a Linux server, which contains a testing version of the SugarCRM. All the development was done, using the testing server and after the features were added and tested, they were deployed to a production system. The separation was extremely important in keeping productive environment constantly running. In case a new feature failed to work or broke the entire system, we could always restore the testing environment and identify the problem, before it happens on a real system.

Linux operating system comes with a remarkable set of tools and utilities which perform a variety of tasks. One of such utilities is a **cron job** which makes it possible to create a scheduled script executions. Such functionality becomes extremely handy for building the automated workflow (Wikipedia website 2016h).

3 REQUIREMENTS

In this chapter I will describe the requirements for a new system design and explain an old process, pointing out the details of why the changes were needed. Before describing the processes, it is important to understand how SugarCRM stores the entries about leads, linked tasks or documents and how the system is capable of recognizing when a user makes changes to SugarCRM UI, leading to the execution of some logic.

3.1 SugarCRM modules

SugarCRM stores the information about potential customers, related tasks, documents or calls in specific sections, called modules. Below you can see an image of the navigation bar UI, showing different modules.

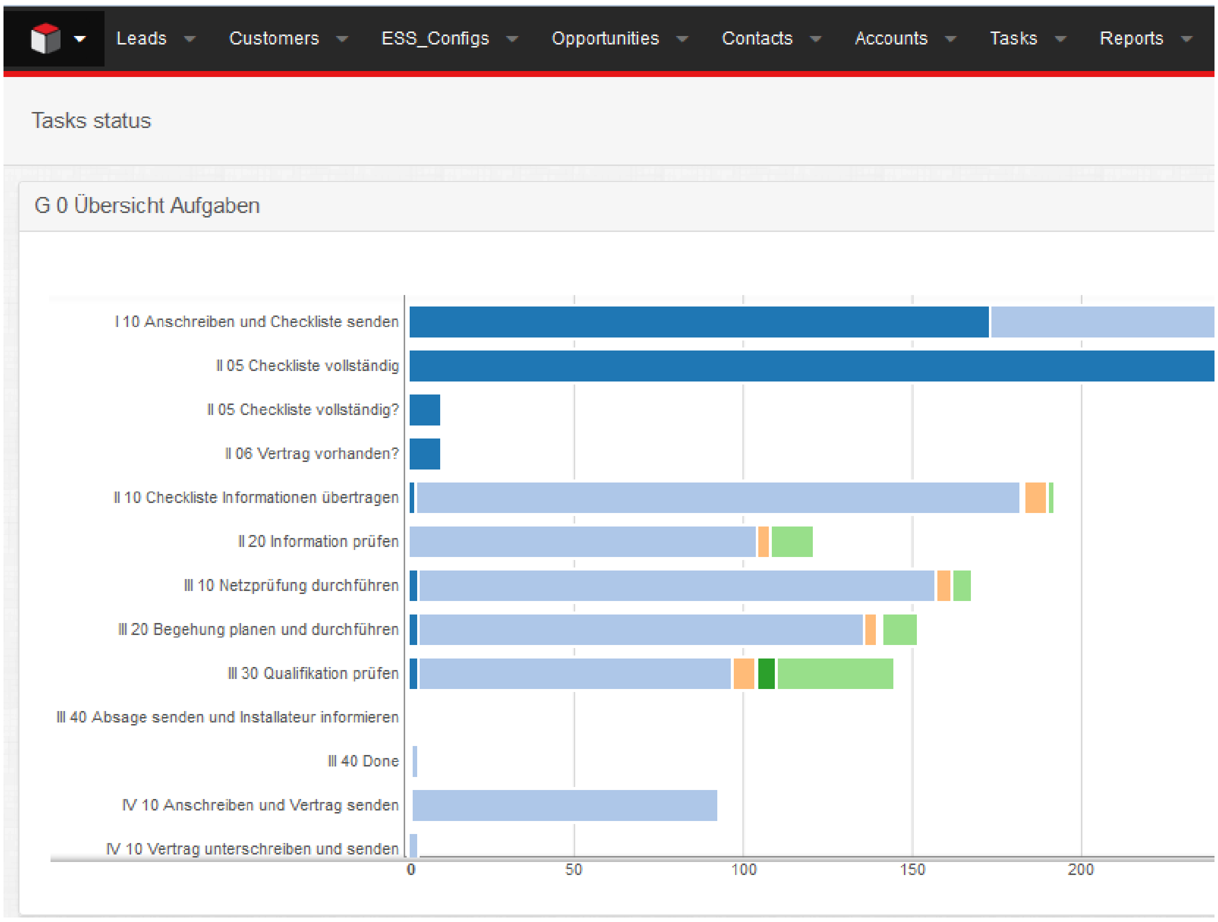
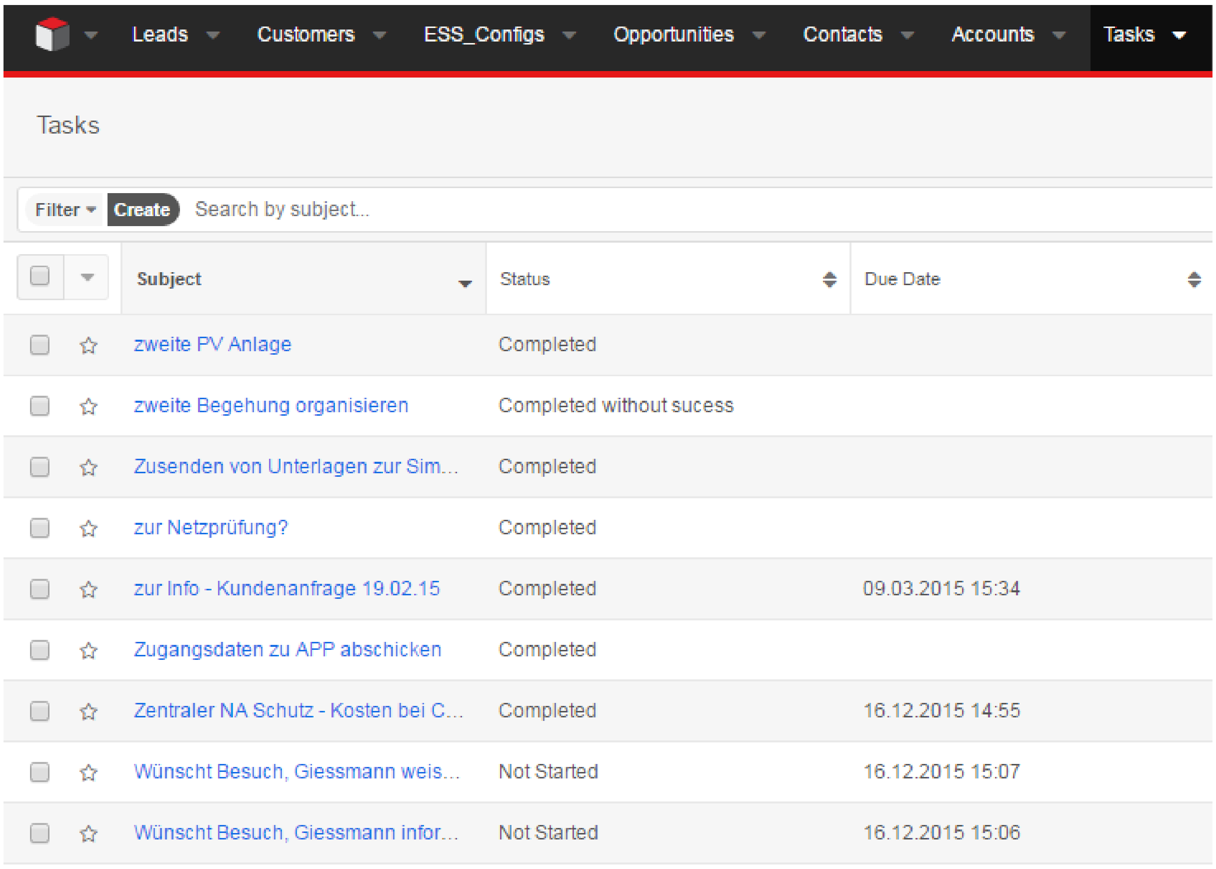

Leads
Customers
ESS_Configs
Opportunities
Contacts
Accounts
Tasks
Reports
Tasks status
G 0 Übersicht Aufgaben
I 10 Anschreiben und Checkliste senden
II 05 Checkliste vollständig
II 05 Checkliste vollständig?
II 06 Vertrag vorhanden?
II 10 Checkliste Informationen übertragen
II 20 Information prüfen
III 10 Netzprüfung durchführen
III 20 Begehung planen und durchführen
III 30 Qualifikation prüfen
III 40 Absage senden und Installateur informieren
III 40 Done
IV 10 Anschreiben und Vertrag senden
IV 10 Vertrag unterschreiben und senden
0
50
100
150
200

Figure 4. SugarCRM Navigation UI

By clicking on one of the modules in the navigation bar will bring a user to a detailed module view, giving us a list of existing entries.



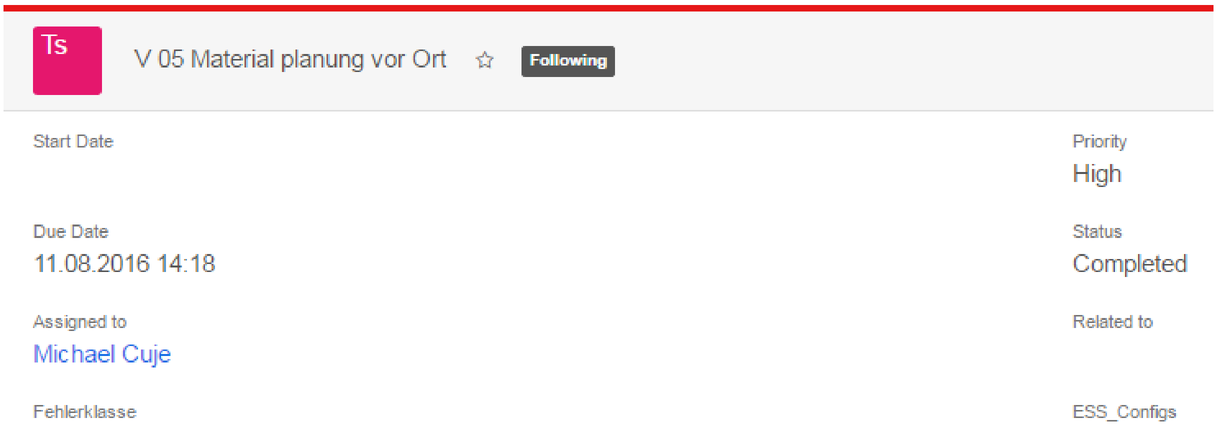
Tasks

Filter Create Search by subject...

Subject	Status	Due Date
zweite PV Anlage	Completed	
zweite Begehung organisieren	Completed without sucess	
Zusenden von Unterlagen zur Sim...	Completed	
zur Netzprüfung?	Completed	
zur Info - Kundenanfrage 19.02.15	Completed	09.03.2015 15:34
Zugangsdaten zu APP abschicken	Completed	
Zentraler NA Schutz - Kosten bei C...	Completed	16.12.2015 14:55
Wünscht Besuch, Giessmann weis...	Not Started	16.12.2015 15:07
Wünscht Besuch, Giessmann infor...	Not Started	16.12.2015 15:06

Figure 5. SugarCRM Tasks module detailed view

The picture above (Figure 5) provides an overview of how different Task items are arranged within a Tasks module. Clicking on a particular entry will bring us to a specific Task overview.


Ts V 05 Material planung vor Ort Following

Start Date

Priority
High

Due Date
11.08.2016 14:18

Status
Completed

Assigned to
Michael Cuje

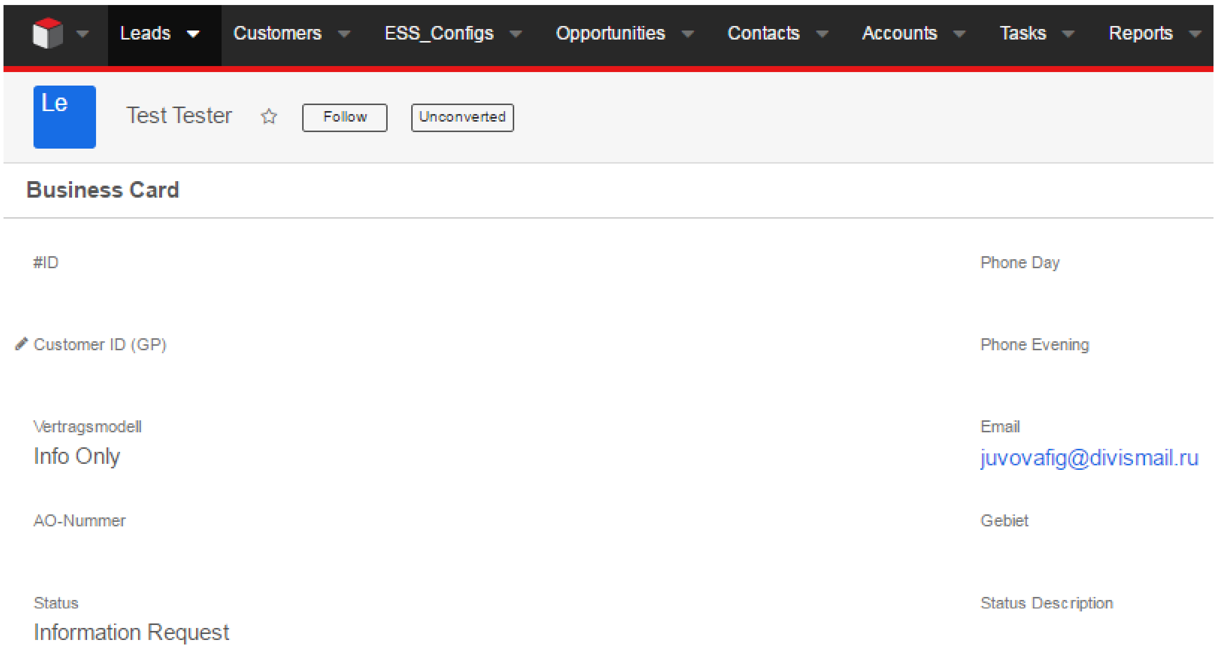
Related to

Fehlerklasse

ESS_Configs

Figure 6. Detailed Task item view

A Task item contains a status field which can be changed by a SugarCRM user (Figure 6). The picture also demonstrates an “Assigned to” field, which specifies who is responsible for the task.



Test Tester

Business Card

#ID	Phone Day
Customer ID (GP)	Phone Evening
Vertragsmodell Info Only	Email juvovafig@divismail.ru
AO-Nummer	Gebiet
Status Information Request	Status Description

Figure 7. Lead item detailed view

Using the same way we used, accessing the Task item, we can select a Lead entry, which contains a set of information about the Lead, like name, email, phone and status. A Status field can also be changed the same way it was done in Task item (Figure 7).

SugarCRM provides the logical software behaviour, associated with the user's actions within the system. For example, it is possible to change a lead status field which will trigger a logic, creating a couple of new tasks. Additionally, it is possible to link documents, tasks, and notes to a module item. For instance, a Task item can have a couple of notes or documents associated with it. The idea of linking data to module items as well as the functionality of the logic, triggered after the user makes an action, will be described in the following implementation chapters.

3.2 Old process overview

Before new requirements were introduced, Caterva GmbH relied on a simplified version of a customer acquisition process in which some tasks had to be done manually and some steps were not centralized with the SugarCRM system.

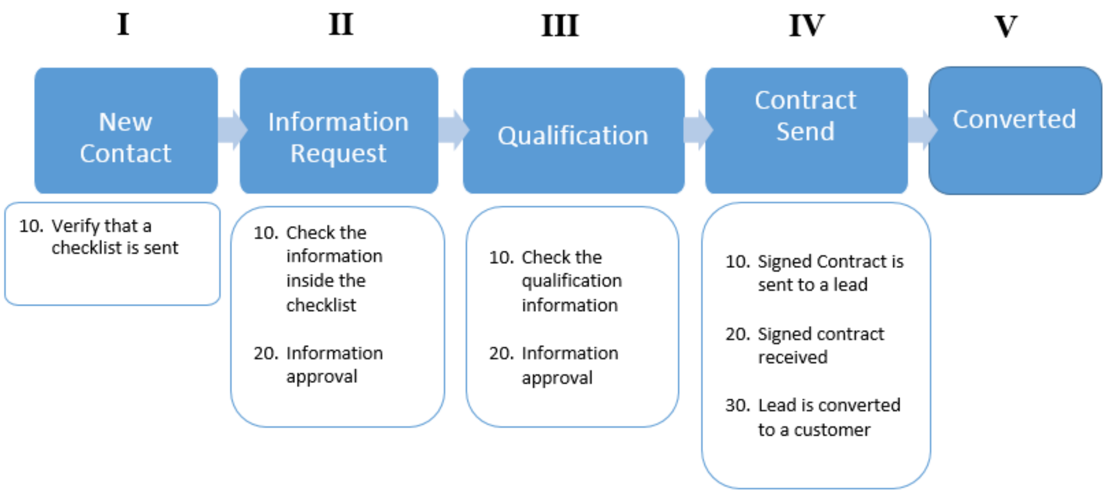
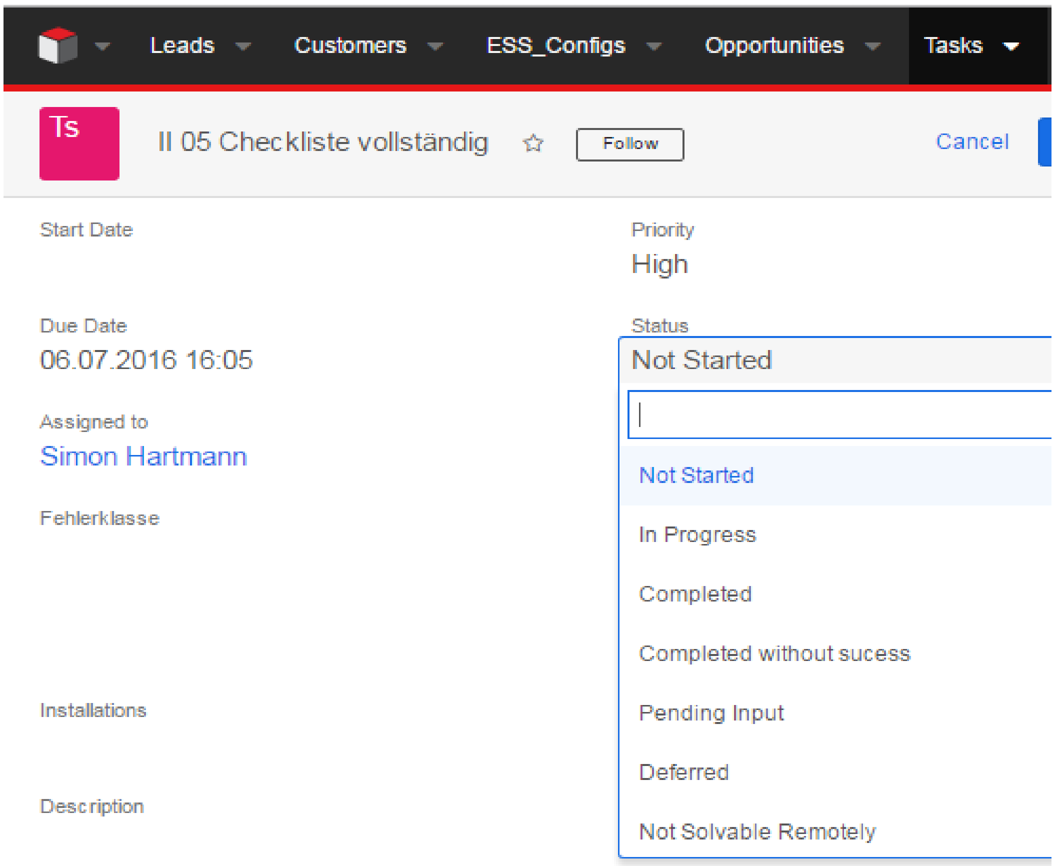

I
II
III
IV
V
New Contact
Information Request
Qualification
Contract Send
Converted
10. Verify that a checklist is sent
10. Check the information inside the checklist
20. Information approval
10. Check the qualification information
20. Information approval
10. Signed Contract is sent to a lead
20. Signed contract received
30. Lead is converted to a customer

Figure 8. Original process workflow

A diagram of an old customer acquisition process workflow is shown in the image above (Figure 8). The numbers *I* to *V* indicate the steps. The names in the blue boxes specify the name of a Lead item status, and boxes with blue border indicate the linked tasks to a Lead. The names for task statuses are in the picture below (Figure 9).



II 05 Checkliste vollständig

Start Date

Priority
High

Due Date
06.07.2016 16:05

Status
Not Started

Assigned to
Simon Hartmann

Fehlerklasse

Not Started
In Progress
Completed
Completed without sucess
Pending Input
Deferred
Not Solvable Remotely

Installations

Description

Figure 9. Example of the task statuses

Based on Figure 9 we can examine the process in more detail:

- First, the information has to be entered into SugarCRM manually, in order to create a lead entry (Figure 7). After the Lead has been created and saved, the SugarCRM logic triggers the task creation, which is linked to a Lead entry. A Lead status is automatically set to “New Contact”. A linked task name is “I 10 Verify that a checklist is sent”, indicating that a task assigned a user (Figure 8) has to process it.
- After accomplishing a task, by setting its status to ”Completed” (Figure 9), the lead status can be changed to ”Information Request”, which automatically creates two tasks ”II 10 Check the information inside the checklist” and “II 20 Information approval”. This means that after checking the data inside the checklist it has to be approved by an employee. After two tasks are done, the lead status can be changed to “Qualification”.
- In “Qualification” status a user has to process two more tasks, the description of which are in the (Figure 8). After tasks are successfully closed, the lead status can be changed to “Contract Send”.
- Having done all the tasks related to “Contract Send” status, a lead can be converted to a customer, by setting his status to “Converted”

The main disadvantage of the old process was scalability. As the number of leads has increased, information typing manually slowed the process dramatically. In addition, Caterva GmbH outsourced the communication with the potential customers to an external company, which has changed the process at its root.

3.3 New process requirements

For a new process, Caterva GmbH provided a clear requirement which had to be transformed into a software system. The requirements were defined as follows:

- Communication with leads has to be outsourced to an external company called ARANES
- ARANES stores the information about the leads in their own CRM and sends regular CSV data imports with lead-related data to Caterva GmbH email address.
- ARANES provides a unique five digit lead identifier inside the CSV file.
- The system should be capable of regularly checking the email inbox, detecting a newly delivered CSV file, and parsing it.
- The system must be able to automatically create lead entries inside SugarCRM, based on the information in a CSV file.
- ARANES should send a checklist in the form of a zip archive to Caterva email inbox
- The system should be able to detect the checklist presence and link it to an appropriate lead within SugarCRM.

- Appropriate tasks should be created and linked to lead entries.
- Changing the task or lead statuses must trigger internal SugarCRM process, navigating the workflow.

The general requirements were divided into groups which represent different independent functionality parts.

3.3.1 ARANES external CRM system

ARANES has provided a web interface to their CRM system for Caterva GmbH so that we could understand how information is stored there and how the CSV structure was organized. Even though a new software system should not communicate with their CRM directly, the access to the system was essential for testing, especially for downloading CSV files, sending it to Caterva email. By having the ability to do these steps, I could simulate the beginning of the process.

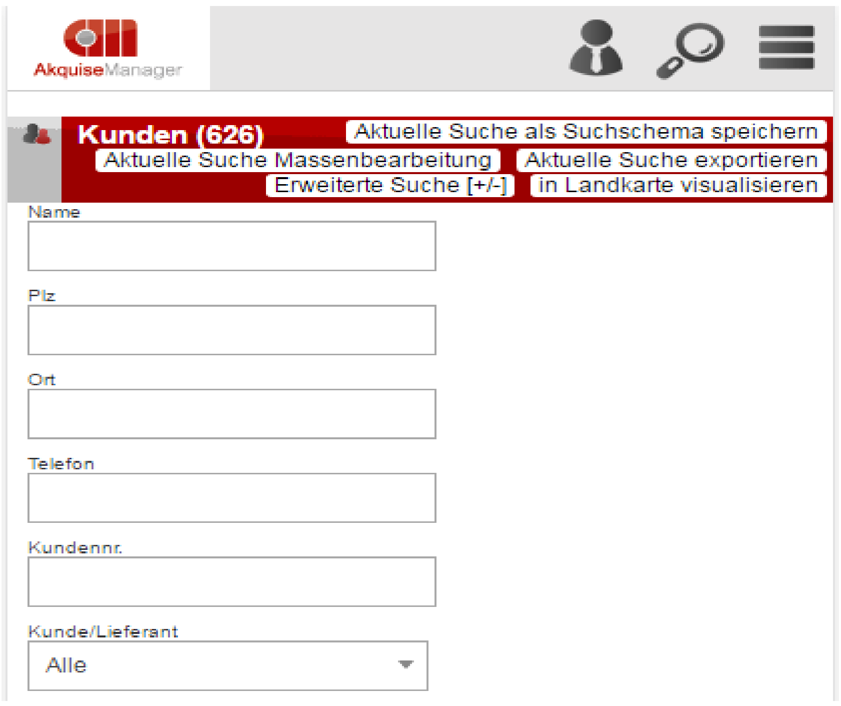

AkquiseManager
Kunden (626)
Aktuelle Suche als Suchschema speichern
Aktuelle Suche Massenbearbeitung
Aktuelle Suche exportieren
Erweiterte Suche [+/-]
in Landkarte visualisieren
Name
Plz
Ort
Telefon
Kundennr.
Kunde/Lieferant
Alle

Figure 10. ARANES web interface

3.3.2 Data handling

For successful data handling it was required to build the software parts, capable of:

- Scanning Caterva email inbox
- Detecting email with CSV imports or checklists with zip files
- Extracting the files, parsing CSV information about the customers, and preparing it for SugarCRM system
- After the data is prepared, it should be posted to SugarCRM, using RESTful API, which is provided by Sugar.
- If information already exists, it has to be updated, using Sugar REST APIs
- Checklist zip files must be linked to appropriate lead entries, which were created from CSV file, also using Sugar RESTful APIs

Data handling part of the requirements include the design of custom functions for data processing and the implementation of a communication system which delivers the data to SugarCRM.

3.3.3 Workflow automation

In the previous chapters, I mentioned internal SugarCRM logic many times. This is functionality, which is provided by SugarCRM, in which it is capable of listening to events, happening inside the system and act accordingly. For instance, when a task status is changed, call a function, creating another task, or when a document was linked, create a note object. The requirements for the software system included several internal logic behaviours:

- When a lead’s status is changed, trigger a logic, creating new tasks
- When a task’s status is changed, perform validation, making sure that some tasks are successfully closed, so that new ones could be created

Internal SugarCRM logic and real application examples are discussed in the 6th chapter. Appendix 1 and Appendix 2 are available for a visual presentation of the requirements and the full software system specification.

4 INTERACTION WITH EXTERNAL CRM

This is the first part of the implementation section in which I will explain how I analyzed and implemented the part of the requirement, in particular:

- Scanning Caterva email inbox, searching for CSV and checklist.

- Parsing CSV data and preparing appropriate data structure for SugarCRM

4.1 Reading emails with PHP

As it was discussed in Technologies chapter, PHP is the primary programming language for SugarCRM. PHP, like most other languages, has a very strong community of supporters, who created a very useful set of functions, which is known as a library.

PHP provides an IMAP library, which deals with the connection to an email inbox, fetching emails and attachments. The library had already been installed before I started on a project.

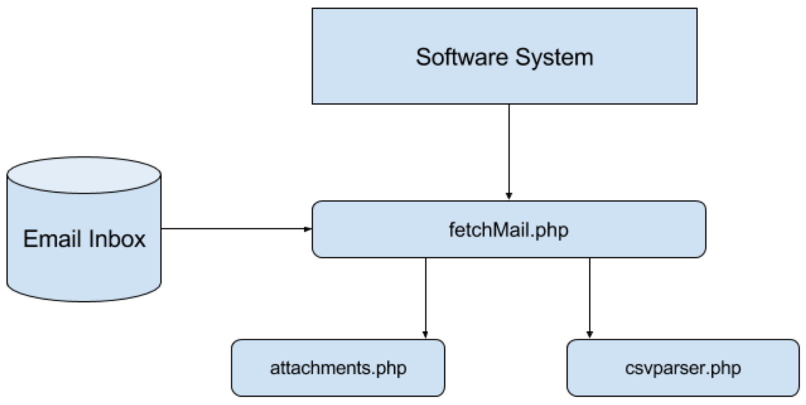
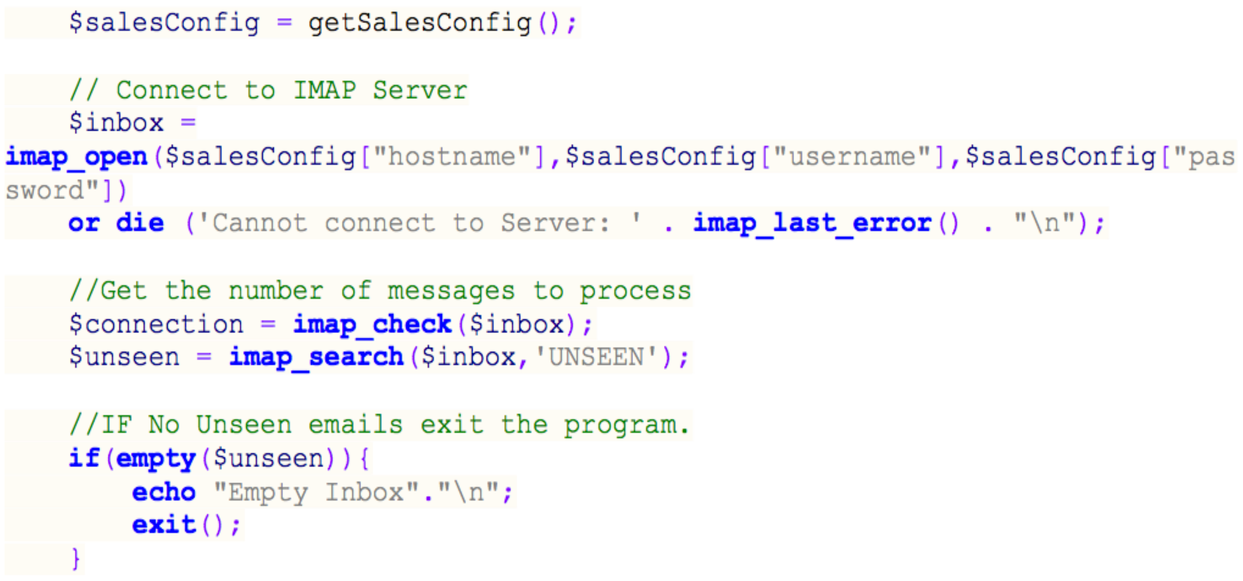


Figure 11. Initial structure of the software system

In the picture above (Figure 11) you can see the initial system’s structure:

- **fetchMail.php** is an entry point of the project, in which PHP code gets executed, scanning email inbox and detecting if there is a checklist or a CSV file with the leads information, sent from ARANES system.
- **attachments.php** is a set of functions, provided by Caterva GmbH software developers, which was handling the extraction of CSV or zip attachments from ARANES emails.
- **csvparser.php** contains the code related to parsing the CSV information and preparing the custom data structure, which is then used for SugarCRM lead entry creation

Now we can take a look at some code, explaining how the inbox is being scanned and the basic configurations to make use of the IMAP library.

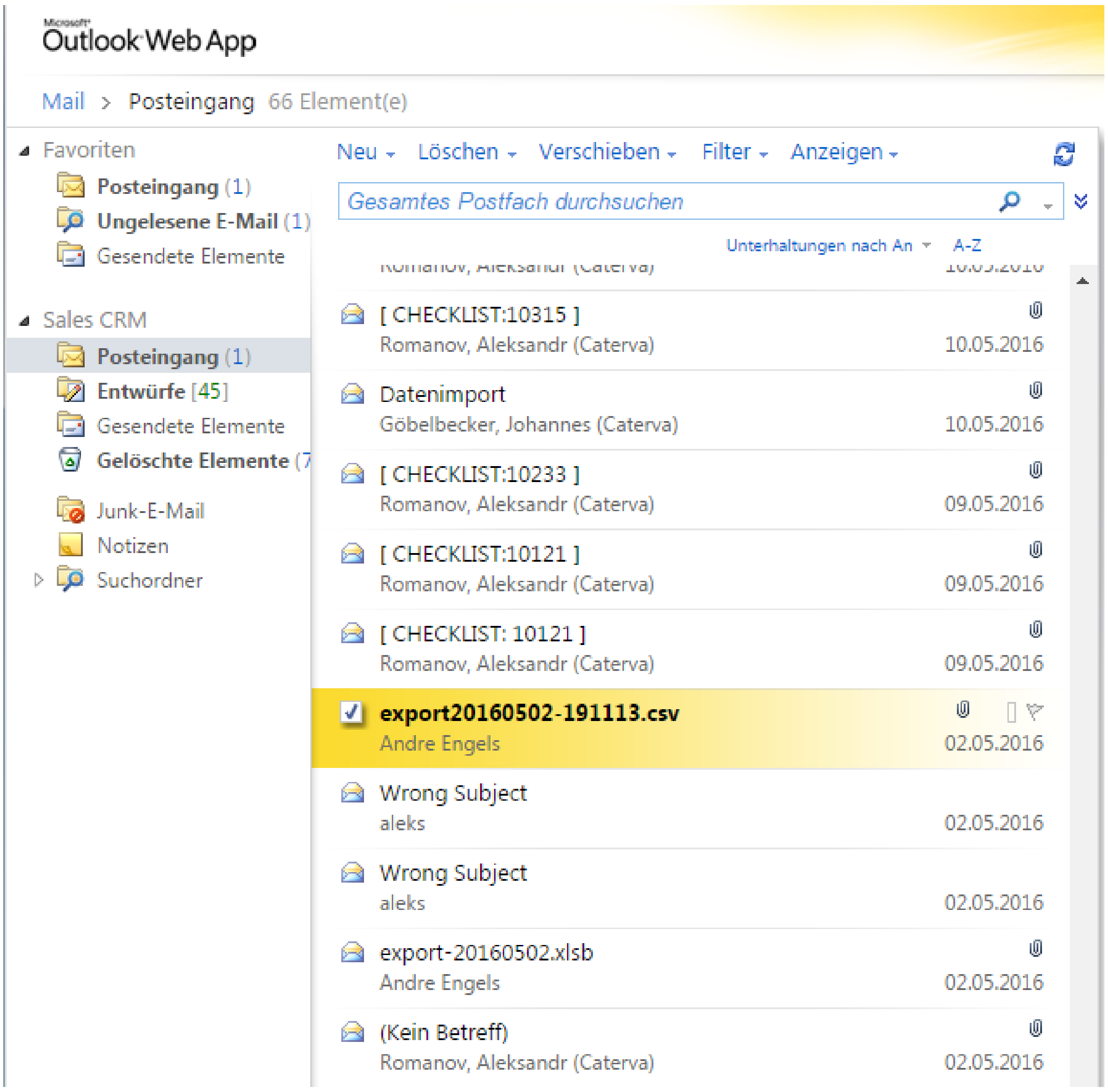


```
    $salesConfig = getSalesConfig();

    // Connect to IMAP Server
    $inbox =
imap_open($salesConfig["hostname"],$salesConfig["username"],$salesConfig["pas
sword"])
    or die ('Cannot connect to Server: ' . imap_last_error() . "\n");

    //Get the number of messages to process
    $connection = imap_check($inbox);
    $unseen = imap_search($inbox,'UNSEEN');

    //IF No Unseen emails exit the program.
    if(empty($unseen)){
        echo "Empty Inbox"."\n";
        exit();
    }
```

Code 1. IMAP library configuration

The code above shows (Code 1) how the connection to the inbox is made:

- First, the code receives the configuration for the email server, storing it in a **$salesConfig** variable. Configuration is received from another file, which contains passwords and username.
- Next step is to construct an inbox object, which represents the connection to an email server. It is achieved by calling **imap_open** function, which receives all configuration parameters from the configuration object. If the connection is successful, then the code will be executed further, otherwise **die** function terminates the execution, giving the error to a programmer.
- As you can see from the code, getting the number of unread emails is relatively easy, it is sufficient to call the **imap_search** function with the inbox object and **UNSEEN** flag, what return a number of newly arrived messages
- If the number is 0, then it indicates that the inbox is empty and the code exits execution, otherwise the message must be processed.



Outlook Web App

Mail > Posteingang 66 Element(e)

Favoriten
- Posteingang (1)
- Ungelesene E-Mail (1)
- Gesendete Elemente

Sales CRM
- Posteingang (1)
- Entwürfe [45]
- Gesendete Elemente
- Gelöschte Elemente (7
- Junk-E-Mail
- Notizen
- Suchordner

Neu Löschen Verschieben Filter Anzeigen

Gesamtes Postfach durchsuchen

Unterhaltungen nach An A-Z

Subject	From	Date
[CHECKLIST:10315]	Romanov, Aleksandr (Caterva)	10.05.2016
Datenimport	Göbelbecker, Johannes (Caterva)	10.05.2016
[CHECKLIST:10233]	Romanov, Aleksandr (Caterva)	09.05.2016
[CHECKLIST:10121]	Romanov, Aleksandr (Caterva)	09.05.2016
[CHECKLIST: 10121]	Romanov, Aleksandr (Caterva)	09.05.2016
export20160502-191113.csv	Andre Engels	02.05.2016
Wrong Subject	aleks	02.05.2016
Wrong Subject	aleks	02.05.2016
export-20160502.xlsb	Andre Engels	02.05.2016
(Kein Betreff)	Romanov, Aleksandr (Caterva)	02.05.2016

Figure 12. Unread email example inside Caterva inbox

The picture above (Figure 12) demonstrates the highlighted message, indicating that it is unread. This is how the software system determines, whether the message should be processed.

```
foreach ($unseen as $msgNumber) {

    $message = imap_fetch_overview($inbox,$msgNumber,0)[0];
    //TODO recognize the appropriate subject [ CHECKLIST:XXXXX ]
    $subject = $message->subject;
    $from = $message->from;

    echo "Email subject: ".$subject."\n";
```

Code 2. Main email scanning loop

From (Code 2) you can see the part of email scanner implementation. Once the system recognized that there is an unread email (or multiple unread emails), it attempts to go through every message, which is marked as unread, and do some logic with it. The message object is constructed by taking the message number out of every unseen email and call **imap_fetch_overview** function, which is a part of IMAP library. As a result, a programmer can refer to the parts of an email like the subject and from fields.

All other logic happens inside this main loop, going through unread emails, calling the functions, defined in **attachments.php** and **csvparser.php.**

4.2 Extracting appropriate attachments

At this point, there is a built connection to Caterva email server and the working loop which goes through every single unread email. However, looping over unread emails is not sufficient. It is required to validate the content of every email and determine the following points:

- If the email contains attachment or not
- Whether the attachment is CSV or not
- Whether the email contains a checklist

The feature which simplified the solution to a problem, was a well-specified email subject, provided by ARANES. If the checklist is included into the attachment of the email, the email must contain a subject: [CHECKLIST:XXXXX], where XXXXX is a five digit number, representing the id of the lead in a CSV.

Now we can get back to the main loop and see how the solution was implemented.

```
if(strpos($subject,"[") !== false)
    {
        //THIS condition excludes the case when id is not present at all
        preg_match_all("/\[CHECKLIST:\d{5}\]/is", $subject,
$checklist_match);


        //TODO checking if CHECKLIST has arrived
        if(count($checklist_match[0]) != 0)
        {
            $zip_archive =
handleAttachment($subject,$message,$inbox,$zip,$from);
            if($zip_archive != NULL)
            {

//syncCheckList($zip_archive["id"],$from,$subject,$zip_archive["data"][2]);

attachFileToChecklist($zip_archive["id"],$zip_archive["data"][2], $from,
$subject);
            }
        }

        else
        {
            echo "Formatting error!"."\n";
        }
    }
```

Code 3. Checklist attachment handler

The picture above (Code 3) demonstrates the mechanism, in which the system recognizes the checklist. This part of the code is placed inside the main email scanning loop (Code 2).

- First, the system checks for the presence of the square bracket, in order to determine if the message should be treated as a checklist.
- If the square bracket was not detected the code should proceed further. On the other hand, if the square bracket was detected, the code uses a regular expression to determine if the subject matches the ARANES defined subject ([CHECKLIST:XXXXX]). It is done by calling built-in PHP function **preg_match_all**. The function will return 0, stored in a **$checklist_match** variable if the subject was wrongly formatted, otherwise, the execution continues.
- If the number of matches was not zero the system tries to extract the attachment, by calling custom function **handleAttachment** (Code 4).

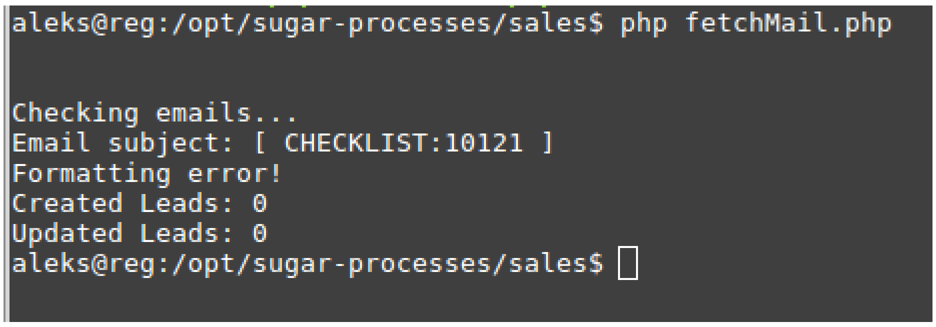
```
//TODO a function dealing with checklists and planung attachments
function handleAttachment($subject,$message,$inbox,$type,$from){

    //Checking for the sales id from the email subject
    preg_match_all('!\d+!', $subject, $numbers);

    if(count($numbers[0]) != 0)
    {
        //Extracting the sales id from the subject
        $id = $numbers[0][0];
        $uid = imap_uid($inbox, $message->msgno);
        //Returns a zip archive (Checklist)
        $archive = mailAttachments($inbox,$uid,$message->msgno,$type);

        //Handle the case when attachment is missing
        if(count($archive) == 0)
        {
            echo "No attachemnt was sent with the ".$subject."\n";
            notifyNoAttachment($from,$subject);
            return NULL;
        }

        else
        {
            //echo var_dump($zip_archive);
            return  array("data"=>$archive,"id"=>$id);
        }

    }
}
```

Code 4. Attachment handling function

The purpose of **handleAttachment** function was only for simplifying the interaction with the incoming checklists. In general, the function relied on **mailAttachments** function, which was defined in **attachments.php** file, provided by Caterva engineers. Since the **mailAttachment** function was not implemented by me and its usage was documented, it is enough to explain what the function performs. The **mailAttachment** function returned the array with all the attachments found in the email, where each element of the array was a path to the file.

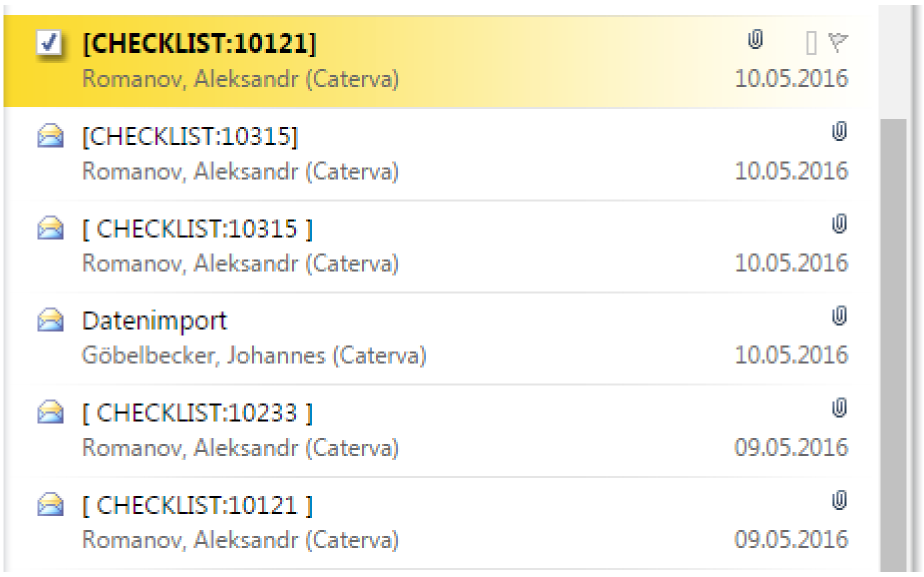
The result of the **handleAttachment** function is an array, containing the array with the paths to the attachments and an id, which was specified in the subject of the email (Figure 14). Eventually, in the main loop, when **handleAttachment** returns an array, we can check is the attachments actually exist and then execute some logic.

Figure 13 demonstrates the functionality of the attachment extraction functions, indicating that the formatting was wrong since two spaces were added to the beginning and ending of square brackets.



```
aleks@reg:/opt/sugar-processes/sales$ php fetchMail.php


Checking emails...
Email subject: [ CHECKLIST:10121 ]
Formatting error!
Created Leads: 0
Updated Leads: 0
aleks@reg:/opt/sugar-processes/sales$
```

Figure 13. Formatting error log



[CHECKLIST:10121] Romanov, Aleksandr (Caterva) 10.05.2016
[CHECKLIST:10315] Romanov, Aleksandr (Caterva) 10.05.2016
[CHECKLIST:10315] Romanov, Aleksandr (Caterva) 10.05.2016
Datenimport Göbelbecker, Johannes (Caterva) 10.05.2016
[CHECKLIST:10233] Romanov, Aleksandr (Caterva) 09.05.2016
[CHECKLIST:10121] Romanov, Aleksandr (Caterva) 09.05.2016

Figure 14. Checklist email example

Another part of the main loop is the CSV attachments handler. If the attachment is not a checklist, then another part of the logic gets executed inside the main loop.

```
$uid = imap_uid($inbox, $message->msgno);
        //Returns a csv attachment array
        $csv_path = mailAttachments($inbox,$uid,$message->msgno,$csv);

        if(count($csv_path) == 0){
            //echo "This is a broken path: ".var_dump($csv_path)."\n";
            echo "ERROR: The email does not contain attachment"."\n";

        else{
            $data = parseCSV($csv_path);
```

Code 5. Extracting CSV attachment

As you can see from (Code 5) **$csv_path** variable contains the result of the same function, **mailAttachment**, which was defined in the **attachments.php** file. After checking if the attachment is actually present, we can start parsing data.

4.3 Creating CSV parser and data structure

The CSV parser was one of the most important parts of the system. It has to be able to read the data from the CSV and construct the appropriate structure, ready for SugarCRM. There are three important components:

- Identify if the attachment is CSV and raise the error if not
- Convert the CSV data into the programmatically manageable data structure
- Create a list of customer objects, which can be easily put into the SugarCRM

The function is described in (Code 6). It receives the array of all found attachments inside the email and attempts to determine if the attachments have CSV formats. Eventually, if the attachment is indeed a CSV file, then **readCSV** function gets executed, reading the data from the CSV file and populating an array with lead items, representing CSV information.

```php
/**
 * The function takes the array with csv paths,
 * received from mailAttachment function and performs the parsing
 * by calling readCSV function. It also handles the error cases, when the
 * attahcment is not a csv. It returns the result of the readCSV function.
 * @param array $csv_paths_array - A collection of available attachments
 * @return NULL or array $data - Array respresenting the lead items with
necessary fields
 */
function parseCSV($csv_paths_array)
    {
    // Loop over all csv attachments in the array
    foreach($csv_paths_array as $att)
        {
        if (strpos($att, ".csv") === false)
            {
            continue;
            }
        echo "(" . date("l jS \of F Y h:i:s A") . "):" . " Attachment
received: " . $att . "\n";
        // The condition checking if the attachment is a CSV
        echo $att . "\n";
        if (strpos($att, ".csv") !== false)
            {
            $path_to_csv = $att;
            $data = readCSV($path_to_csv);

            if ($data != NULL)
                {
                echo "Parser started..." . "\n";
                return $data;
                }
              else echo "FAILED!" . "\n";
            }
          else
            {
            echo "Parse Failed!" . "\n";
            echo "The attachment received: " . $att . "\n";
            }
        }
    }
```

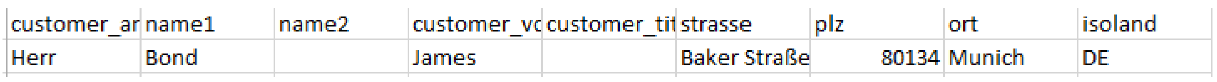
Code 6. CSV parser function

```
/**
 * The function takes the path to the CSV
 * and populates the $data stack with array elements,
 * representing a lead row inside the csv. The $data stack
 * is passed to prepareData function and the result of it is returned.
 * (Important notice: every element contains an array with ALL the
information
 * about the lead. Part of it must be filtered by prepareData function)
 * @param string $path_to_csv - A string representing a path to a file
 * @return NULL or array $collection - Array respresenting the lead items
with necessary fields
 */
function readCSV($path_to_csv)
    {
    $i = 0;
    $path_to_csv = fopen($path_to_csv, 'r');
    if ($path_to_csv !== FALSE)
        {
        while (!feof($path_to_csv))
            {
            $i++;
            $data[] = fgetcsv($path_to_csv, 0, ';', '"', '"');
            }
        fclose($path_to_csv);
        $collection = prepareData($data);
        $i = $i - 2;
        echo "Total leads in csv is: " . $i . "\n";
        return $collection;
        }
      else
        {
        echo "The path does not exist" . "\n";
        }
    }
```

Code 7. CSV reader

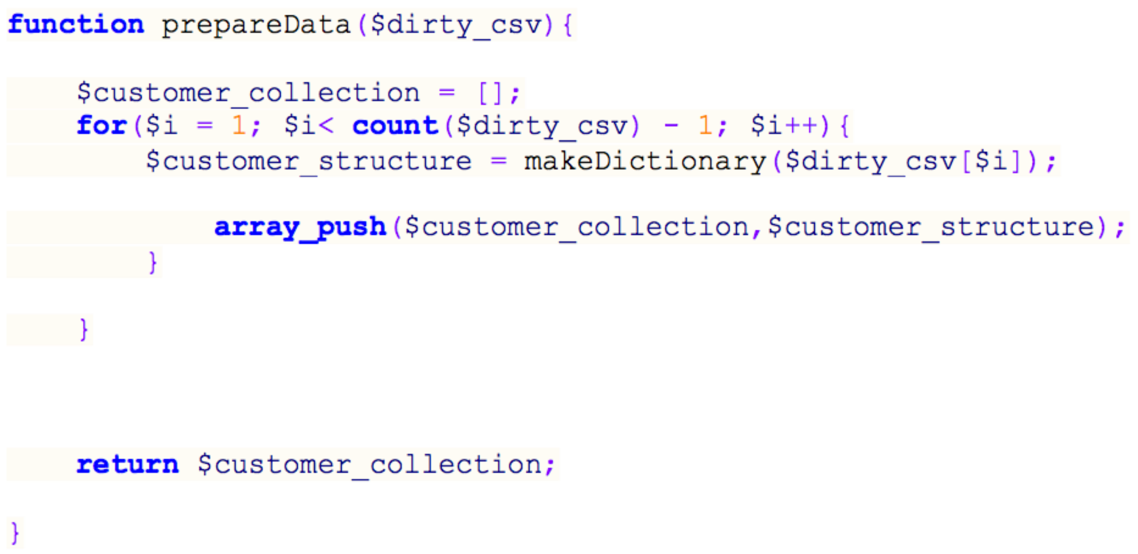
A picture above (Code 7) shows how standard file system functions were used, like **feof**, **fopen** and **fgetcsv**, in order to read file's content line by line, turning the CSV line into an array and then putting it on the data stack.

However, the CSV contains a lot of fields, which are not necessary for SugarCRM, for example, **name2** and **customer_title** (Figure 15). The only data Caterva needs was personal details and the id, given by ARANES system. For this reason, it was required to create the data structure which only contains needed information. In order to accomplish this task prepareData function was designed.



customer_ar	name1	name2	customer_vc	customer_tit	strasse	plz	ort	isoland
Herr	Bond		James		Baker Straße	80134	Munich	DE

Figure 15. CSV example with redundant fields



```
function prepareData($dirty_csv){

    $customer_collection = [];
    for($i = 1; $i< count($dirty_csv) - 1; $i++){
        $customer_structure = makeDictionary($dirty_csv[$i]);

            array_push($customer_collection,$customer_structure);
        }

    }



    return $customer_collection;

}
```

Code 8. Prepare data function

The function showed above (Code 8) demonstrates the simple loop through the data stack which was passed to the **prepareData** function (Code 7). In the scope of **prepareData,** I called a variable **$dirty_csv**, which contains the 2D array, in which every element contains all the information related to a lead. In order to make the data structure appropriate for SugarCRM, I created an empty array **$customer_collection**, which is supposed to contain the lead elements with only required fields. To get the required values, I built a **makeDictionary** function, which was taking only appropriate data and creating a dictionary, representing a lead.

Code 9 demonstrates the core design decision, making the reference to a lead much easier and more expressive. Every element passed into the **makeDictionary** function is turned into a dictionary, where every key is a field needed for SugarCRM and the value is a piece of data, which was taken by a CSV parser from a CSV file. The indexes you can see from (Code 9) are taken from a CSV file. By knowing at which position an entry was, I could refer to it in the code. Finally, **makeDictionary** returned a dictionary structure which was pushed into a **$customer_collection** variable, by calling the **array_push** function.

Eventually, we have reached the point where the information from the CSV is stored in a custom data structure, known as **$customer_collection**, in which every element is a dictionary, containing necessary lead details for SugarCRM system (Code 9). The next logical step is to understand how to use this data structure and populate the CRM with the data, using RESTful APIs, provided by SugarCRM, what will be discussed in the next chapter.

```
/**
 * The function builds the custom dictionary, representing the lead
 * element in the csv, containing only necessary fields.
 * @param array $c - An array with lead fields
 * @return array $dictionary - Dictionary with filtered lead fields
 */
function makeDictionary($c)
    {
    $dictionary = array(
        "customer_id" => utf8_encode($c[27]) ,
        "first_name" => str_replace('"', "", $c[3]) ,
        "last_name" => str_replace('"', "", $c[1]) ,
        "street" => $c[5],
        "place" => $c[7],
        "plz" => utf8_encode($c[6]) ,
        "phone" => utf8_encode($c[9]) ,
        "mobile" => utf8_encode($c[10]) ,
        "email" => utf8_encode($c[12]) ,
        "status" => utf8_encode(str_replace('"', "", $c[26])) ,
        "berater" => utf8_encode($c[28]) ,
        "contract_signed" => false,
    );
    return $dictionary;
    }
```

Code 9. Custom data structure creation function

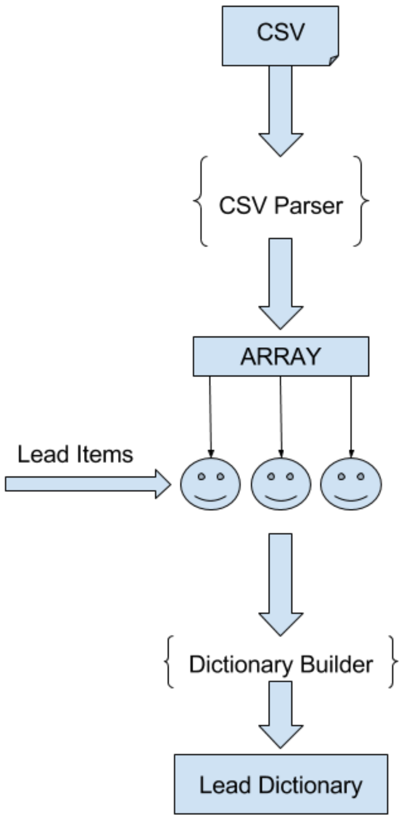


Figure 16. CSV data handler visualization

5 SUGAR CRM API

This chapter is designed to provide the reader with important implementation steps, required for safe and efficient interaction with SugarCRM. In particular, the chapter describes:

- How to configure OAuth protocol for the safe communication with SugarCRM
- How to build the API URLs to which the requests are sent
- How to make API calls to SugarCRM in order to create and update the entries
- How to link the files to appropriate leads

A picture below (Figure 17) demonstrates how the initial project structure has expanded, having two additional PHP files, designed for handling the interaction with SugarCRM. One of the files is **businessProcess.php**, which defined a set of functions, needed for creating new leads from CSV data, updating the entries and linking the documents. The **sugarAPI.php** file was designed to simplify the API calls, by dynamically constructing the URL address, to where the HTTP message was sent.

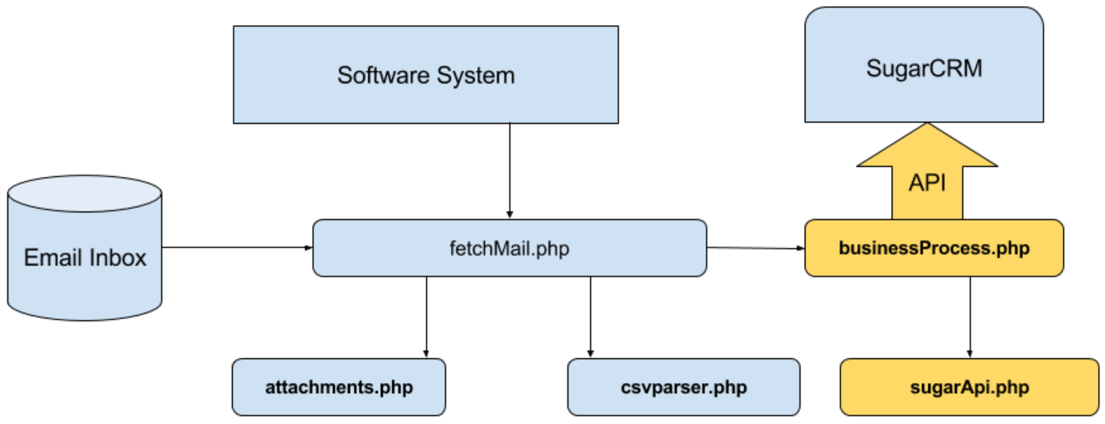
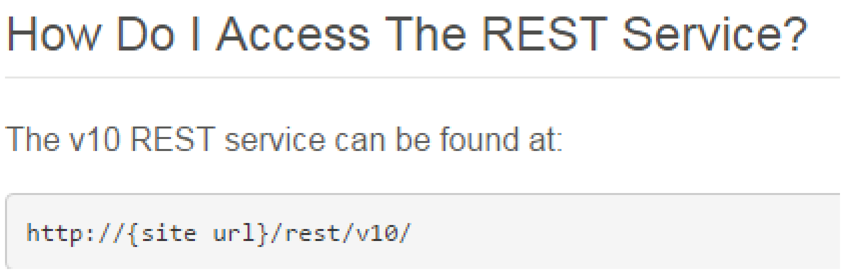


Figure 17. Structure of the software system with API handlers

5.1 Developer's page and OAuth configuration

The way how to use RESTful APIs provided by Sugar is described on the developer's page.



How Do I Access The REST Service?

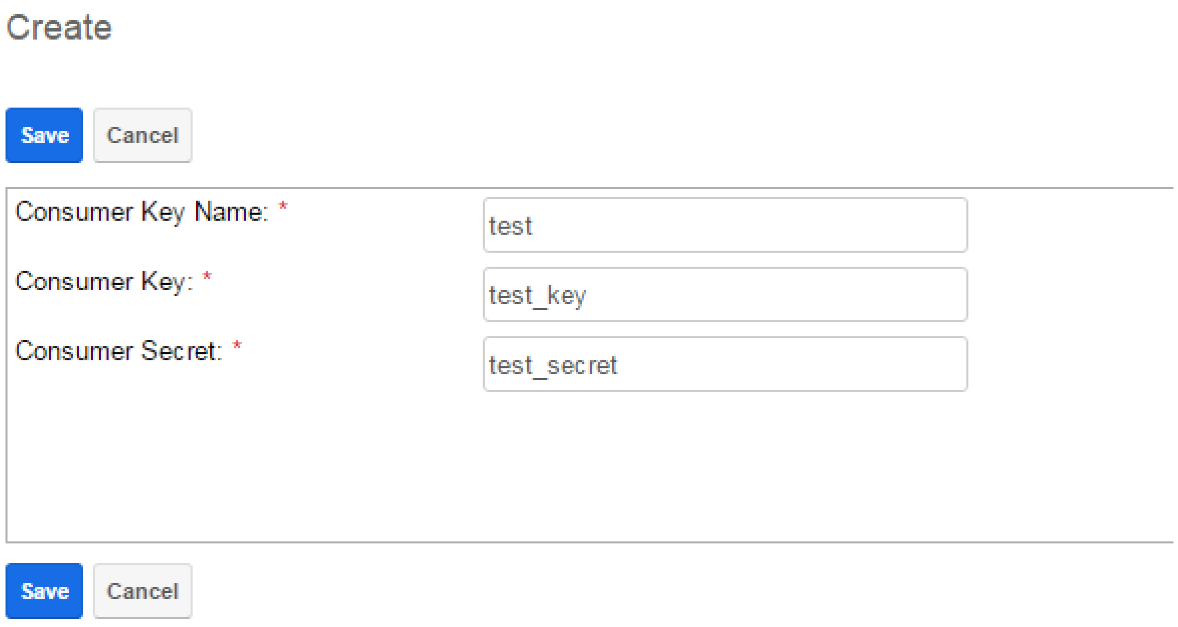
The v10 REST service can be found at:

```
http://{site url}/rest/v10/
```

Figure 18. How to access the REST service

Sugar CRM developer page provides multiple examples on how to make the API calls and how to configure the URL address, to where the requests are sent. The example above (Figure 18) shows which address should be used for accessing the data. In the example, {site url} part should be replaced with the real address of the server where SugarCRM is deployed. V10 indicates that version 10 of the API should be used.

In order to start sending the HTTP requests to the SugarCRM, it is required to configure OAuth protocol, so that the code which will be requesting the information from the system is registered within SugarCRM. In this case, SugarCRM acts as a resource provider and an authorization server, granting the access to the application, which is a software system, I am implementing. First of all, we will need to create the key and a secret, in order to identify the application.



Create

Consumer Key Name: *	test
Consumer Key: *	test_key
Consumer Secret: *	test_secret

Figure 19. OAuth configuration

From the figure 19, you can see how SugarCRM provides the way for creating the needed parameters, in order to authenticate the application. Having saved the configurations, I can refer to the parameters in the code and authenticate the software system to interact with SugaCRM (Code 10).

Code 10 also demonstrates the URL address configuration concept, shown on figure 18, the address is replaced by the IP of the testing server, where the development version of SugarCRM was deployed.

```php
<?php
$config = array(
"url"=>"http://192.168.9.11/sugarcrm/rest/v10",
"username"=>"admin",
"password"=>"password",
"client_id"=>"test_key",
"client_secret"=>"test_secret"
);
function getConfiguration(){

  global $config;
  return $config;

}
?>
```

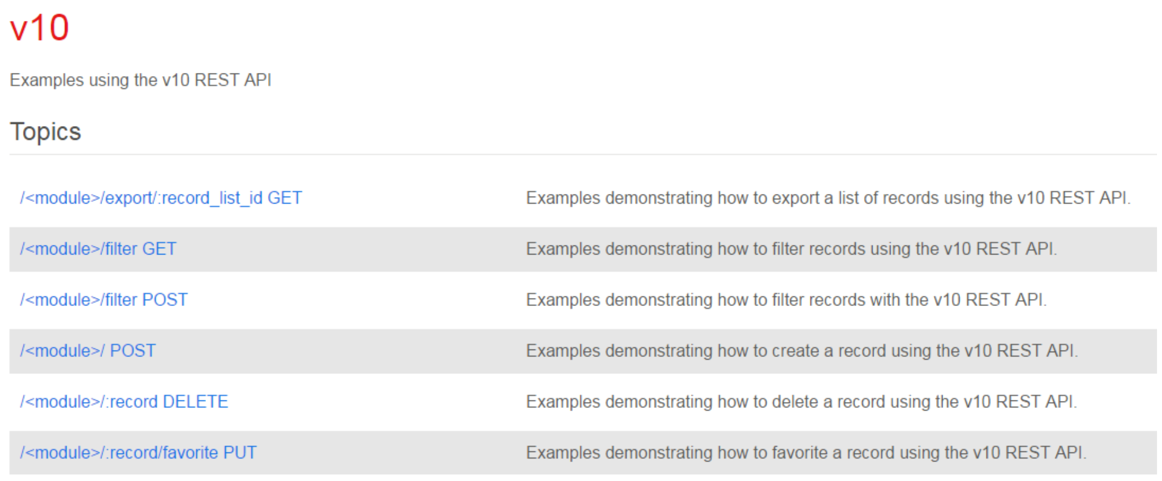
Code 10. Oauth configuration inside PHP code

5.2 Building API URLs

In order to send the HTTP request to SugarCRM and perform the appropriate operation, it is important to construct the URL address, indicating what is requested from service. For example:

- If the creation of a new entry is needed, the HTTP message POST must be sent to the service.
- If the update of the entry is needed, the HTTP message PUT should be sent.
- In order to request the entry's information the GET request should be sent
- If a programmatic login to SugarCRM is needed, the POST request should be sent

For every single case, the URL is totally different (figure 20).



v10

Examples using the v10 REST API

Topics

/<module>/export/:record_list_id GET	Examples demonstrating how to export a list of records using the v10 REST API.
/<module>/filter GET	Examples demonstrating how to filter records using the v10 REST API.
/<module>/filter POST	Examples demonstrating how to filter records with the v10 REST API.
/<module>/ POST	Examples demonstrating how to create a record using the v10 REST API.
/<module>/:record DELETE	Examples demonstrating how to delete a record using the v10 REST API.
/<module>/:record/favorite PUT	Examples demonstrating how to favorite a record using the v10 REST API.

Figure 20. Different URLs for API calls

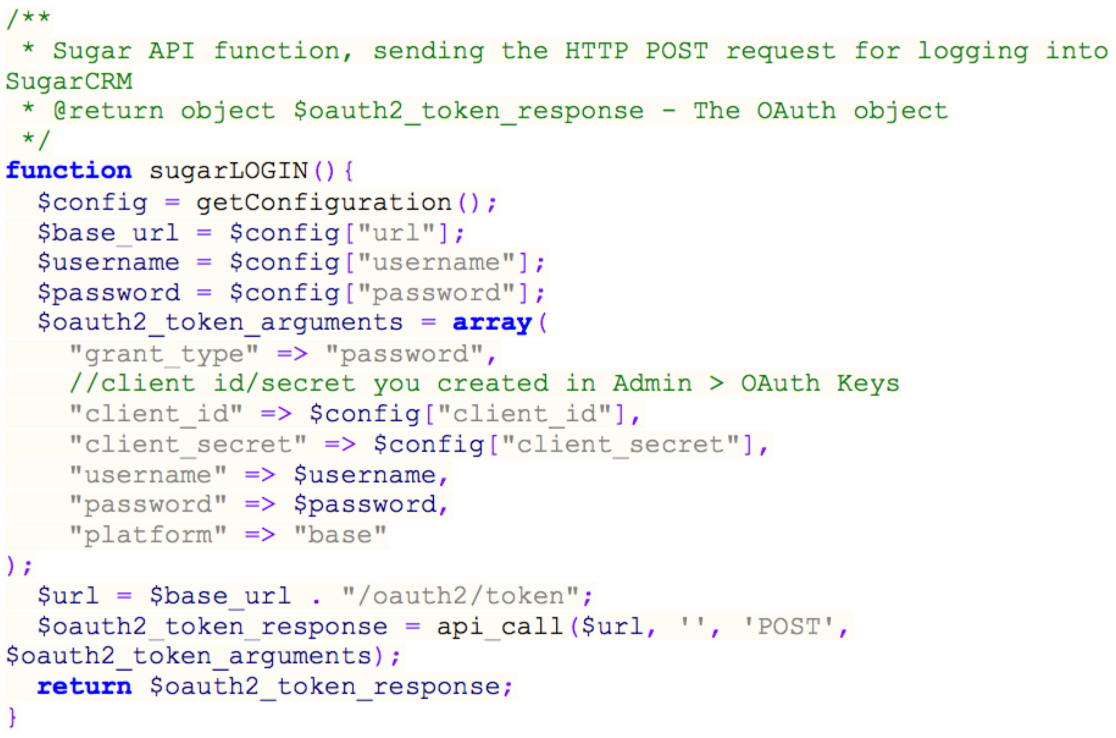
Figure 20 shows the examples from the developer's page, giving examples on how to construct URLs, describing what each of them performs and which HTTP method must be used. In addition, SugarCRM developer page provided the setup code for sending the API requests to the system, which you can see in Appendix 3.

The main important implementation decision was to write a layer of abstraction on top of the setup code, so that the future developers could easily call the high-level functions, without having to look at the setup code. The main functions inside **sugarApi.php** file (Code 11) are:

- **sugarPOST** is a function which is called, when the lead needs to be created

- **sugarUPDATE** is a function which is updating the lead entries
- **sugarGET** is a function, retrieving the specified entry
- **sugarLINK** is a function used for linking the documents (checklists) to the lead entries
- **sugarLOGIN** is a function designed for authenticating the application, returning the authorization token.

Now we can look at the implementation example inside the **sugarLOGIN** and **sugarUPDATE** functions, in order to understand how the URLs were built and how the setup code was used.



```
/**
 * Sugar API function, sending the HTTP POST request for logging into
SugarCRM
 * @return object $oauth2_token_response - The OAuth object
 */
function sugarLOGIN(){
  $config = getConfiguration();
  $base_url = $config["url"];
  $username = $config["username"];
  $password = $config["password"];
  $oauth2_token_arguments = array(
    "grant_type" => "password",
    //client id/secret you created in Admin > OAuth Keys
    "client_id" => $config["client_id"],
    "client_secret" => $config["client_secret"],
    "username" => $username,
    "password" => $password,
    "platform" => "base"
);
  $url = $base_url . "/oauth2/token";
  $oauth2_token_response = api_call($url, '', 'POST',
$oauth2_token_arguments);
  return $oauth2_token_response;
}
```

Code 11. SugarLOGIN function implementation

As you can see from the picture above (Code 11), first the code requests the configuration, containing all the parameters for using OAuth protocol, by calling the **getConfiguration** function (Code 10). After this step, the code has references to the main URL (the IP address of the deployed test version of SugarCRM), username, password, client id and client secret, which were set up previously. Having the parameters, we can construct the authentication array, held in the variable **$oauth2_token_arguments**. Next, I built the **$url** variable, appending to the main URL a path **/oauth2/token**, indicating that I requested the login from the service. Finally, I called the **api_call** function, the one which was provided by Sugar as the setup code, supplying **$url**, POST method and authentication array as arguments. The result of the call was the authentication token, which is returned by the function.

A **sugarLOGIN** function is now can be used to request the authentication token and the developer does not need to know about the URL and configurations. The URL has been constructed dynamically, by taking the main URL from the configuration and appending the specific string, describing what is requested. In the example described it was **/oauth2/token**, which is pretty much self-explanatory.

```
/**
 * Sugar API function, sending the HTTP PUT request for updating the item in
SugarCRM
 * @param object $oauth_token - The object ruturned by sugarLOGIN
 * @param array $parameters - The parameter array needed for making an
RESTful call
 * @return object $record_response
 */
function sugarUPDATE($oauth_token, $parameters){
   $config = getConfiguration();
   $url = $config["url"] .
$parameters["cstm_module"].$parameters["cstm_database_id"];
   echo "URL UPDATE: ".$url."\n";
   $record_response = api_call($url, $oauth_token, 'PUT', $parameters);
   return $record_response;
}
```

Code 12. Update function example

Code 12 demonstrates almost the same functionality, described in **sugarLOGIN** function. First, the code requests the OAuth protocol details, returned by **getConfiguration**. Next step is the URL building, which is different from the **sugarLOGIN** function. In this example, URL consists of the base URL, which is stored in the configuration and some custom parameters which are appended to the base URL. The first parameter is a custom module, telling SugarCRM, what kind of information is requested, for example, Lead, Document or Task. The second one is the database id of the entry, needed to perform the update operation. The actual parameters will be shown in the **businessProcess.php** file because it contains all the functions, which supply data to the parameters list. In the end, the **api_call** is executed, returning the response, indicating if the update was successful or not.

5.3 Running business processes

At this point, the system is capable of reading the CSV file, constructing the data structure with lead entries and abstracting the API calls to SugarCRM system. The following logical step would be to bring the functionality together and implement the logic, which will actually create a lead entry, update it, when needed, or link the checklist documents to the lead. The described functionality is implemented inside the **businessProcess.php** file, which:

- Contains the functions, responsible for creation, update and the link operations
- Uses the API abstraction functions, defined in **sugarApi.php**

Essentially every single function inside **businessProcess.php** follows one principle:

- Retrieve the access token, by calling the **sugarLOGIN** function.
- Construct the argument list, which will be sent to the SugarCRM
- Make the API call, using abstraction API functions
- Return the response from the API call

Each function in the file is responsible for a unique operation.

5.3.1 Creating a lead

```
include (dirname(__DIR__) . '/sugar-conn/sugarApi.php');
/**
 * A function performs a lead creation operation
 * by taking a $data item as an argument
 * @param array $data - A dictionary with lead fields
 */
function createNewLead($data)
    {
    $oauth = sugarLOGIN();
    $record_arguments = array(
        "cstm_module" => "/Leads/",
        "cstm_relationship" => "",
        "cstm_database_id" => "",
        "first_name" => utf8_encode($data["first_name"]) ,
        "last_name" => utf8_encode($data["last_name"]) ,
        "phone_day_c" => str_replace('"', "", $data["phone"]) ,
        "email1" => $data["email"],
        "id_lead_c" => $data["customer_id"],
        "custom_zip_c" => $data["plz"],
        "phone_evening_c" => str_replace('"', "", $data["mobile"]) ,
        "custom_city_c" => utf8_encode(str_replace('"', "", $data["place"]))
,
        "custom_address_c" => utf8_encode(str_replace('"', "",
$data["street"])) ,
        "berater_c" => utf8_encode($data["berater"]) ,
    );
    $record_response = sugarPOST($oauth->access_token, $record_arguments);
    echo "\n";
    echo "-------------------------" . "\n";
    echo "A new Lead " . utf8_encode($data["first_name"]) . " " .
utf8_encode($data["last_name"]) . " was created --> sales given id: " .
$data["customer_id"] . "\n";
    echo "\n";
    }
```

Code 13. New Lead creation function

Code 13 shows the function, which creates a new lead entry inside SugarCRM, by using the API abstraction functions, defined in **sugarApi.php**. On top of the code, you can see an include statement, which demonstrates how the Sugar API functions were included. The **createNewLead** function accepts a **$data** parameter, representing the piece of data from a dictionary I built previously. To demonstrate where the data comes from, we need to take a look back to the main email loop (Code 14).

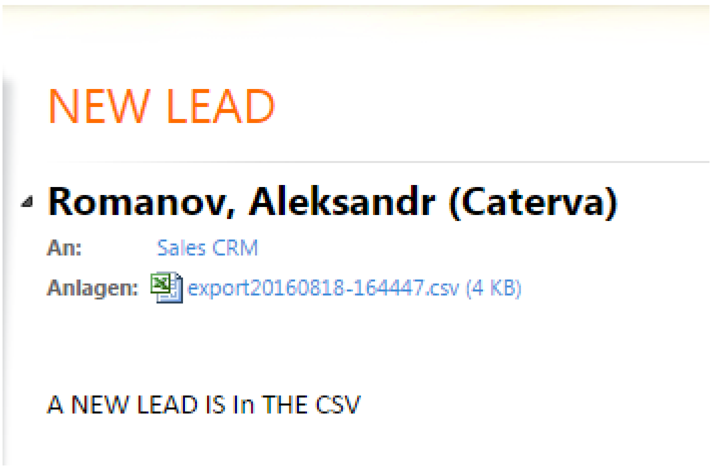
```
            if($data!=NULL){
                foreach ($data as $customer){
                    //syncDB($customer);
                    $lead_db_id = getLeadRecord($customer["customer_id"]);
                    if($lead_db_id == NULL){
                        createNewLead($customer);
```

Code 14. A loop through dictionary elements

Code 14 provides a code, demonstrating where the **createNewLead** function is called. This code lives inside the main email loop, meaning that after the dictionary with CSV data was constructed, the code loops through every single element in the dictionary and calls the API functions, providing the data, stored in the element to a parameter list, **$record_arguments** array (Code 13).

In addition, code 13 demonstrates the parameters keys like **cstm_module** or **cstm_database_id**, which are used to construct an URL inside API functions in **sugarApi.php** (Code 12). Since for a new lead, there is no existing database id, the field is left empty. However, the **cstm_module** field is set to Leads (Code 13).

Eventually, we can run the code and perform the creation of the lead inside the SugarCRM. For running the test we will need to send the CSV to Caterva inbox.


NEW LEAD

Romanov, Aleksandr (Caterva)

An: Sales CRM

Anlagen: export20160818-164447.csv (4 KB)

A NEW LEAD IS In THE CSV

Figure 21. Test CSV import

Figure 21 shows the test CSV import I sent to Caterva inbox. Now I can run fetchMail.php to see the creation process. Code 13 shows the log output after running the code.

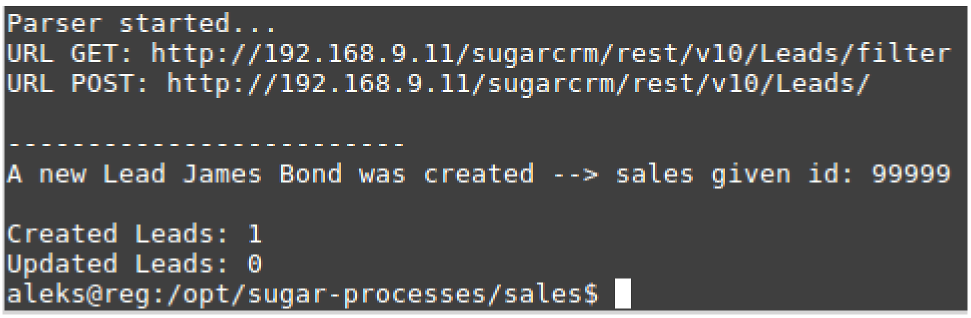
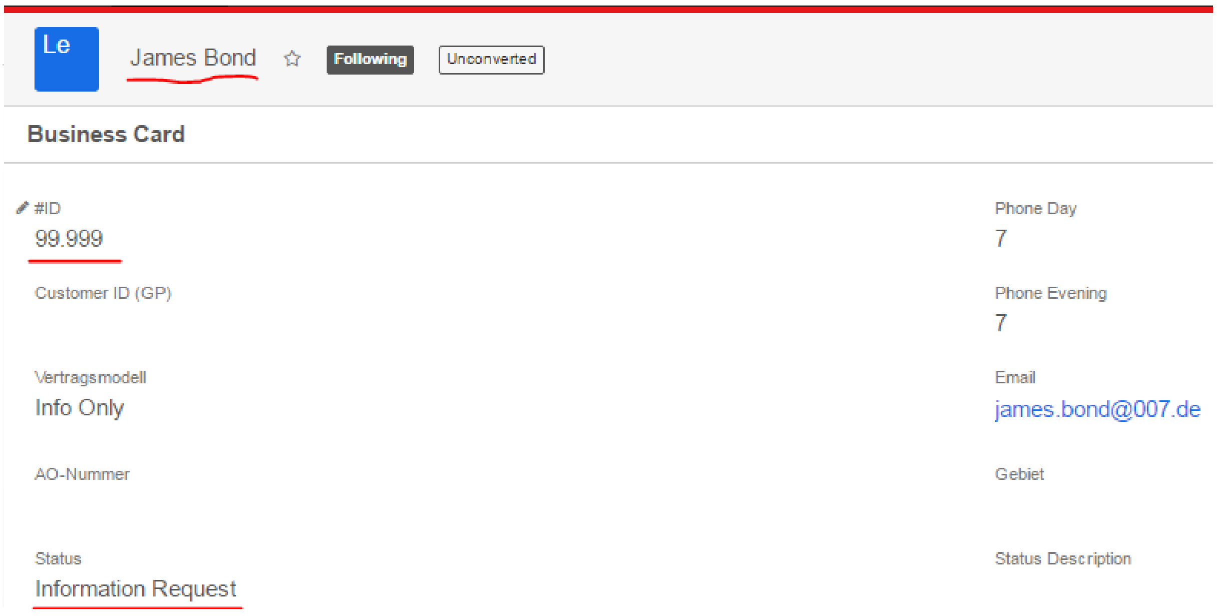

```
Parser started...
URL GET: http://192.168.9.11/sugarcrm/rest/v10/Leads/filter
URL POST: http://192.168.9.11/sugarcrm/rest/v10/Leads/

-------------------------
A new Lead James Bond was created --> sales given id: 99999

Created Leads: 1
Updated Leads: 0
aleks@reg:/opt/sugar-processes/sales$
```

Figure 22. Newly created Lead



Le James Bond Following Unconverted

Business Card

#ID
99.999

Customer ID (GP)

Vertragsmodell
Info Only

AO-Nummer

Status
Information Request

Phone Day
7

Phone Evening
7

Email
james.bond@007.de

Gebiet

Status Description

Figure 23. Created Lead inside SugarCRM

Figure 23 demonstrates how the code execution created the entry inside the SugarCRM system, assigning the status to Information Request. The status will be important in the following chapters, describing workflow automation.

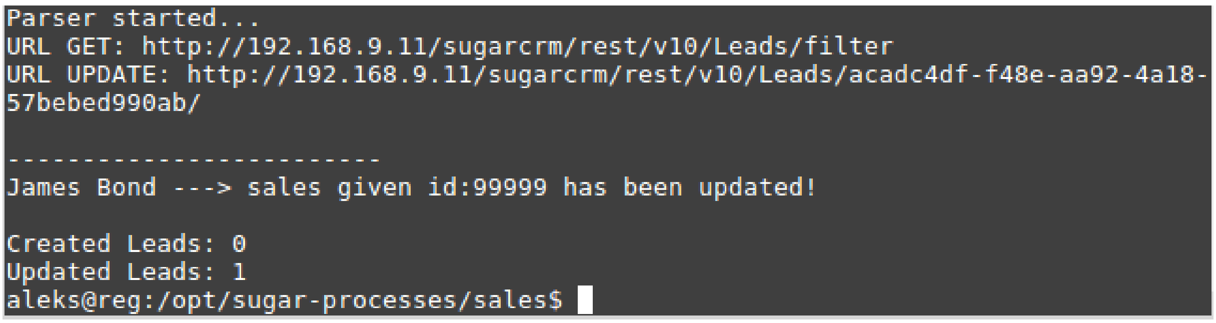
5.3.2 Updating the lead

In order to update the lead entry, let us first take a look at the code (Code 15)

```
/**
 * A function performs an update of the existing lead
 * by taking the $data parameter, representing the lead,
 * and a database id
 * @param array $data - A dictionary with lead fields
 * @param $id - Database id of the lead
 */
function updateNewLead($data, $id)
    {
    $oauth = sugarLOGIN();
    $record_arguments = array(
        "cstm_module" => "/Leads/",
        "cstm_database_id" => $id . "/",
        "cstm_relationship" => "",
        "first_name" => utf8_encode($data["first_name"]) ,
        "last_name" => utf8_encode($data["last_name"]) ,
        "phone_day_c" => str_replace('"', "", $data["phone"]) ,
        "email1" => $data["email"],
        "id_lead_c" => $data["customer_id"],
        "custom_zip_c" => $data["plz"],
        "phone_evening_c" => str_replace('"', "", $data["mobile"]) ,
        "custom_city_c" => utf8_encode(str_replace('"', "", $data["place"]))
,
        "custom_address_c" => utf8_encode(str_replace('"', "",
$data["street"])) ,
        "berater_c" => utf8_encode($data["berater"]) ,
    );
    $record_response = sugarUPDATE($oauth->access_token, $record_arguments);
    echo "\n";
    echo "--------------------------" . "\n";
    echo $data["first_name"] . " " . $data["last_name"] . " ---> sales given
id:" . $data["customer_id"] . " has been updated!" . "\n";
    echo "\n";
    }
```

Code 15. Update Lead function

As you can see from the code 15, the function takes the data item from the dictionary and the database id of the existing lead. As usual, the first step is to get the authentication token, by calling **sugarLOGIN** function, then constructing the argument list, making API call and returning the response. This function differs from lead creation function because it accepts a database id, needed for URL construction, indicating that the lead is not a new one and it must be updated. Below is the example of the output, when the lead gets updated after the code gets executed.



```
Parser started...
URL GET: http://192.168.9.11/sugarcrm/rest/v10/Leads/filter
URL UPDATE: http://192.168.9.11/sugarcrm/rest/v10/Leads/acadc4df-f48e-aa92-4a18-
57bebed990ab/

-------------------------
James Bond ---> sales given id:99999 has been updated!

Created Leads: 0
Updated Leads: 1
aleks@reg:/opt/sugar-processes/sales$ █
```

Figure 24. Updated Lead

Figure 24 shows the output of the update operation. As you can see the UPDATE URL contains the database id, making sure that the lead has been updated.

5.3.3 Get the lead record

As part of the requirement, ARANES was responsible for providing a unique id number inside the CSV file, identifying a lead. You can see the sales given id number in figure 24 and figure 23. The id was used to interact with SugarCRM, in order to request the lead entries based on it. The following code demonstrates the functionality (Code 16). The function takes the ARANES given id (sales given id) as a parameter and performs the API call, the result of which is either the lead record, indicating that the entry might be updated, or 0, indicating that the lead entry does not exist. If the entry exists, the function returns the database id, supplied inside an **updateLead** function, or **Null**, if the API call returned 0. In the case of latter, the **createNewLead** function is executed (Code 17).

```
/**
 * A function returns a database id of the lead
 * based on the ARANES given id, if the entry exists.
 * Otherwise NULL is returned.
 * @param int $sales_id - An ARANES given id (e.g. Sales id 12345)
 * @return NULL or $lead_id - Database id of the lead
 */
function getLeadRecord($sales_id)
    {
    $oauth = sugarLOGIN();
    $filter_arguments = array(
        "cstm_module" => "/Leads/filter",
        "cstm_relationship" => "",
        "cstm_database_id" => "",
        "filter" => array(
            array(
                "id_lead_c" => $sales_id
            )
        ) ,
        "max_num" => 2,
        "offset" => 0,
        "fields" => "name,description",
        "order_by" => "name:DESC",
        "favorites" => false,
        "my_items" => false,
    );
    $res = sugarGET($oauth->access_token, $filter_arguments);
    if (count($res->records) == 0)
        {
        return NULL;
        }
      else
        {
        $lead_id = $res->records[0]->id;
        return $lead_id;
        }
    }
```

Code 16. Get Lead entry function

```
if($data!=NULL){
                foreach ($data as $customer){
                    //syncDB($customer);
                    $lead_db_id = getLeadRecord($customer["customer_id"]);
                    if($lead_db_id == NULL){
                        createNewLead($customer);
                        $counter_create++;
                    }

                    else{
                        updateNewLead($customer,$lead_db_id);
                        $counter_update++;
                    }
```

Code 17. Update Lead or create Lead logic

Whenever **createNewLead** or **updateNewLead** are called, there is always a preceding call to the **getLeadRecord** function, the result of which determines whether the entry should be updated or created from scratch (Code 17).

5.3.4 Linking checklist to a lead entry

We already know how the main email loop detects the checklist presence inside the email. However, after detecting it there is now logic executed yet. For document linking the software system had to perform the following actions:

- Create a document entry inside SugarCRM
- Link the document entry to the appropriate lead entry
- Attach a real zip file from the email to a document entry
- Create two linked to a lead tasks, which are assigned to a responsible employee, needed for starting the customer acquisition process

The implementation also relies on the written Sugar API functions. The starting point was the implementation of the **createDocument** function.

```
/**
 * A function creates a document entry inside SugarCRM.
 * It returns a database id of a newly created entry.
 * @param int $salesID - An ARANES given id (e.g. Sales id 12345)
 * @return $doc_id - Database id of the document
 */
function createDocument($salesID)
    {
    $oauth = sugarLOGIN();
    $record_arguments = array(
        "cstm_module" => "/Documents/",
        "cstm_relationship" => "",
        "cstm_database_id" => "",
        "name" => "CHECKLIST:" . $salesID,
    );
    $res = sugarPOST($oauth->access_token, $record_arguments);
    $docID = $res->id;
    return $docID;
    }
```

Code 18. Create document function

The function relies on the same structure, as the functions, creating and updating lead entries. The result of the function is the document id, stored in the database after the document was created. Having the id of the document we can link it to the appropriate lead since we know the ARANES given id from the subject of the email, taken by **handleAttachment** function (Code 19).

```php
/**
 * A function performs the linking between a document and a lead
 * The function relies on getLeadRecord and createDocument functions,
 * which return lead and document database id's respectively
 * @param int $salesID - An ARANES given id (e.g. Sales id 12345)
 * @param string $checklist_email - Email of the checklist sender
 * @param string $checklist_subject - Subject of the email
 * @return NULL or array - A dictionary with the database id of the lead and
a database id of the document
 *
 */
function linkDocumentToLead($salesID, $checklist_email, $checklist_subject)
    {
    $oauth = sugarLOGIN();
    $leadID = getLeadRecord($salesID);
    if ($leadID != NULL)
        {
        $docID = createDocument($salesID);
        $record_arguments = array(
            "cstm_module" => "/Documents/",
            "cstm_relationship" => "link/leads_documents_1/" . $leadID,
            "cstm_database_id" => $docID . "/",
        );
        $res = sugarLINK($oauth->access_token, $record_arguments);
        return array(
            "document_id" => $docID,
            "lead_id" => $leadID
        );
        }
      else
        {
        notifyWrongLeadId($checklist_email, $checklist_subject);
        return NULL;
```

Code 19. Linking document to a Lead

The code above (Code 19) demonstrates the flow in which the document was linked to a lead. Inside the body of the **linkDocumentToLead** function, there are calls to the **getLeadRecord** function, which gets the lead database id, based on the ARANES given id, and to **createDocument** function, which returned the database id of the document. The **$record_arguments** variable contains the known keys, needed to construct the URL for the API call. In this example, the **cstm_relationship** key is used, which specifies the name of the relationship. Since we are linking a document to a lead, the relationship name is **leads_documents**. Having called the function, the returned result is a dictionary, containing the database id of the document and the lead, which are needed for attaching the real checklist file from the email to a linked document.

```
/**
 * A function attaches a real file attachment to a document entry,
 * linked to an appropriate lead.
 * @param int $salesID - An ARANES given id (e.g. Sales id 12345)
 * @param string $checklist_email - Email of the checklist sender
 * @param string $checklist_subject - Subject of the email
 * @param string $path- Path to the checklist attachment
 */
function attachFileToChecklist($salesID, $path, $checklist_email,
$checklist_subject)
    {
    echo "PATH: " . $path . "\n";
    $dictionary = linkDocumentToLead($salesID, $checklist_email,
$checklist_subject);
    if ($dictionary != NULL)
        {
        $oauth = sugarLOGIN();
        $record_arguments = array(
            "cstm_module" => "/Documents/",
            "cstm_relationship" => "file/filename/",
            "cstm_database_id" => $dictionary["document_id"] . "/",
            "format" => "sugar-html-json",
            "delete_if_fails" => true,
            "oauth_token" => $oauth->access_token,
            "filename" => "@" . $path
        );
        $res = sugarPOST($oauth->access_token, $record_arguments);
        createEntryTasks($dictionary["lead_id"]);
        notifyOnSuccessfulChecklist($checklist_email, $checklist_subject);

        }
      else
        {
        echo "(Sales Error) Message has been sent to: " . $checklist_email .
"\n";
        echo "(Sales Error) Wrong sales id supplied" . "\n";
```

Code 20. Attaching real file to a Document entry

Code 20 shows the example, how the checklist file is attached to a document entry. The function first makes a call to the **linkDocumentToLead** function, receiving the dictionary with database ids of the lead and a document and after constructs the argument list for an API call. In this case, the list contains the keys with the filename, which is assigned to the path of the attachment. In addition, you can see how the document database id is used for pointing to a newly created document. The lead database id is supplied to **createEntryTasks** function, which will be linked to a lead.

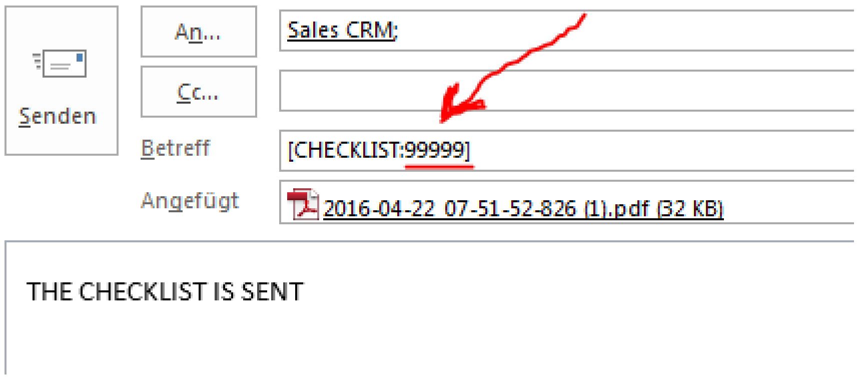
The **createEntryTasks** function (Code 21) performs the creation of two SugarCRM tasks, which are necessary for getting started the acquisition process. As you can see the function constructs two argument lists, in which the database id of the lead is added as a parent id parameter. This tells the API to link the tasks to the appropriate lead.

```
/**
 * A function creates 2 entry tasks after the
 * checklist has been attached, and links the tasks
 * to a lead.
 * @param $lead_db_id - Lead database id
 */
function createEntryTasks($lead_db_id)
    {
    $oauth = sugarLOGIN();
    $description = "The checklist has been sent and is available under
Documents subpanel in a Lead";
    $parameters_checklist_sent = array(
        "cstm_module" => "/Tasks/",
        "cstm_relationship" => "",
        "cstm_database_id" => "",
        "description" => $description,
        "status" => "Not Started",
        "priority" => "High",
        "name" => "II 05 Checkliste vollständig?",
        "parent_id" => $lead_db_id,
        "parent_type" => "Leads",
        // SIMON
        "assigned_user_id" => "90769b6d-5abf-8356-1ce0-56604d9fc7a7",
    );
    $task_checklist_sent_response = sugarPOST($oauth->access_token,
$parameters_checklist_sent);
    $parameters_checklist_complete = array(
        "cstm_module" => "/Tasks/",
        "cstm_relationship" => "",
        "cstm_database_id" => "",
        "description" => "",
        "status" => "Not Started",
        "priority" => "High",
        "name" => "II 06 Vertrag vorhanden?",
        "parent_id" => $lead_db_id,
        "parent_type" => "Leads",
        // SIMON
        "assigned_user_id" => "90769b6d-5abf-8356-1ce0-56604d9fc7a7",
    );
    $task_checklist_compelete_response = sugarPOST($oauth->access_token,
$parameters_checklist_complete);
    }
```

Code 21. Tasks creation function

Now I can demonstrate how the combination of the functions described is used in order to create an attached document inside SugarCRM, linked to a lead. The linking process starts from the call to the **attachFileToChecklist** function (Code 20) within the main email scanning loop.

First ARANES sends the email with the checklist, specifying the correctly formatted subject, with the right sales id, which they wrote to the CSV.



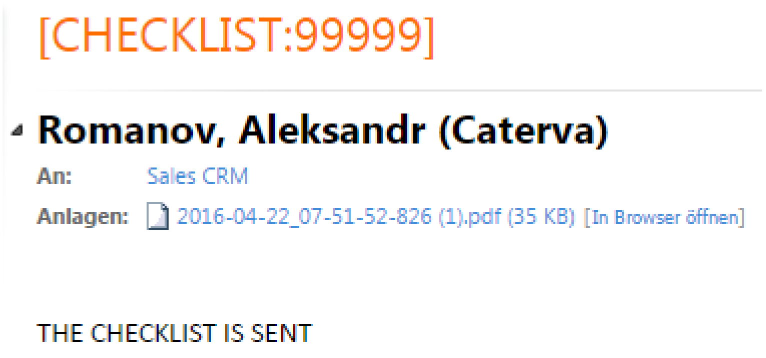
Sales CRM;

[CHECKLIST:99999]

2016-04-22 07-51-52-826 (1).pdf (32 KB)

THE CHECKLIST IS SENT

Figure 25. Preparing checklist email


[CHECKLIST:99999]

Romanov, Aleksandr (Caterva)

An: Sales CRM

Anlagen: 2016-04-22_07-51-52-826 (1).pdf (35 KB) [In Browser öffnen]

THE CHECKLIST IS SENT

Figure 26. Received checklist email

From the figure 25 and figure 26, you can see the example of how the email is sent to Caterva inbox with the correct subject formatting and the sales id 99999, pointing to the lead which had already been created.

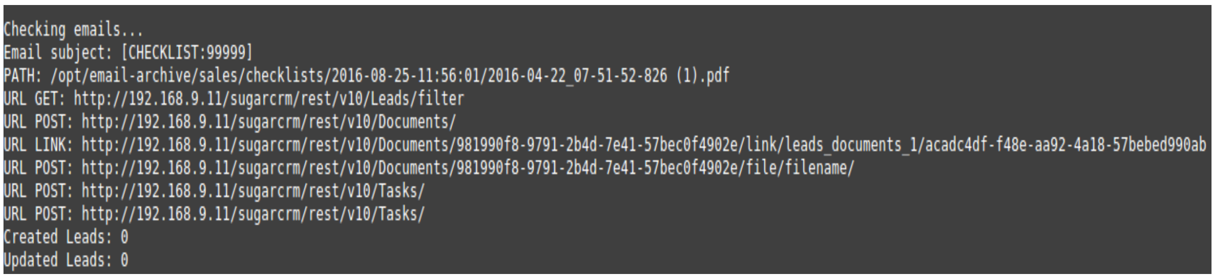
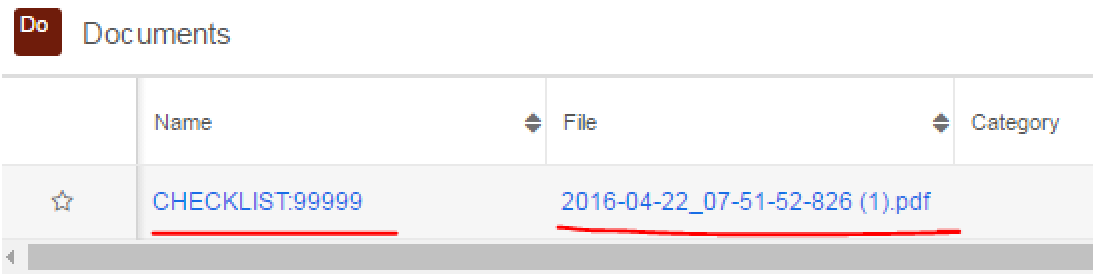

```
Checking emails...
Email subject: [CHECKLIST:99999]
PATH: /opt/email-archive/sales/checklists/2016-08-25-11:56:01/2016-04-22_07-51-52-826 (1).pdf
URL GET: http://192.168.9.11/sugarcrm/rest/v10/Leads/filter
URL POST: http://192.168.9.11/sugarcrm/rest/v10/Documents/
URL LINK: http://192.168.9.11/sugarcrm/rest/v10/Documents/981990f8-9791-2b4d-7e41-57bec0f4902e/link/leads_documents_1/acadc4df-f48e-aa92-4a18-57bebed990ab
URL POST: http://192.168.9.11/sugarcrm/rest/v10/Documents/981990f8-9791-2b4d-7e41-57bec0f4902e/file/filename/
URL POST: http://192.168.9.11/sugarcrm/rest/v10/Tasks/
URL POST: http://192.168.9.11/sugarcrm/rest/v10/Tasks/
Created Leads: 0
Updated Leads: 0
```

Figure 27. Checklist link log output

After executing **fetchMail.php**, the logic detects that the email was a formatted as a checklist email and attempts to perform the linking of the document to a lead. From the log output (figure 27) you can see the number of URLs printed, which demonstrates the sequence in which the API calls happened. The list below explains the flow in-depth.

- First, a GET request is sent, in order to retrieve the database id of the lead
- Second is the POST request, which creates a document item in SugarCRM
- Third is a LINK request, which linked the document with the lead
- Fourth is a POST request, which attaches a real attachment from the email to a document entry
- Fifth and sixth are POST request, which created two tasks, linked to a lead entry

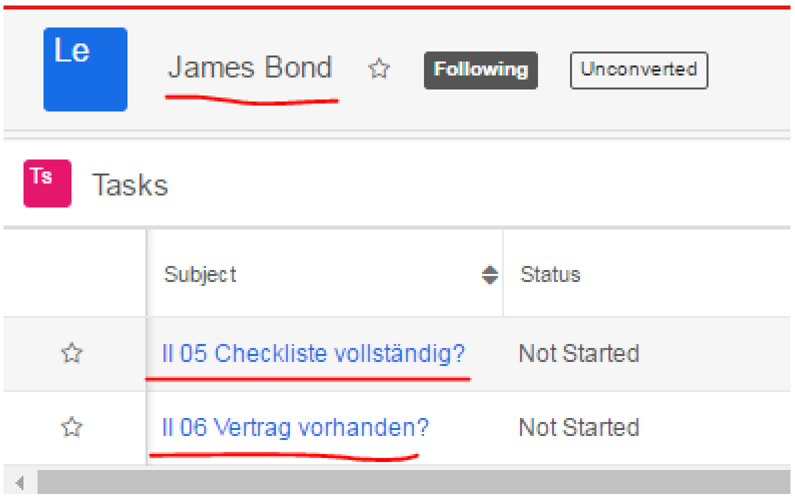
Now we can see the changes, which took place in SugarCRM.



	Name	File	Category
☆	CHECKLIST:99999	2016-04-22_07-51-52-826 (1).pdf	

Figure 28. Linked checklist to a Lead

Figure 28 demonstrates how the attachment from the email has become available inside the SugarCRM, linked to an appropriate lead with 99999 id.



	Subject	Status
☆	II 05 Checkliste vollständig?	Not Started
☆	II 06 Vertrag vorhanden?	Not Started

Figure 29. Linked tasks to a Lead

Figure 29 shows how tasks have been linked to a lead which indicates to an assigned person that he has some work to do.

6 TASK FLOW AUTOMATION

This chapter focuses on demonstrating how SugarCRM provides the logic, capable of detecting if an action was triggered within the system by the user. For example:

- if the lead status was changed, more tasks must be created (Appendix 1)
- if the task status is changed, then detect the status of the another task and execute some logic (Appendix 1)

Figure 29 demonstrates the tasks which were linked to a lead, indicating that a responsible employee has to process them. Now we can demonstrate how changing lead status from “Information Request” to “Qualification Process” will create two more tasks, notifying the assigned user, that the workflow was started.

Figure 30 demonstrates how the status is changed from SugarCRM user interface which triggers two task creations, linked to the lead.

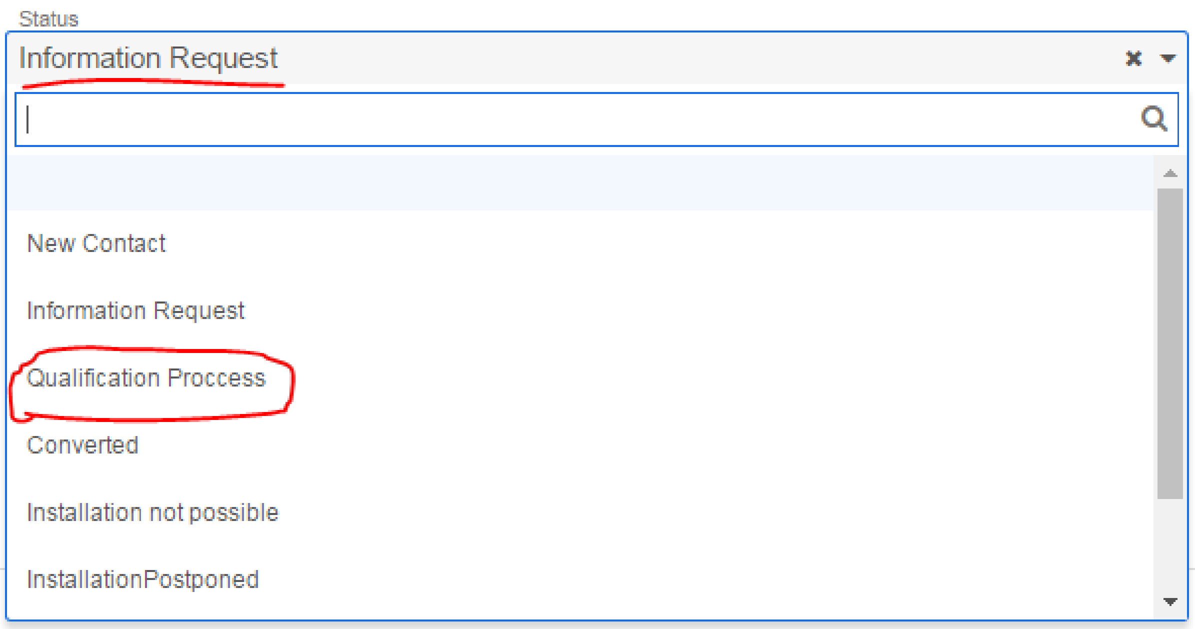
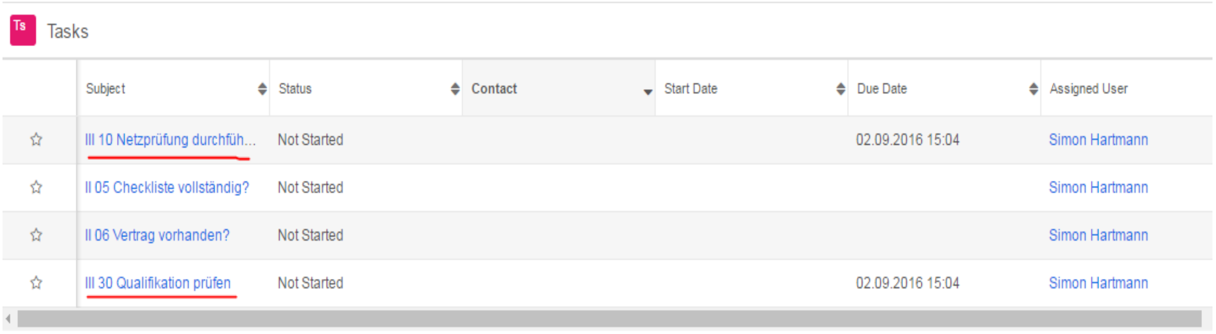


Figure 30. Lead status changed

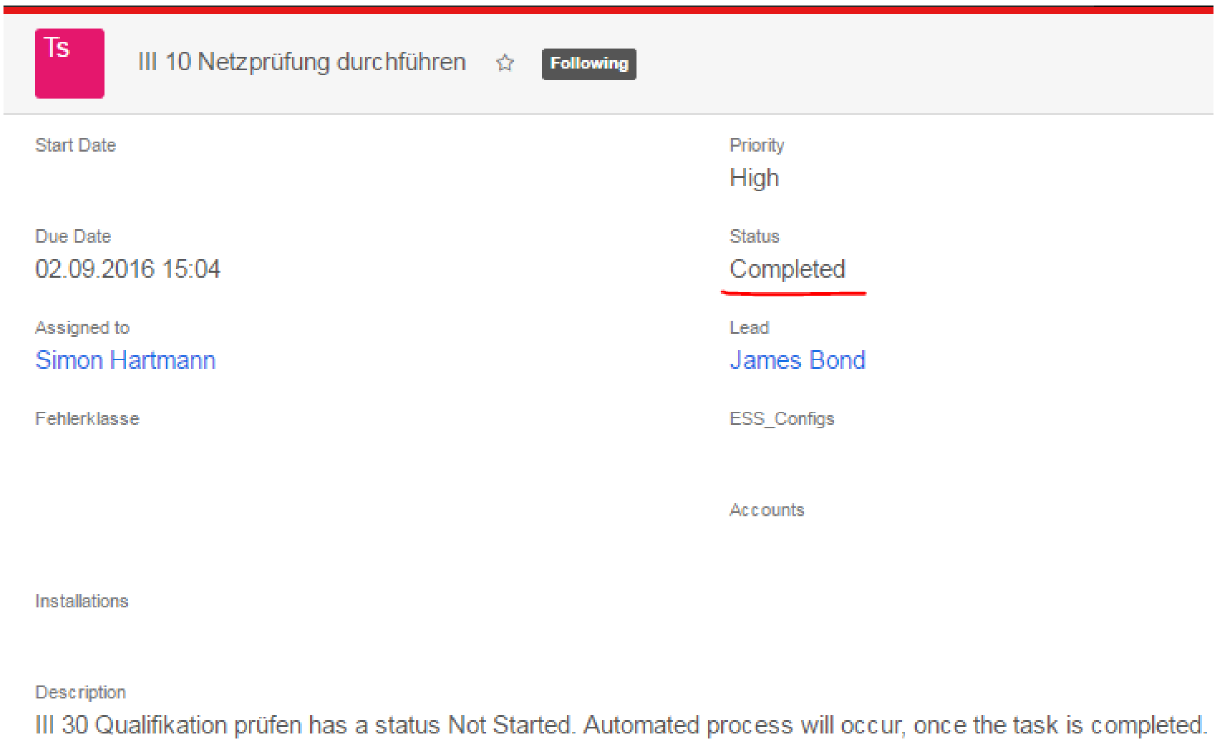


Tasks

	Subject	Status	Contact	Start Date	Due Date	Assigned User
☆	III 10 Netzprüfung durchfüh...	Not Started			02.09.2016 15:04	Simon Hartmann
☆	II 05 Checkliste vollständig?	Not Started				Simon Hartmann
☆	II 06 Vertrag vorhanden?	Not Started				Simon Hartmann
☆	III 30 Qualifikation prüfen	Not Started			02.09.2016 15:04	Simon Hartmann

Figure 31. Newly created Tasks after Lead status is changed

Another important functionality was the ability to change the status of the task and simultaneously detect whether another complementary task was processed. Based on the task statuses some logic was executed, which you can see in figure 32, demonstrating how marking the task as ”Completed” automatically posts the information about another task into the description field. Eventually, by having this description a responsible employee can immediately recognize what should be done next.



III 10 Netzprüfung durchführen ☆ Following

Start Date	Priority High
Due Date 02.09.2016 15:04	Status Completed
Assigned to Simon Hartmann	Lead James Bond
Fehlerklasse	ESS_Configs
	Accounts
Installations	

Description
III 30 Qualifikation prüfen has a status Not Started. Automated process will occur, once the task is completed.

Figure 32. Changed task status with updated description

6.1 Task flow PHP logic

Essentially, all the logic, which gets executed after a user performs an action, is located inside the SugarCRM system, on the server where it is deployed. Every module whether it is the Lead or the Task defines a PHP script which gets executed when a user interacts with the SugarCRM UI. The script can be customized according to the company's needs by adding the properties which have to be tracked.

```
public function executeBusinessProcess($bean, $event, $arguments) {
// call on updated records only
$datetime =  TimeDate::getInstance()->getNow(true)->modify("+1 days")->asDb();

if ( isset (self::$fetchedId[$bean->id]) && $bean->fetched_row['status'] !=
self::$fetchedRow[$bean->id]['status'] ){

if (self::$fetchedRow[$bean->id]['status'] == 'New' && $bean-
>fetched_row['status'] == 'Assigned'){

$task = new Task();
$task->status = "Not Started";
$task->name = "II 10 Checkliste Informationen übertragen";
$task->priority = "High";
// Speicher
$task->assigned_user_id = "ab6ea65b-6b0a-4934-7216-54329a62d1aa";
$task->description = "";
$task->parent_id = $bean->id;
$task->date_due = $datetime;
$task->save();
$bean->load_relationship('tasks');
$bean->tasks->add($task->id);
$bean->save();
}
```

Code 22 Automated Task creation script

The Code 22 demonstrates the code which gets executed behind when a user changes a status of the Lead from **New** to **Assigned.** The same way the Task properties can be tracked in the corresponding PHP script. Code 23 shows a simple PHP script which checks for the Task status and if it is **Completed** then a new Task object gets created and saved to the system.

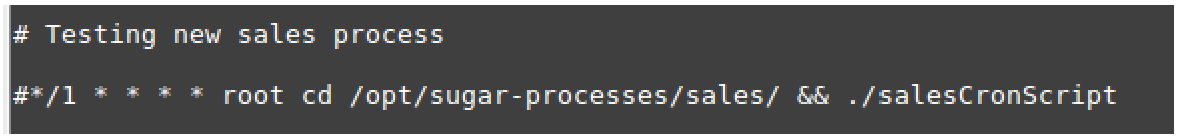
```php
<?php
if( !empty($bean->id) && ($bean->status == "Completed")){

if($bean->name == "V 40 Installationstermin vereinbaren"){

$task = new Task();
$task->status = "Not Started";
$task->name = "V 50 Zugangsdaten versenden";
$task->priority = "High";
$task->assigned_user_id = "bf7f86cf-7317-35e2-4f61-54183926cff2";
$task->description = "";
$task->date_due = $datetime;
$task->parent_id = $bean->id;
$task->save();
$accounts = $bean->get_linked_beans('accounts','Account');
foreach ($accounts as $account){

$account->load_relationship('tasks');
$account->tasks->add($task->id);
$account->save();

}
}
}
```

Code 23 Automated Task status tracker

By using the functionality which SugarCRM exposes to the developer, such as automation scripts, makes it straightforward to implement various custom workflow mechanism based on what employees require. It gives a powerful method to trigger any action in response to the user's interaction inside the system. For instance, an email can be sent from the script when a status of the Task object gets changed.

6.2 Linux cron jobs

Finally, the whole system needs to operate without the human involvement. The scripts which perform the work have to be initiated on a regular basis, for instance every workday in the morning. In order to implement the automated script execution, the Linux cron job was developed.



```
# Testing new sales process

#*/1 * * * * root cd /opt/sugar-processes/sales/ && ./salesCronScript
```

Figure 33 Cron job implementation

Figure 33 demonstrates the text file which contains the instruction for the cron job utility on how to execute the script which starts the whole customer acquisition process. The code pre cedes with a hash symbol which indicates a comment. I did this on purpose in order to test the system by executing the script every minute. The code first executes the cd command which moves to the appropriate folder in the file system and then perform the script execution. It is worth mentioning that the final script is run on behalf of the root user.

7 CONCLUSION

To conclude the work I did during this project, it is essential to observe the final state of the system and discuss the capabilities of the final product. Based on the set requirements the system performs flawlessly the following tasks:

- Scanning of the Caterva inbox dedicated for CSV files on a daily basis which contain lead personal data.
- Performing computations in order to parse CSV files, construct a data structure which contains the lead data and extract other file attachments like a PDF checklist.
- Sending the network requests to the SugarCRM system using a RESTful API interface which creates, updates and links new entries in the system.
- Automatically detecting when certain properties like task or lead statuses are changed in order to move the customer acquisition process forward by using existing SugarCRM functionality.

The project is continuously maintained by Caterva and the new requirements will add up in the future. This means that the system will be improved further and new features will be added. For instance, the company needs to integrate battery installation companies into the process which will require to implement a calendar event system capable of being in sync with the rest of the process which is implemented by me at this stage. In addition, the current state of the system does not provide any feedback to the responsible employees which have to initiate actions when certain parameters change. Currently, they need to be logged in the SugarCRM and track important information. As a future feature request, it is a clever idea to implement an automatic email notification when the action from a person is pending.

Finally, the result of the project made crucial improvements to an existing customer acquisition process which was handled manually by one person. When the company grew then the demand for the automatisation increased. Currently the system handled more than thirty new leads on one day by adding them to the CRM system within seconds. It is obvious that the previous manual approach would have been fully irrelevant and slow. In my personal opinion, I am convinced that the product I built carries helpful features to the company which improve their business operations and let the gain more customers over time. It is especially rewarding to witness how maintenance of the product became a requirement for the future generation of interns at Caterva.

BIBLIOGRAPHY

IETF Tools. The OAuth 2.0 Authorization Framework, RFC6749. WWW document. Available at: https://tools.ietf.org/html/rfc6749 Updated 01 October 2012. Referred 20 August 2016.

NGDATA website 2016. What is customer acquisition? NGDATA USA. WWW document. Available at: http://www.ngdata.com/what-is-customer-acquisition/ Updated 17 June 2016. Referred 20 July 2016

Service Architecture website. Representational State Transfer (REST). WWW document. Available at: http://www.service-architecture.com/articles/webservices/representational_state_transfer_rest.html No Update Information. Referred 03 August 2016

Salesforce website 2016. What is CRM? Salesforce USA. WWW document. Available at:

http://www.salesforce.com/eu/crm/what-is-crm.jsp No Update information. Referred 20 July 2016

TechTarget website 2016a. CRM. TechTarget USA. WWW document. Available at:

http://searchcrm.techtarget.com/definition/CRM No Update Information. Referred 20 July 2016

TechTarget website 2016b. Application Programming Interface. WWW document. Available at: http://searchexchange.techtarget.com/definition/application-program-interface Updated 01 July 2014. Referred 20 August 2016

TechTarget website 2016c. RESTful API. WWW document. Available at: http://searchcloudstorage.techtarget.com/definition/RESTful-API

Updated 01 December 2016. Referred 10 March 2017

Wikipedia website 2016a. Customer Relationship Management. Wikipedia Foundation, Inc. WWW document. Available at: https://en.wikipedia.org/wiki/Customer_relationship_management Updated 02 August 2016. Referred 03 August 2016

Wikipedia website 2016b. SugarCRM. Wikipedia Foundation, Inc. WWW document. Available at: https://en.wikipedia.org/wiki/SugarCRM Updated 16 July 2016. Referred 03 August 2016

Wikipedia website 2016c. Software Stack. Wikipedia Foundation, Inc. WWW document. Available at: https://en.wikipedia.org/wiki/Solution_stack Updated 08 June 2016. Referred 03 August 2016

Wikipedia website 2016d. PHP. Wikipedia Foundation, Inc. WWW document. Available at: https://en.wikipedia.org/wiki/PHP / Updated 25 July 2016. Referred 03 August 2016

Wikipedia website 2016e. MySQL. Wikipedia Foundation, Inc. WWW document. Available at: https://en.wikipedia.org/wiki/MySQL

Updated 30 July 2016. Referred 03 August 2016

Wikipedia website 2016f. HTTP. Wikipedia Foundation, Inc. WWW document. Available at: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol Updated 30 July 2016. Referred 03 August 2016

Wikipedia website 2016g. Linux. WWW document. Available at: https://en.wikipedia.org/wiki/Linux Updated 30 July 2016. Referred 20 August 2016

Wikipedia website 2016h. Cron. WWW document. Available at: https://en.wikipedia.org/wiki/Cron

Updated 20 March 2017. Referred 03 April 2017

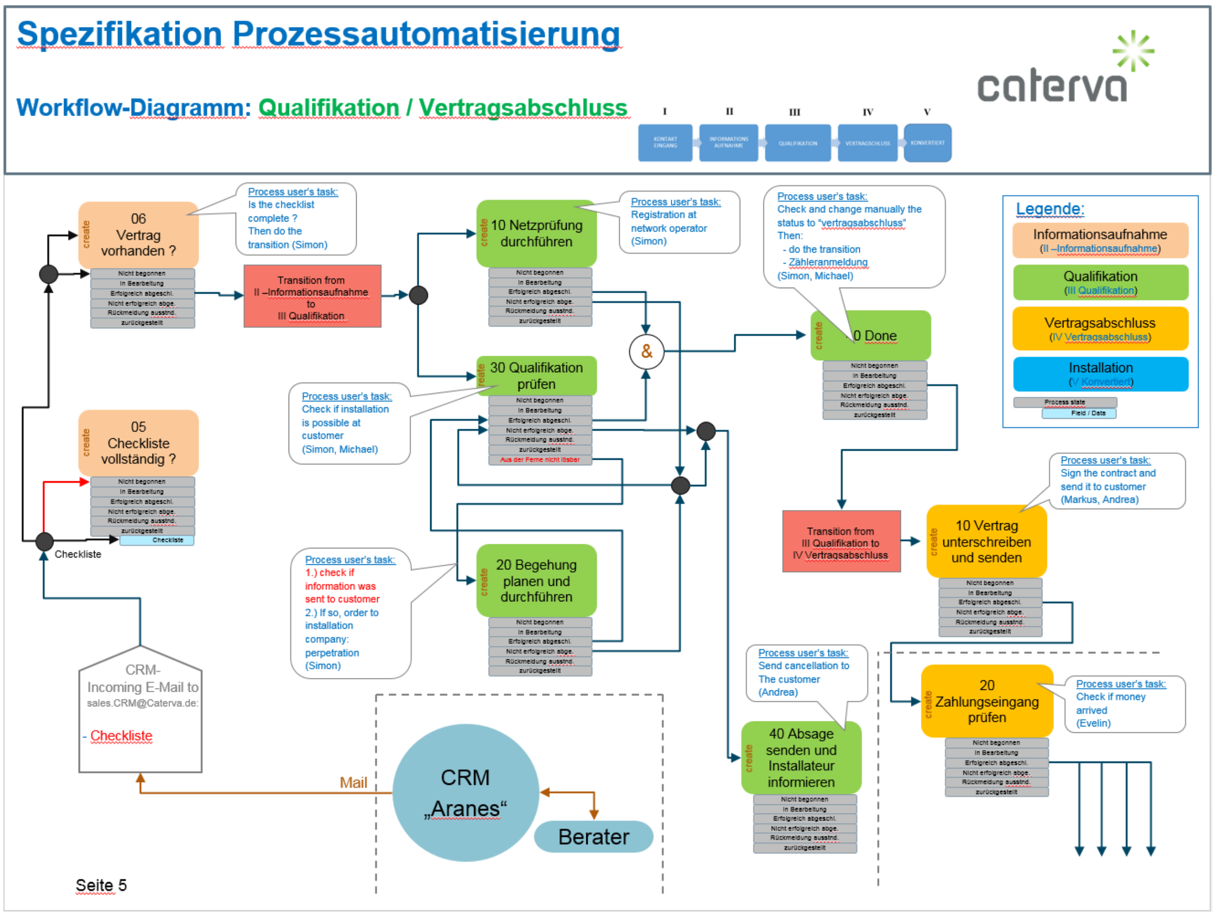
Wikipedia website 2016i. Apache HTTP Server. WWW document. Available at: https://en.wikipedia.org/wiki/Apache_HTTP_Server

Updated 13 April 2017. Referred 20 April 2017

APPENDICES

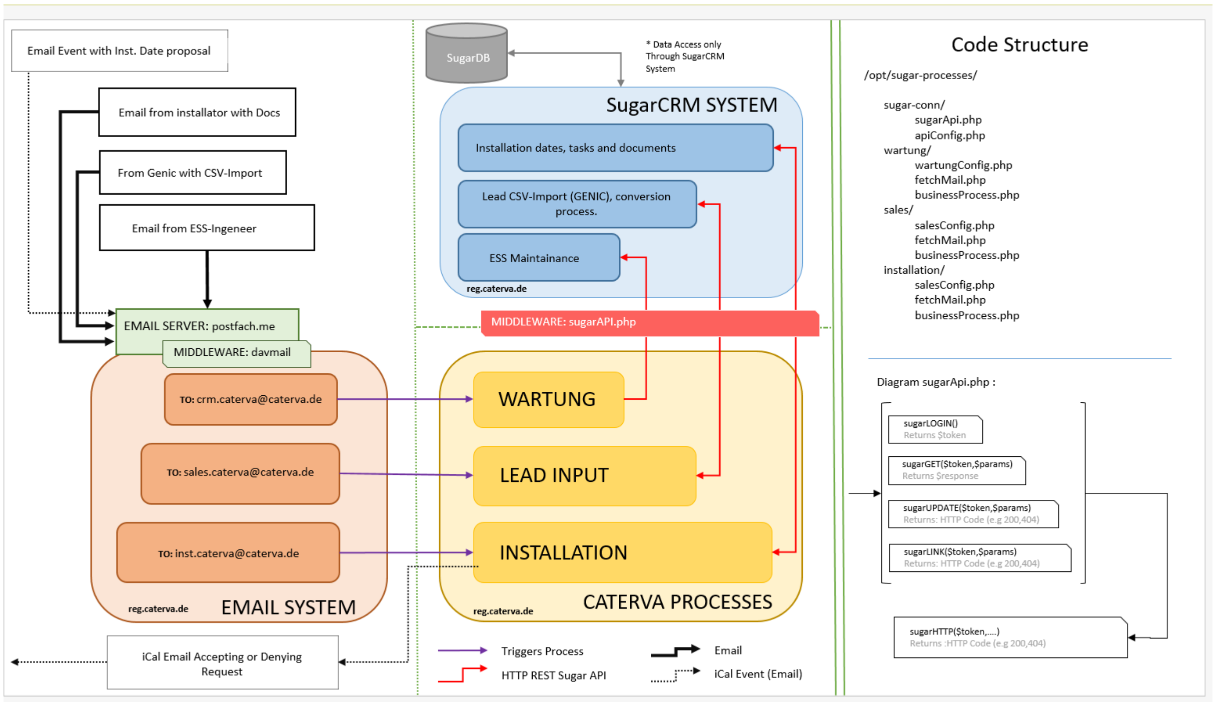
Appendix 1

New customer acquisition process requirement visualization


Spezifikation Prozessautomatisierung
Workflow-Diagramm: Qualifikation / Vertragsabschluss
caterva
I II III IV V
KONTAKT EINGANG
INFORMATIONS AUFNAHME
QUALIFIKATION
VERTRAGSSCHLUSS
KONVERTIERT
Legende:
Informationsaufnahme (II –Informationsaufnahme)
Qualifikation (III Qualifikation)
Vertragsabschluss (IV Vertragsabschluss)
Installation (V Konvertiert)
Process state
Field / Data
create
06 Vertrag vorhanden ?
Nicht begonnen
In Bearbeitung
Erfolgreich abgeschl.
Nicht erfolgreich abge.
Rückmeldung ausstnd.
zurückgestellt
Process user's task: Is the checklist complete ? Then do the transition (Simon)
Transition from II –Informationsaufnahme to III Qualifikation
10 Netzprüfung durchführen
Process user's task: Registration at network operator (Simon)
Process user's task: Check and change manually the status to "vertragsabschluss" Then: - do the transition - Zähleranmeldung (Simon, Michael)
0 Done
&
30 Qualifikation prüfen
Aus der Ferne nicht lösbar
Process user's task: Check if installation is possible at customer (Simon, Michael)
05 Checkliste vollständig ?
Checkliste
Process user's task: Sign the contract and send it to customer (Markus, Andrea)
Transition from III Qualifikation to IV Vertragsabschluss
10 Vertrag unterschreiben und senden
Process user's task: 1.) check if information was sent to customer 2.) If so, order to installation company: perpetration (Simon)
20 Begehung planen und durchführen
Process user's task: Send cancellation to The customer (Andrea)
20 Zahlungseingang prüfen
Process user's task: Check if money arrived (Evelin)
CRM- Incoming E-Mail to sales.CRM@Caterva.de:
- Checkliste
Mail
CRM „Aranes“
Berater
40 Absage senden und Installateur informieren
Seite 5

Appendix 2

Full software system structure


Email Event with Inst. Date proposal
Email from installator with Docs
From Genic with CSV-Import
Email from ESS-Ingeneer
EMAIL SERVER: postfach.me
MIDDLEWARE: davmail
TO: crm.caterva@caterva.de
TO: sales.caterva@caterva.de
TO: inst.caterva@caterva.de
reg.caterva.de
EMAIL SYSTEM
iCal Email Accepting or Denying Request
SugarDB
* Data Access only Through SugarCRM System
SugarCRM SYSTEM
Installation dates, tasks and documents
Lead CSV-Import (GENIC), conversion process.
ESS Maintainance
reg.caterva.de
MIDDLEWARE: sugarAPI.php
WARTUNG
LEAD INPUT
INSTALLATION
reg.caterva.de
CATERVA PROCESSES
Triggers Process
HTTP REST Sugar API
Email
iCal Event (Email)
Code Structure
Diagram sugarApi.php :
sugarLOGIN()
Returns $token
sugarGET($token,$params)
Returns $response
sugarUPDATE($token,$params)
Returns: HTTP Code (e.g 200,404)
sugarLINK($token,$params)
Returns: HTTP Code (e.g 200,404)
sugarHTTP($token,....)
Returns :HTTP Code (e.g 200,404)

```
/opt/sugar-processes/

    sugar-conn/
        sugarApi.php
        apiConfig.php
    wartung/
        wartungConfig.php
        fetchMail.php
        businessProcess.php
    sales/
        salesConfig.php
        fetchMail.php
        businessProcess.php
    installation/
        salesConfig.php
        fetchMail.php
        businessProcess.php
```

Appendix 3

Foundational codebase for RESTful API requests.

```php
/**
 * Generic function to make cURL request.
 * @param $url - The URL route to use.
 * @param string $oauthtoken - The oauth token.
 * @param string $type - GET, POST, PUT, DELETE. Defaults to GET.
 * @param array $arguments - Endpoint arguments.
 * @param array $encodeData - Whether or not to JSON encode the data.
 * @param array $returnHeaders - Whether or not to return the headers.
 * @return mixed
 */
function api_call(
    $url,
    $oauthtoken='',
    $type='GET',
    $arguments=array(),
    $encodeData=true,
    $returnHeaders=false
)
{
    $type = strtoupper($type);

    if ($type == 'GET')
    {
        $url .= "?" . http_build_query($arguments);
    }

    $curl_request = curl_init($url);

    if ($type == 'POST')
    {
        curl_setopt($curl_request, CURLOPT_POST, 1);
    }
    elseif ($type == 'PUT')
    {
        curl_setopt($curl_request, CURLOPT_CUSTOMREQUEST, "PUT");
    }
    elseif ($type == 'DELETE')
    {
        curl_setopt($curl_request, CURLOPT_CUSTOMREQUEST, "DELETE");
    }

    curl_setopt($curl_request, CURLOPT_HTTP_VERSION, CURL_HTTP_VERSION_1_0);
    curl_setopt($curl_request, CURLOPT_HEADER, $returnHeaders);
    curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
    curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

    if (!empty($oauthtoken))
    {

        $token = array("oauth-token: {$oauthtoken}");
        curl_setopt($curl_request, CURLOPT_HTTPHEADER, $token);
    }

    if (!empty($arguments) && $type !== 'GET')
    {
        if ($encodeData)
        {
            //encode the arguments as JSON
            $arguments = json_encode($arguments);
        }
```