

Information Page Internship / Graduation Report

Fontys University of Applied Sciences

School of Technology and Logistics

Post Office Box 141, 5900 AC Venlo, Netherlands

Type of report:	Bachelor thesis
Student name:	Jurian Janssen
Student number:	2573644
Study:	Informatics – Business Informatics
Period:	August 2018 – January 2019
Company name:	TYPO3 GmbH
Address:	Am Wehrhahn 41
Postal code + City:	40211 Düsseldorf
Country:	Germany
Telephone:	+49(0)2212054360
Company supervisor:	Mathias Schreiber
Supervising Lecturer:	Stefan Sobek
External commissioner:	Mr. Kersemakers
Company Confidential:	No
Number of words:	11972

Marketing Automation with TYPO3

Bachelor Thesis

Jurian Janssen

Düsseldorf, January 7, 2019

Information Page

Fontys Hogeschool Techniek en Logistiek
Postbus 141, 5900 AC Venlo

Bachelor Thesis

Name of student: Jurian Janssen
Student number: 2573644
Course: Informatics - Business Informatics
Period: August 2018 - January 2019

Company name: TYPO3 GmbH
Address: Am Wehrhahn 41
Postcode, City: 40211, Düsseldorf
Country: Germany

Company coach: Mathias Schreiber
Email: mathias.schreiber@typo3.com
University coach: Stefan Sobek
Email: s.sobek@fontys.nl

Examinator: Geert Monsieur
External domain expert: Mr. Kersemakers

Non-disclosure agreement: Yes (but not for the project itself)

Summary

TYPO3 is an enterprise content management system. This project will build an integration between a marketing automation system (in this case Mautic) and TYPO3.

A research was conducted in order to determine the marketing automation solution that would best fit this project. Following that, an analysis was conducted in order to determine the scopes and requirements of this extension.

The extension is written in PHP. Using Travis CI for continuous integration and Github for collaboration.

Software designs and implementations were made for the extension that allow for: synchronizing forms between TYPO3 and Mautic, tracking users on a TYPO3 website via Mautic, displaying dynamic content based in TYPO3 based on decision made by Mautic. These features were built using the Mautic API library package.

A fully object oriented TYPO3 extension written in PHP connecting the content management system with Mautic was delivered in the end.

Samenvatting

TYPO3 is een enterprise content management system. Dit project draait er om een integratie (extensie) te bouwen tussen een marketing automation systeem (in dit geval Mautic) en TYPO3.

Er is onderzoek gedaan naar de verschillende aanbieders van marketing automation software om te zien welke aanbieder het beste bij het project past. Daarnaast heeft er een analyse plaatsgevonden om de requirements en scopes van de extensie vast te leggen.

De extensie is geschreven in PHP. Daarnaast is Travis CI gebruikt voor continuous integration en Github voor samenwerking tussen ontwikkelaars.

Voor alle uiteindelijke features zijn er design-en-implementatiefases doorlopen. Deze features zijn: synchronisatie van formulieren tussen TYPO3 en Mautic, gebruikers volgen over een TYPO3 website via Mautic, dynamische content weergeven in TYPO3 op basis van Mautic data. Al deze features maken gebruik van het Mautic API library pakket.

Een volledig object-georiënteerde TYPO3 extensie met alle hierboven genoemde functionaliteiten is het eindresultaat van dit project.

Acknowledgments

Developing this project would never have been possible without the amazing support from both the TYPO3 community and the Mautic community. A special thanks goes out to the following people for their contributions, support and amazing help during the development of this project. Without them, this project would never be where it stands today.

- Nicole Cordes
- Ekkehard Gumbel - *Bitmotion GmbH*
- Helmut Hummel
- John Linhart - *Mautic Inc.*
- Timo Poppinga - *Zdreicom*
- Florian Wessels - *Bitmotion GmbH*

Also thanks to the companies *Beech* (Venlo, NL) and *Bitmotion GmbH* (Hannover, DE) for sharing their thoughts on Marketing Automation in relation to TYPO3 with me.

Another thank you to the organizers of the events *Meet TYPO3 Rotterdam* and *TYPO3 Camp RheinRuhr* for giving me the opportunity to present my work and gather impressions and feedback.

A big thank you goes out to my colleagues at the *TYPO3 GmbH*. Both for supporting me whenever I was in need of help and for making my time at the office an unforgettable experience.



"I was amazed to see the work Jurian and his team have done to integrate the TYPO3 CMS with Mautic. They do a magnificent job demonstrating what an integrator can do with the Mautic platform."

David Hurley

*Founder at Mautic
Boston, United States*

"As contributor of several CMS plugins connecting Mautic including Wordpress, Joomla, Drupal and Grav I'm amazed what Jurian and contributors did with TYPO3. It's by miles the most powerful CMS integration Mautic has so far."

John Linhart

*Software Engineer at Mautic Inc.
Prague, Czech Republic*

"At Webmecanik, being the first Mautic Open Source vendor, we believe that the CMS integration (the deepest offered by the community) Jurian and contributors have developed, is an incredible bridge for the TYPO3 users to enjoy fully the power of open source marketing automation."

Norman Pracht

*General Manager at Webmecanik
Annecy, France*

"This integration is a game changer for us as a TYPO3 agency. We can now provide deep, intelligent integration with Mautic, with clever features and much better user experience than what would normally be possible."

Ekkehard Gumbel

*CEO at Bitmotion GmbH
Hannover, Germany*

Statement of Authenticity

I, the undersigned, hereby certify that I have compiled and written the attached document / piece of work and the underlying work without assistance from anyone except the specifically assigned academic supervisors and examiners. This work is solely my own, and I am solely responsible for the content, organization, and making of this document / piece of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document / piece of work will not be accepted for evaluation or for the award of academic credits if it is determined that it has not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document / piece of work and the underlying work will not be accepted for evaluation or for the award of academic credits if it is determined that it embodies plagiarism.

Name:	Jurian Janssen
Student Number:	2573644
Place/Date:	Düsseldorf, January 7, 2019

Signature:

Contents

1	Introduction	1
1.1	Background and Context	1
1.2	Goal	1
1.3	TYPO3 GmbH	1
1.4	Team	2
1.5	Infrastructure and Organization	2
2	Analysis	3
2.1	Project	3
2.2	Target audience	3
2.3	Marketing Automation Systems	3
2.3.1	Oracle Eloqua	4
2.3.2	Hubspot	4
2.3.3	SalesForce	4
2.3.4	Mautic	4
2.4	Tools	5
2.4.1	Composer	5
2.4.2	PHPStorm	5
2.4.3	Github	5
2.4.4	Travis CI	6
2.4.5	LaTeX	6
2.5	Requirements	6
2.5.1	Lead tracking	7
2.5.2	Forms	7
2.5.3	Serving dynamic content	7
2.5.4	Installation	7
2.5.5	Security	7
2.6	Quality Management	8

2.6.1	Testing	8
2.6.2	Code standards	8
2.7	Planning	9
3	Research	10
3.0.1	Questions	10
3.0.2	Methods	10
3.0.3	Systems	10
3.1	Lead generation	11
3.1.1	Hubspot	11
3.1.2	Mautic	11
3.2	Lead segmentation	11
3.2.1	Hubspot	11
3.2.2	Mautic	12
3.3	Costs	12
3.3.1	Hubspot	12
3.3.2	Mautic	12
3.4	System requirements	12
3.4.1	Hubspot	12
3.4.2	Mautic	12
3.5	Security	13
3.5.1	Hubspot	13
3.5.2	Mautic	13
3.6	Analysis results	13
3.6.1	Extension requirements	13
3.6.2	API requirements	13
3.6.3	Costs	13
3.6.4	Opinions from stakeholders	14
3.6.5	Software philosophy	14
3.6.6	Security	14
3.7	Concluding	14
4	Design	16
4.1	Authorization and security	16

4.1.1	Handling credentials	17
4.1.2	Handling errors and warnings	17
4.2	Tracking	18
4.3	Dynamic Content	19
4.3.1	Personas	19
4.4	Forms	20
4.4.1	Data structures	20
4.4.2	Transformation	21
4.4.3	Mapping lead properties	21
4.4.4	Data storage	22
4.5	Installation	22
4.5.1	Extension manager	23
4.5.2	Composer	23
5	Implementation	24
5.1	Authorization	24
5.2	Tracking	26
5.3	Forms	27
5.3.1	Form synchronization	27
5.3.2	Form finishers	29
5.3.3	Submitting forms	30
5.4	Dynamic Content	31
5.4.1	Fetching Mautic segments	31
5.4.2	Mapping segments to personas	32
5.4.3	Serving dynamic content	33
6	Conclusion	34
7	Recommendations	35
7.1	Mautic API	35
7.2	Automatically filling forms	35
7.3	Creating more variations of finishers	35
	Appendices	39

List of Figures

2.1	A Gantt chart representing the planning	9
4.1	A simplified overview of system components	16
4.2	A class implementation of a dependency injection between the authorization object and a repository class	17
4.3	A flash-message example	18
4.4	A class implementation of a the tracking service	19
4.5	A simplified visualization of personas	20
4.6	A form field in the Mautic UI that is matched to 'lead email'	22
4.7	A form field in the TYPO3 UI that is matched to 'lead email'	22
5.1	Authorization form in the TYPO3 extension manager	24
5.2	Tracking form in the TYPO3 extension manager	26
5.3	MauticPointsFinisher configuration in the backend	30

List of Tables

7.1 Project Skills Level 36

Acronyms

API: Application Programming Interface

CI: Continuous Integration

CMS: Content Management System

GmbH: Gesellschaft mit beschränkter Haftung (Limited liability company)

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

MA: Marketing Automation

OAuth: Open Authorization

PHP: Hypertext PreProcessor

PSR: PHP Standards Recommendation

REST: Representational State Transfer

TDD: Test Driven Development

UI: User Interface

Terminology

Composer: A dependency management tool for PHP projects.

Continuous Integration: In software engineering, continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. (*Continuous Integration* 2018)

GitHub: A public version control system based on the Git protocol.

Marketing Automation: Marketing automation refers to software platforms and technologies designed for marketing departments and organizations to more effectively market on multiple channels online (such as email, social media, websites, etc.) and automate repetitive tasks. (Koetsier 2018)

Mautic: An open source marketing automation system that lives on GitHub.

Travis: A continuous integration tool for open source software.

TYPO3: An open source enterprise content management system.

Chapter 1

Introduction

This document will contain a detailed description of all the work done surrounding the internship of the Author at the TYPO3 GmbH over the period of August 2018 till January 2019. This internship is done in the faculty of Business Informatics. The goal of this internship is to create an integration between a marketing automation solution and the TYPO3 CMS.

1.1 Background and Context

Marketing Automation is a relatively new concept in the world of web development. However, many companies slowly start to realize the value of such systems in terms of generating leads (contacts) and providing personalized marketing journeys to their customers. In short, marketing automation is the process of automating generic marketing tasks such as lead collection, lead filtering, lead nurturing and sending out marketing messages over various channels (i.e. email, sms, social media). As this new trend is on the rise, TYPO3 wants to have a quality hands-on integration for such systems. Having a well thought out and functioning integration can be a good selling point for the system in the future. À propos it can not only prove to be a win on the grounds of development, but also on the grounds of marketing the system as a whole.

1.2 Goal

The goal of the project is to create a well thought out marketing automation integration within the TYPO3 system. This includes (and requires) research on the different marketing automation solutions available at this moment. This is done in order to get a clear view of the entities, functionalities and features that will be needed to create a useful integration between the two systems.

In the end, the products will be a feature analysis of the different systems available for marketing automation, as well as an integration between TYPO3 and one or multiple marketing automation systems.

1.3 TYPO3 GmbH

The TYPO3 GmbH is a company located in the heart of Düsseldorf, Germany. First, one must note that despite the name, the GmbH is not a TYPO3 agency. Its main purpose is to market and support TYPO3 as a product. In short this means that the GmbH does not do TYPO3 projects. What it does do however is offer extended support for versions that are no longer supported by the community (eLTS) and offer service level agreements to businesses using TYPO3.

As a second responsibility the GmbH promotes the TYPO3 product as a whole. This is done in numerous ways. Having a stand at events, giving talks at developer conferences and selling merchandise just to name a few. The general intent of this responsibility should be clear; expanding the TYPO3 world beyond the markets it already has. This also includes overseas markets in the America's, where TYPO3 is trying to gain a stronger foothold.

1.4 Team

The TYPO3 GmbH has 15 employees. Their fields of profession vary widely from development to account managing (PR) to finances to video editing. The company is lead by Mathias Schreiber, the CEO of the GmbH. Not all employees work in Düsseldorf, some also work remotely. Either from home, or from an office space elsewhere.

Many other companies are involved with the GmbH. They are known as TYPO3 partners. Partners are companies that possess qualified expertise within the TYPO3 project. In short, this means that the company has an X amount of TYPO3 certified developers, integrators, editors and consultants, as well as that the company has applied for partnership with the GmbH. Certifications can be obtained from the TYPO3 association (note that this entity is entirely separate from the GmbH) through the certification committee. To get certified an individual must pass a written exam.

1.5 Infrastructure and Organization

The GmbH uses multiple tools to assure a streamlined workflow. Peer-to-peer communication is done via Slack. Slack is a channel based communication tool that supports text, voice and video chat. For ticketing and development Jira and Bitbucket are used. Where Jira is the ticketing (issues) system and Bitbucket is the version control system. However, collaborative projects may differ in this regard and use other systems such as GitHub or Forge. This depends on the other parties involved.

Planning and organizing is done via Scrum. Every morning at 10:00 there is a daily scrum meeting on Slack. All employees are involved in this meeting, this also means the non-developers. During this meeting each person will present what they have done the day before and what they will do today. Short discussions are allowed, but are usually done after the 'daily'. Next to that there is a conference room where more general issues are discussed if it need be.

Chapter 2

Analysis

This chapter will cover the analysis part of the project in detail. To find a rough overview without all the details, refer to appendix A: project plan.

2.1 Project

The project in general is to create a well thought out connection between the TYPO3 content management system and a (or multiple) marketing automation solution(s).

This chapter will list certain aspects of the project such as requirements, tools, planning etc. In depth explanation about these various topics can be found in this chapter.

2.2 Target audience

The target audience for this project are agencies that create TYPO3 websites, and their customers. Most of these can be categorized under small and medium sized businesses. See the TYPO3 partnered agencies on [TYPO3.com](https://typo3.com) to see how many TYPO3 certified employees are working there. This usually lies anywhere between 2 and 40. (*Find a TYPO3 partner* 2018)

The goal is to target as many TYPO3 users and agencies as possible. In order to encompass this group of users, two main factors need to be taken into account: functionality and costs.

Functionality is a major factor. In order to create a well designed system between TYPO3 and a marketing automation solution, the marketing automation solution should have a broad selection of functionality and features we can work with. Of course, these both need to be present and work consistently.

Costs is another factor. Depending on the size of the user or user company, it will vary what their budget is or can be for marketing automation. We can assume that one thousand euro per month is not a major issue for larger companies, but it may well be for smaller companies or freelancers.

2.3 Marketing Automation Systems

The following list is comprised of the four most present tools regarding marketing automation. Each of them chosen because of their unique characteristics. This was done to get a varied list, in order to find out which solution with what characteristics fits the best with TYPO3. Why these were chosen can be read in Appendix C.

Hubspot's main characteristic is that it is currently the market leader in Marketing automation. Salesforce

was chosen because of its countless other tools that integrate directly with its marketing automation solution. Oracle was chosen because it is a well established brand in the tech world. Lastly, Mautic was chosen because it is free to use and open source.

2.3.1 Oracle Eloqua

Eloqua is a paid, closed source marketing automation system developed by Oracle. It has all the classic marketing automation features one would expect. Lead generation, campaign building, lead segmentation and means to contact leads. (*Oracle Eloqua: Drive Dynamic Journeys* 2018) All this data is also accessible via an API.

Eloqua also has an API that allows us to both work with lead segments and forms (as per the requirements defined later on in this document).

However, Eloqua has a starting price of two thousand US Dollar per month. Which is quite expensive for smaller businesses. This plan allows up to ten users in the system.

Eloqua is a cloud solution, and cannot be hosted at third parties or in-house.

2.3.2 Hubspot

Hubspot is a paid, closed source marketing automation system. Currently, Hubspot is the market leader when it comes to marketing automation. (*Best Marketing Automation Software* 2018)

Hubspot has all the main features of marketing automation out of the box. Lead generation, campaign building, lead segmentation and means to contact leads. All this data is also accessible via an API.

The hubspot product starts at a price of seven hundred euro a month. This allows you to have one thousand contacts. You can pay more to increase this limit.

Hubspot is a cloud solution, and cannot be hosted at third parties or in-house.

2.3.3 Salesforce

SalesForce has a marketing automation solution called Pardot.

Pardot has all the main features of marketing automation out of the box. Lead generation, campaign building, lead segmentation and means to contact leads. All this data is also accessible via an API.

Pardot starts at a price level of one thousand two hundred and fifty euro. This allows you to have up to ten thousand contacts.

2.3.4 Mautic

Mautic is an open source and free to use marketing automation system written in PHP.

Mautic has all the main features of marketing automation out of the box. Lead generation, campaign building, lead segmentation and means to contact leads. All this data is also accessible via an API.

As is with open source solutions, there are no costs associated with Mautic other than a hosting environment.

Mautic is supported by a community of developers.

2.4 Tools

2.4.1 Composer

Composer is the de facto dependency management system for PHP projects. It allows developers to manage project dependencies in one place. When developing, the developer can define the dependencies in a composer JSON file. Composer then downloads all the dependencies for you at the latest version (unless specifically instructed to pull an earlier version). Once the dependencies are downloaded, Composer will write a composer LOCK file. This file contains exactly which versions were downloaded and when. This lock file can then be shared with other developers via a version control system. It makes sure that all developers have exactly the same dependencies at the same version.

There are several reasons why Composer was chosen as the dependency management tool for this project. The first one being that TYPO3 also uses Composer, and since this project is a TYPO3 extension it would only be a logical assumption that the extension should be installable through Composer as well.

Secondly, like stated before, Composer is the de facto dependency management system in the PHP world (Hampton 2014). This tool is a given when developing PHP projects with external dependencies, hence it is not so much a matter of decision to use this tool, it is simply a must.

Lastly, Composer allows multiple developers to make sure they all have the same dependencies installed. Since this project is being worked on by multiple developers in multiple locations, a tool is needed that guarantees unity when it comes to dependencies. Composer gives this guarantee.

To see an extract from a Composer json file, please refer to 4.5.2.

2.4.2 PHPStorm

PHPStorm is an IDE, or Integrated Development Environment specifically designed for PHP projects. There are multiple tools on the market that fulfill this purpose. Microsoft has VSCode and NetBeans also has a dedicated PHP environment.

In the end all of these environments do a fine job at making the life of a developer easier. One can think of features like integrated version control systems, intelligent code completion and static code analysis.

The decision to use PHPStorm comes down to flavour. The author is familiar with PHPStorm and likes PHPStorm. Reasons for this are the intelligent code completion that comes with PHPStorm and the fact that PHPStorm can automatically fix code standard errors (which will be talked about at a later point).

In the end PHPStorm was chosen by the author mainly because it is his tool of preference.

2.4.3 Github

Github is an online version control system based on the git technology. Github is favoured by many open source development teams as it is completely cloud based and entirely free to use (as long as your code is publicly available). Next to that Github has many integrations with other services in the PHP environment such as Packagist (which is the service where Composer gets its dependency packages from) and Travis CI, which allows you to automate all testing of your project. This creates a hub where open source projects can do most, if not all their important tasks in one place (hence the name, Github).

Github covers multiple criteria for this project. First, the project has more than one developer, and these developers should be able to work together on one project via version control. Following up on that, the

project was always meant to be open source, and thus making the code publicly available on Github simply makes sense. Not only does this allow other people to view the code, but it also allows outsiders to improve on the code and propose their changes to us via so called "pull requests".

Then we have the integrations with Travis CI and Packagist, which are both of importance to us. The project will be available as a composer installed dependency, and thus it will also need to live on Packagist (see 2.4.1). Github takes care of the Packagist releases automatically, giving us the option of making one central release on Github, while Github takes care of the rest (Šufraj 2015).

Lastly Github can also be integrated with Travis CI. Every time someone makes a code change in the repository Github will tell Travis to create a build, run all the tests and check the code style. This way, the team can always be sure that certain changes did not have any unintended effects on the project. Merging code can then also be limited. Meaning that when Travis tests do not pass, Github will not allow you to perform any merge actions until all tests return positive. This can be seen as a quality control enforcer.

2.4.4 Travis CI

Travis CI is a continuous integration tool. Continuous integration is a practice in the software development world that runs checks on shared code repositories whenever certain actions are performed. These can include but are not limited to running tests and checking code style of new changes. This allows developers to make sure that changes made to the code have no unintended consequences and are in line with the standards of the project (*Continuous Integration* 2018).

There are other CI alternatives out there. One well know example is Jenkins. However, Jenkins must be self hosted, meaning that a server where it runs on is required. With Travis, this is different. Travis allows you to connect your public (this is key) repository to its systems. You can then configure on what actions Travis is triggered via Github. The same is true for another major player, Gitlab. Gitlab also requires a self hosted instance of the software, which is simply too much relative to the scale of this project.

The biggest plus of Travis is that you do not need to self host it. As long as your repository is public, you will be able to run all your CI related jobs on machines provided by Travis for free. The combination of easy to use, easy to set up and free is what made us decide to use Travis as a CI system.

Travis will handle both the software tests as well as the code style checks for this project.

To see an example of such a configuration for this particular project, see Appendix B: Travis configuration.

2.4.5 LaTeX

LaTeX was used to create this document. The reason being that this is the personal preference of the author.

2.5 Requirements

Gathering the requirements for this project was done by looking at two businesses currently using TYPO3 and marketing automation in production environments. By listening to their needs and ideas requirements for a functional, useful extension can be created.

The two companies involved in this endeavor were Beech, situated in Venlo, The Netherlands. And Bitmotion, situated in Hannover, Germany.

2.5.1 Lead tracking

Both companies agreed that in order to properly use the functionalities of marketing automation in combination with a content management system a functioning integration must be built. Both companies want to use the page visit history of users in order to later use them in marketing decisions. In essence this means that certain page visits should be available to the marketing automation system, so that these can be used in other elements like campaigns or segments.

Having the history of contacts can also be used retroactively when a contact has been identified. These history elements can then be used either by the marketing automation system or by humans to make decision. What the nature of these decisions is is up to the users.

2.5.2 Forms

According to Beech, their main source of acquiring user data is via forms on a website. Both TYPO3 and the marketing automation solution have their own implementation of forms. Beech says they would like their employees to be able to use their trusted TYPO3 environment while also connecting the forms to the marketing automation solution. During the form creation process in TYPO3 the form should be converted to a compatible structure and then sent over. Ideally the forms are kept in sync.

A natural result of this process is that created forms should also be able to be submitted by users. This data, after being submitted to TYPO3, should also be submitted to the marketing automation solution in order to use the data for contact identification.

2.5.3 Serving dynamic content

Bitmotion said they wanted to serve content to users via TYPO3 based on certain data from the marketing automation solution. Specifically the segments a user is in. A configuration element should be present with each TYPO3 content element in order to show or hide it from users who are or are not in a specific segment in the marketing automation solution.

2.5.4 Installation

There are two types of TYPO3 installations. A Composer mode installation and a non-Composer mode installation. A Composer mode installation is installed via the PHP dependency management tool Composer. Using Composer to install TYPO3 will force you to install your TYPO3 extensions also via composer. Installing TYPO3 without composer allows you to upload extensions as a zip file through the extension manager of TYPO3.

We want to support both types of installations. Meaning that there is a need for two packages. One installable via composer, and one zip file uploadable through the extension manager.

2.5.5 Security

Marketing automation deals with personal information. Email addresses, ip addresses, names, phone numbers etc. Handling this kind of sensitive information needs to be done with care. Transporting this information between TYPO3 and the marketing automation solution must be done in a secure way. Since this data is

transported over the HTTP protocol, the secured version of HTTP will be used, HTTPS. This is the standard communication protocol for both systems.

However, on a local development environment HTTPS may not always be present. In this case, the extension will allow non-secured HTTP connections, but will advise the user via the UI to secure the connection with HTTPS.

2.6 Quality Management

2.6.1 Testing

The TYPO3 GmbH works via the principles of TDD or Test Driven Development. This means that during development, you test your code frequently. Ideally you write the tests before you write the code. In this case you have thought of what parameters a piece of code takes, and what parameters it returns. You write the tests beforehand. Then you write the code. This way you can immediately test if the code behaves like you expect it to behave.

The second reason for testing is that the TYPO3 GmbH would like to include the integration into their industry partner program. Extensions in this program require a code test coverage of sixty percent or above. This is a quality requirement set by the GmbH.

2.6.2 Code standards

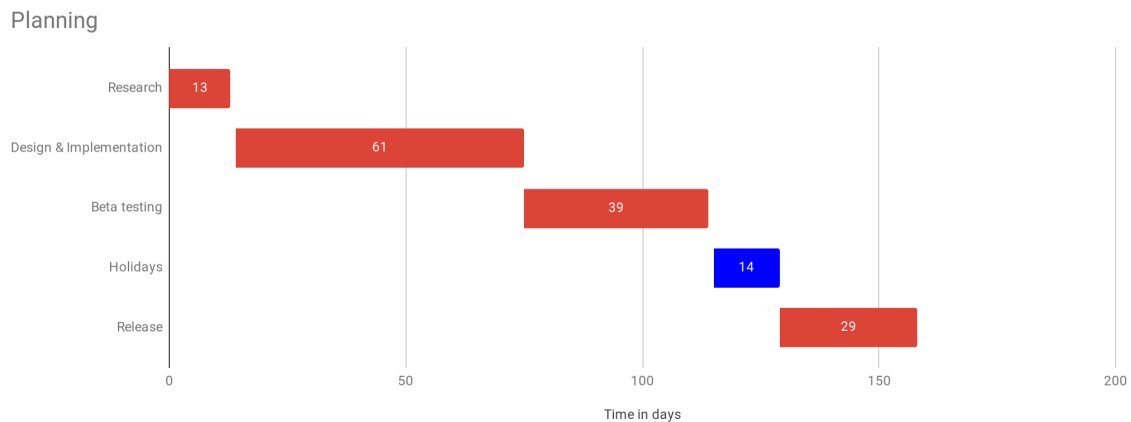
TYPO3 uses PSR-2 as their code style standards. The goal of using PSR-2 is creating an environment where all code is styled the same way, making it easier for developers (both in and outside of the TYPO3 world) to scan over the code base. (*PSR-2: Coding Style Guide* 2018)

Of course, building an extension, we are not forced to use any code standards. However, we would like other developers to join the development of this extension somewhere along the line. Having code standards and clean code will make it easier for them to read work done previously.

Lastly, good looking code simply has a more professional look to the outside world compared to having different styles here and there as a result of different developers writing different parts of the application.

2.7 Planning

Figure 2.1: A Gantt chart representing the planning



The duration of the project is set at five months. From September 2018 until January 2019. During this period certain stages of the development process need to be completed. These phases are research, design, implementation, user testing, beta release and stable release.

The research phase will consist of analyzing existing marketing automation solutions in order to determine which one fits best for an integration with TYPO3. This will consist mainly out of analyzing various features and the pricing of such systems. This phase has been given a time frame of two weeks.

The second phase will be combination of the application design and implementation. Since the workflow is done in sprints both design and implementation of separate parts of the application will be done in the same sprint. In essence this means that the application will be built part by part, design and then implement. For this reason those two aspects of the development have been put into one phase. This phase has been given a time frame of two months.

The third phase will consist of user testing and the beta release. These two aspects were grouped together as we want to gather as much feedback as possible before the actual stable release. Having an open beta will allow us to gather feedback from anyone interested in using the application without having to deal with an early access test release and the hassle of dealing with invites. During this phase the feedback gathered will also be implemented wherever possible. In total, this phase has been given a time frame of one month.

The last phase of the project is the release phase. During this phase the application will be released in an official repository. It will also be made available on Packagist, the package manager for PHP projects. During this phase we will aim at fixing any minor issues that might pop up. Next to that we will do our best to promote the application over various channels in order to gain a stable user base. It is then also the plan to keep the development up along with new releases of TYPO3.

Chapter 3

Research

In this chapter we will look at a narrowed down selection of marketing automation tools. Comparing their features and putting them into perspective against our extension requirements. At the end one suitable system will be chosen for this project based on the results of this analysis.

Before this chapter starts, a short note on the marketing automation systems. As mentioned in chapter two, we ended up with four systems. However, after discussion with various stakeholder companies and potential users, we determined that the costs for Oracle and Salesforce were too high to sufficiently target a broad TYPO3 audience. As such, the remaining contenders are Mautic and Hubspot.

3.0.1 Questions

An array of questions need to be answered in order to make this research the basis of the software project. The questions listed below will eventually determine the decision as to which system is the most suitable.

- How well does the system fit the extension requirements?
- How open is the system in terms of connecting external systems, for instance API's?
- How well does the system fit within the general philosophy of TYPO3?
- What are the costs of effectively employing the system in a marketing automation environment?

3.0.2 Methods

Two methods will be employed during this research. The first one is literature research. This mainly means reading and analyzing the code and API documentation. As a second method experiments will be conducted to test the said systems and see how it interacts with the outside world.

3.0.3 Systems

Our narrowed down list consists out of two major players in the marketing automation field at the moment. The first one is Hubspot. Hubspot is an all round system in the marketing world that comes with many features such as CRM, social media analysis and marketing automation. However, for this research we will be narrowing it down to the marketing automation part. This is done for two reasons, one for the sake of simplicity and the ability to make clear comparisons, and two, because Hubspot is not free, and including the extra modules would drive up the price. This would make the comparison unfair as we are strictly speaking about marketing automation as a standalone.

The second system is called Mautic. Mautic is an open source solution for marketing automation. It is a pure marketing automation tool, meaning that while it can integrate with existing CRM systems, it does not possess one on its own. I shall reference back to the previous paragraph where we stated this is about marketing automation only. Mautic fits that category perfectly.

3.1 Lead generation

Lead generation is the first major requirement of the extension. In essence, it means that the system should be able to take information submitted to or collected on the TYPO3 and turn them into personal profiles of users.

3.1.1 Hubspot

API Hubspot has several API endpoints meant to interact with lead objects. These lead objects are called contacts. Contacts can be created via HTTP POST requests to the Hubspot API. These requests can contain various parts of data about a lead. After the request is made, Hubspot will handle the data and create and/or update the corresponding lead in accordance with the data provided.

Tracking Hubspot tracks users on the website via cookies. These cookies contain unique information about that particular user. Via tracking every user will automatically be turned into an anonymous lead. This anonymous lead is simply an ip address or a cookie id until more information becomes available about this user through other means.

3.1.2 Mautic

API Mautic has several API endpoints for leads, which have recently been renamed to contacts. The contact API allows contact creation, update and delete, as well as some other methods that allow the contact to interact with other elements of the system. Requests are sent via HTTP and will be processed intelligently in case a contact already exists.

Tracking Mautic tracks users via cookie placement and a short javascript implementation on the website. In addition Mautic can also track users by ip address and/or device fingerprint. Though those are not as accurate as the cookie and should be used as a last resort.

3.2 Lead segmentation

Lead segmentation is the filtering of leads into certain groups (called segments). These segments can be used to target specific groups of leads. Segmentation can be done via various routes.

3.2.1 Hubspot

Hubspot allows segmentation of leads via workflows. These workflows are in essence a journey of events or characteristics a lead has or has done. Typically one would define these so that they fit a certain persona. This way leads can be grouped together that have similar (business) interests. In addition directly assigning a lead to a segment is always possible.

3.2.2 Mautic

Mautic has two main routes a lead can take to end up in a segment. The first one is by completing a segment trigger. Segment triggers are certain events a lead can complete, such as filling out a specific form on your website. When such an event is triggered, the lead is immediately added to the segment. The second route is via campaigns. These campaigns are similar to the workflows in hubspot. They are a tree based event journey that can modify what segments a lead is in (among many other things).

3.3 Costs

The costs of the tools should also be taken into consideration. Expensive tools greatly limit the user base of the extension, whereas it might also mean that it has more requirement that are supported.

3.3.1 Hubspot

The Hubspot marketing suite starter version costs forty-six euro per month. This version does support all requirements to satisfy the functionality of the extension. However, it does limit what the end user can do with the data they gather using the extension, as certain features are not included in the starter version. These features (such as landing pages) are included in the professional version, which costs seven hundred and forty euro per month.

3.3.2 Mautic

Mautic is an open source project, which means it is entirely free. However, this does mean that one has to pay for hosting the installation of Mautic. A decently performing hosting solution (as taken from Strato) would cost around twenty euro per month. Other than that, Mautic has no costs.

3.4 System requirements

This section shortly touches on the system requirements needed to properly run the software solutions.

3.4.1 Hubspot

Hubspot is a service and can only be purchased as a full software solution. Self hosting is not possible.

3.4.2 Mautic

Mautic is a self hosted solution. It requires a machine that runs PHP between versions 5.6 and 7.2. It also requires MySQL version 5.5 or higher. Mautic can be installed using the dependency management system Composer. However, downloading a zip file from the official website is also sufficient to install it for less technical users.

3.5 Security

This section will touch on the connection security towards the API of the marketing automation system.

3.5.1 Hubspot

Hubspot uses OAuth2 to authenticate the application (in this case TYPO3). OAuth2 allows a user to create tokens. These tokens are bound to certain application scopes. This means that tokens can be limited to only be able to use certain parts of the main application. The user grants access to certain scopes via an authorization grant. This authorization grant is then used to retrieve an access token from the Hubspot server. This access token can then be used to access Hubspot resource (and only if those fall within the granted scopes). (Anicas 2018)

3.5.2 Mautic

Mautic uses OAuth1a to authenticate the application (in this case TYPO3). OAuth1a uses OAuth credentials. The user is forwarded to the application with the credentials. Then the user has to identify themselves (usually with a password). If this is successful, OAuth tokens are returned, these can then be saved. Each request that accesses resources requires these tokens. Along with those, each request also contains a unique signature. This signature is generated from the request body, usually using HMAC-SHA1. The server generates the same signature from the request body. Requests only go through when the signatures are identical. If they are not identical there was either an error or the request has been tampered with. In this case, the request is dropped. (OAuth: The Authorization Flow 2018)

3.6 Analysis results

3.6.1 Extension requirements

Previously, requirements were defined which the marketing automation system should have in order for the extension to function properly. As noted before, both systems implement the needed concepts (albeit in a different way). These requirements are the foundation of the extension, and thus are a must if we are to continue implementing this software solution. Both Mautic and Hubspot possess these classic marketing automation features.

3.6.2 API requirements

Both systems have all that is needed to fulfil the requirements of the extension. Both have a REST API that includes the functionality needed such as lead creation, form creation and updating, as well as filtering leads by segments. Both APIs are secured using some form of OAuth, covering the security requirements as well.

3.6.3 Costs

Costs of the systems have to be seen in a broader way than just the numbers. TYPO3 is used by a lot of agencies. Most of these agencies fall into a category that one would call in German 'der Mittelstand'. Unfortunately the English language lacks a proper translation for this word. In short, these are small and medium

sized, privately owned companies with annual revenues up to 50 million Euro and up to 499 employees. (Süß 2018) For these companies (especially those on the lower, or freelancer spectrum), seven hundred Euro per marketing automation solution per customer is quite a lot of money. A Mautic installation is entirely free, and can be customized to almost any decree. Assuming that TYPO3 and Mautic run on the same platform, it is safe to say that any agency able to set up a TYPO3 installation is also able to set up a Mautic installation. Mautic is the clear winner in this regard, as it falls under the same licensing conditions as TYPO3 (namely the GNU General Public License).

3.6.4 Opinions from stakeholders

In the CMS space, we have been very successful by using and supporting Open Source technology that has the power to disrupt the previous world of commercial vendors. For marketing automation software, we have identified a very similar situation and potential today, which lead us to the strategic decision to invest in building Mautic expertise, push our Mautic service offerings, and support the global and local success of this Open Source project as strongly as we can. So far, our TYPO3 clients - mainly B2B companies - find Mautic highly attractive and absolutely suitable for their needs. And to us, that is only the beginning.

Ekkehard Gümbel (Bitmotion GmbH)

3.6.5 Software philosophy

In general, the philosophy around TYPO3 is a strong one centred around open source software. The two software solutions chosen for marketing automation are opposite of one another in this regard. Hubspot is a strictly controlled environment which is only available as a paid service. Mautic on the other hand is a completely open solution. Its code is freely and openly available on the internet. Next to that, it is open source just like TYPO3 is. TYPO3 and Mautic also share the same license. Communitywise Mautic is the winner in this regard, as its philosophy stands miles closer to the philosophy of TYPO3 than Hubspot.

3.6.6 Security

Both systems use an OAuth implementation to make sure requests are secured and authenticated. However, Hubspot uses OAuth2, while Mautic uses OAuth1a. OAuth2 is generally regarded as more user friendly as its tokens can be automatically refreshed. Also, OAuth2 is less complicated for the developer as it does not require signature generation and all the hassle that comes with it (think of normalization of request parameters on both sides). Both implementations are however secure. While OAuth2 is generally easier to implement and use, I would not state that any of both systems has a clear advantage when it comes to these two security implementations.

3.7 Concluding

Summing up the results of this analysis we can see that both systems work in a very similar way. The difference comes when talking about the factors of costs and data control. Mautic is free and self hosted, meaning that you as the user have full control over your data. Hubspot on the other hand is paid and a software-as-a-service solution. The data control part can also be an important factor for businesses in Europe. Having control of your own data puts the company in a safer stance regarding the GDPR, as data deletions can be known to truly be permanent.

These two factors in combination with the preferences expressed by the stakeholder companies, the decision was made to use Mautic as the software solution for this project.

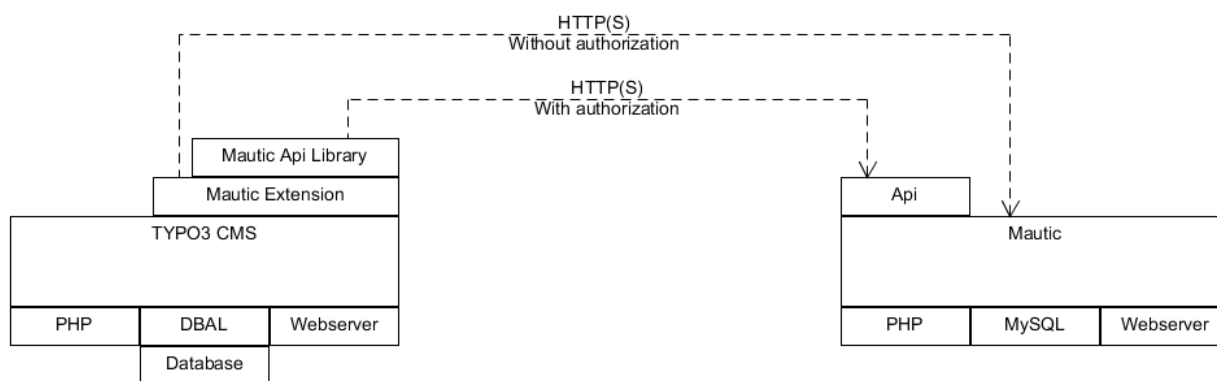
Chapter 4

Design

This chapter will contain a detailed explanation of the application design. How certain problems were tackled, the life of data structures within the code and the communication between API's. This chapter will be sectioned by feature.

Before continuing to the different extension functionalities, a short touch on the global architecture.

Figure 4.1: A simplified overview of system components



Both systems run on a PHP based server environment. There are two types of communication between TYPO3 and Mautic. The authorized kind, which runs through the Mautic API library to Mautic. These are requests for information. Mautic will only respond if the other side can successfully authorize. The second type is between the extension and Mautic directly. These are requests that do not need authorization, like reporting a page hit by a user.

4.1 Authorization and security

In order to safely connect to Mautic an authorization protocol must be used. Mautic supports three different protocols: BasicAuth, OAuth1a and OAuth2. For this application the decision fell to OAuth1a.

BasicAuth requires you to store the credentials of an actual user in the configuration of the TYPO3. We consider this not ideal as it blurs the line between API access and user access. We would like all credentials to have clear responsibilities and scopes, either user or API.

Both OAuth protocols allow us to create credentials only for the API. The main difference is that OAuth2 works with refresh tokens, meaning that every X days it will ask the Mautic server for new credentials, invalidating the old ones. OAuth1a has credentials that do not expire. OAuth1a credentials are used to generate

a signature from the request body. This signature is then included in the request header. Only the client and the server know how this signature is generated. This means that the client will generate the signature from the request it is about to send. Once the server receives the request it will also calculate the signature based on the request body. Then it will compare the signature it calculated with the one in the request header. If they match, the request is valid.

4.1.1 Handling credentials

Every class that needs to interact with the Mautic API will need access to the credentials. We have decided to use an object that contains all the credentials instead of giving the code direct access to the credentials themselves.

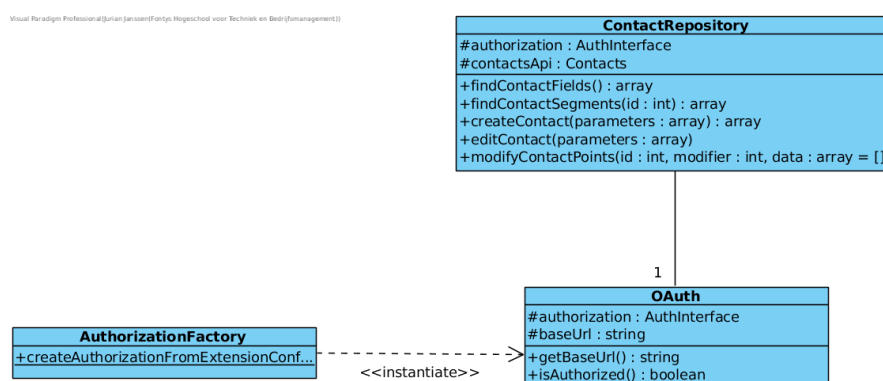
Every class shall be able to fetch such an object via a static factory. This way the object with the credentials can be used to make requests to Mautic without having to handle the actual credential strings. A developer shall be certain to have the right credentials by calling the factory and obtaining an instance of this authorization object.

This object shall contain the following values that are necessary to initiate contact with the Mautic API.

- Mautic installation basic url (entered by user)
- Mautic OAuth1 public key (entered by user)
- Mautic OAuth1 secret key (entered by user)
- Mautic OAuth1 access token (generated by system)
- Mautic OAuth1 access token secret (generated by system)

All current classes that need the credentials receive them via a dependency injection with the Authorization object.

Figure 4.2: A class implementation of a dependency injection between the authorization object and a repository class



4.1.2 Handling errors and warnings

If during any step of the authorization process an error or a situation that requires a warning occurs, the user must in some way be notified. TYPO3 has a system called flash-messages. These messages are triggered in

the backend and can have different contexts, such as warning, error, info and success. We have decided to use them in the following way.

Warnings are used to inform the user of non-critical, but serious issues. These issues do not break the functionality of the extension, but should be looked at by the user. A warning is thrown in the following situations.

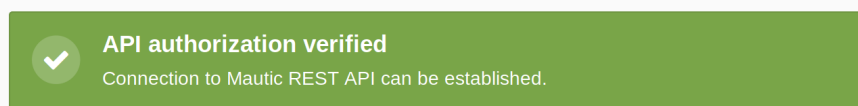
- Authorization was successful, but the connection is not encrypted with an SSL certificate (https)

Errors are used when a situations occurs that impedes the functionality of the extension. Errors must be resolved for the extension to work. Errors are thrown in the following situations.

- The authorization credentials are incorrect
- The Mautic installation is not reachable
- The Mautic installation is reachable and the credentials are correct, but the Mautic installation does not satisfy the minimum version requirements set by the extension

Success messages are thrown when everything is working as intended. If there are no issues, a success message is thrown informing the user everything is ok.

Figure 4.3: A flash-message example



4.2 Tracking

Tracking users on your page is done by including a small piece of JavaScript (this script is generated by Mautic) in your front-end. There are two ways this could be handled. We could simply include it by default when the extension is active, but we have opted to make the whole script configurable as this will allow more flexibility for the more complex use cases people might have.

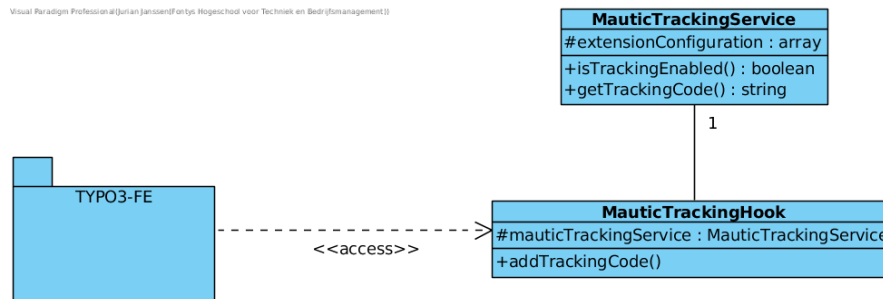
The tracking script can also pass custom parameters to Mautic, of course we cannot account for all of them. Therefore we have decided to give the user the option to override the default tracking script with their own custom version if they so desire. There shall then be three states of the script.

- Tracking script is disabled
- Tracking script is enabled, default script is used
- Tracking script is enabled, default script is overridden by custom script

The tracking script shall be passed to the front-end of TYPO3 via a service class. This service class will have the responsibility of providing the correct script (or none in case it is disabled). Since the tracking script needs the base url of the Mautic installation, it will obtain this directly via the extension configuration. One could argue that the base url could be gotten from an authorization object via dependency injection. However, since this object gets its base url from the extension configuration it is unnecessary to call the same logic twice.

If tracking is disabled it will return an empty string. If tracking is enabled it will generate the default tracking script with the base url obtained from the extension configuration. If tracking is enabled and a custom script has been set it will return the custom script.

Figure 4.4: A class implementation of a the tracking service



4.3 Dynamic Content

In essence, dynamic content is the opposite of static content. Meaning that we want to serve content on our website based on who is visiting. i.e. two people visiting the same page might see different content based on the information present about them.

Mautic can divide users into groups (called segments). These segments are populated based on various decisions configured in Mautic. The actions of a person on the webpage, how they interact with email, where they are from etc. Essentially, Mautic is used to funnel users into smaller, meaningful groups.

These groups are interesting, because they can be configured in such a way that they are meaningful in relation to the content on a website. A simple example would be if one were to sell three types of cars, blue ones, red ones and green ones. Mautic is configured in such a way that it will funnel these users into segments according to their interests. With this information we can now modify the content on our website to show information about blue cars to those interested in blue cars, and information about red cars to those interested in red cars. This example is quite the simplification, but one should see that this can be applied to a much grander scale in which tens or maybe hundreds of different groups of users will see content personalized for them specifically.

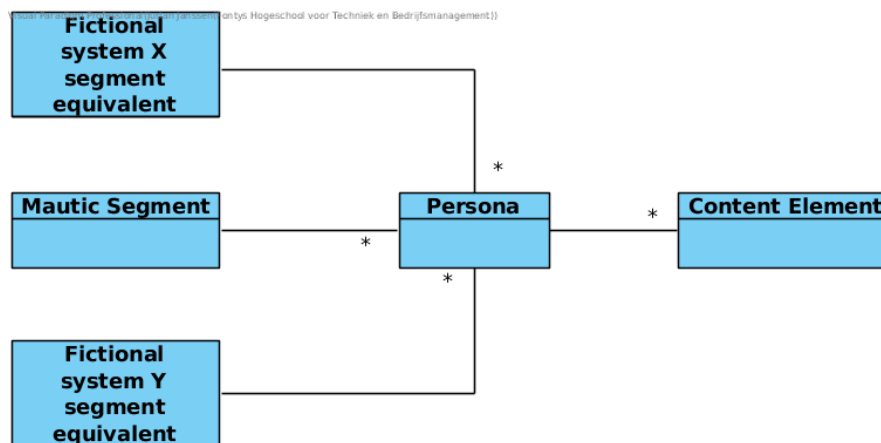
Implementations for other content management systems already exist, like Wordpress (*Mautic Wordpress plugin* 2018) and Drupal (*Mautic Drupal module* 2018). However, they all have one thing in common. They fetch the dynamic content elements from Mautic instead of having their own content elements be dynamic. We have decided to go a different route. We want TYPO3 to do all the content (it is a content management system after all), and then Mautic will decide which content elements TYPO3 will show the user. This way, we combine the content flexibility of TYPO3 with the brains of Mautic without compromising the integrity of our content management functionality.

4.3.1 Personas

The first idea was to directly link the Mautic segments to the content elements in TYPO3 and then show or hide them based on if a user is part of this particular segment or not. However, we decided to put an extra layer of abstraction in place called 'personas'. Personas are objects in TYPO3 that contain one or multiple Mautic segments, and then we show or hide content based on the persona of a user.

This abstraction was chosen for two main reasons. The first being that it gives the user in TYPO3 more

Figure 4.5: A simplified visualization of personas



Forms are an integral part of marketing automation, as they can be used to collect data about leads. Data filled out in a form can be used by Mautic to identify leads. For this reason we want this connection between TYPO3 forms and Mautic forms to be a main feature of the application.

Both TYPO3 and Mautic have their own implementation of forms. While the concept of a form is the same in both systems, their data structures vary widely. In order for both systems to interface with one another the structure of a TYPO3 form must be translated to a structure Mautic can understand.

Mautic will assign an id to the form, as well as all form fields. These need to be saved in TYPO3 as well, as these parameters are required when one wants to edit the form later. The id's are unique values that make sure Mautic knows which form or field you are trying to edit.

Next to that Mautic will return a so called 'alias' per form field. We need to save those as well as they are crucial in the process of submitting the form. A form submit to Mautic must always be in the structure of alias/value. See the example below.

```
1 array(  
2     'firstname_field_alias' => 'Jurian',  
3     'lastname_field_alias'  => 'Janssen',  
4     'email_field_alias'     => 'jurian.janssen@example.com',  
5 );
```

This brings us to the following list of data that we need to save in TYPO3.

- Id of the form returned by Mautic
- Id of each field returned by Mautic
- Alias of each field returned by Mautic

Lastly, TYPO3 and Mautic have different form field types. For instance, a field that is of the type 'RadioButtons' in TYPO3 would be of the type 'radio' in Mautic. These two need to be matched correctly, and this needs to be done for every form field type.

Since the form editor in TYPO3 is highly extendable, the question arises 'what happens to custom field types?'. This extendibility is something that needs to be taken into account in this process. Not only do we want all TYPO3 vanilla fields to work with Mautic, but we also want to give other developers the option to use their custom field types with the extension. We have labeled this concept 'form transformation', and the next section will explain its structure thoroughly.

4.4.2 Transformation

Form transformation is a concept in the extension that makes sure TYPO3 forms are correctly translated to Mautic forms, and crucial Mautic form data is in turn returned to TYPO3 and saved.

We want this concept to transform all vanilla TYPO3 field types out of the box, but we also want to make sure developers can extend this concept with their own field types. We achieve this by dividing the entire form into smaller objects. We handle the form object as a form, that should be a given. But at this point we decouple all the form fields from the form object. Each field type gets their own 'transformation class'. This class will handle the transformation of the TYPO3 structure to the Mautic structure for that specific form field type. A text field of course has to be handled differently than a radio button field. These classes all extend a class called 'AbstractFormFieldTransformation'. Each specific class will contain the logic needed to transform that specific field type to a Mautic structure. There are a couple of advantages to this approach. First, the responsibility of each class is clearly defined. We know exactly which fields get touched by which class, making it easy to find your way through the code. Second, this allows other developers to extend the application with their own field types. All they have to do is create a new transformation class for their specific field type, implement the logic, and register it with the Mautic extension. The extension does not care what your class does, as long as it is an instance of 'AbstractFormFieldTransformation'.

4.4.3 Mapping lead properties

Not only do we want to submit result data from a form to Mautic, we also want Mautic to know what kind of data is submitted in a field. For instance, if we have a form field where the user should enter their email address, we want Mautic to know that this field contains the email address of a lead.

Mautic has an API endpoint that allows us to fetch all known lead fields (these are basically all attributes of a lead. Name, email, address, etc.). By the means of a ViewHelper we can then add an extra dropdown field to the form editor in which the user can select a lead property that this specific form field represents. So, if there is an email field in TYPO3, the user can select in this dropdown that this field contains the Mautic property 'Lead Email'. This property is then sent along to Mautic together with the form structure (as discussed earlier in the transformation section). At this point Mautic knows what kind of data this field contains and can do its lead magic accordingly.

Figure 4.6: A form field in the Mautic UI that is matched to 'lead email'

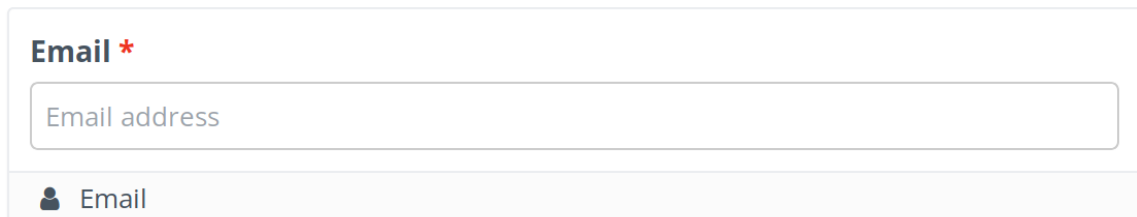
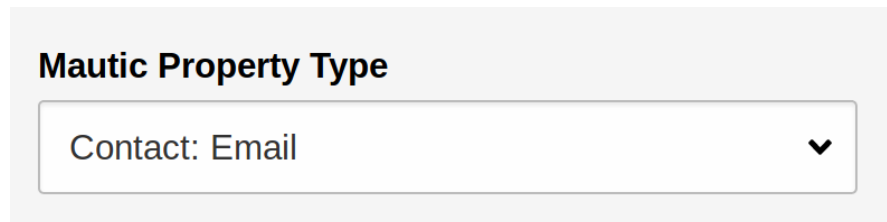


Figure 4.7: A form field in the TYPO3 UI that is matched to 'lead email'



By creating this translation of lead properties, we add a powerful ability to the form integration by allowing leads to be identified directly over TYPO3 forms without having to map the fields manually in Mautic.

4.4.4 Data storage

The extension stores form data in conjunction with the TYPO3 forms extension. This means that it uses the storage protocols of the forms extension. All forms created with the Mautic extension will be saved in a YAML format. (*TYPO3 Forms Documentation* 2018)

4.5 Installation

There are two ways to install an extension in TYPO3. The first one is to upload the extension in the TYPO3 backend via the extension manager. This process is very simple and straight-forward. One simply uploads a zip archive of the extension, and the system will take care of the rest.

The second method is to install the extension via Composer, the dependency management tool for PHP projects.

4.5.1 Extension manager

The TYPO3 extension manager cannot resolve dependencies on its own. This means that dependencies need to be included in the package. This will be solved by having a 'Libraries' folder in the extension that contains all the dependencies pre-installed.

4.5.2 Composer

In order for the extension to work within a Composer environment two things need to be done. First the extension needs to come with a 'composer.json' file containing all the dependencies (along with other configuration, however not relevant in this section) as well as a connection between the repository of the extension (on GitHub in this case) and Packagist (the service Composer installs packages from). This will be done via a webhook from GitHub to Packagist.

The dependencies in Composer will be defined as follows:

```
1  "require": {  
2      "typo3/cms-core": "^8.7",  
3      "bitmotion/marketing-automation": "^1.0.0",  
4      "mautic/api-library": "^2.14.2"  
5  },  
6  "autoload": {  
7      "psr-4": {  
8          "Bitmotion\\Mautic\\": "Classes/"  
9      }  
10 },
```

Chapter 5

Implementation

This chapter will contain details about the implementation of the software solution. The concepts described in earlier chapters will be laid out in detail, with code examples wherever relevant.

5.1 Authorization

The authorization process begins in the extension manager of TYPO3. The user will be prompted with a form. The user must provide Mautic authorization credentials in this form.

Figure 5.1: Authorization form in the TYPO3 extension manager

Base URL of the Mautic instance
authorization.baseUrl (string)

Public Key from Mautic API Credentials
authorization.publicKey (string)

Secret Key from Mautic API Credentials
authorization.secretKey (string)

Once the user fills in the required fields, and the 'save' button is pressed, the data will be checked and processed by the 'Authorize' class, defined under the 'UserFunctions' namespace. It is defined as such, because it post-processes the actions of a user, in this case saving the credentials. The following code is executed.

```
1 if (empty($this->extensionConfiguration['baseUrl']))
2     || empty($this->extensionConfiguration['publicKey'])
3     || empty($this->extensionConfiguration['secretKey'])
4 ) {
5     return $this->showCredentialsInformation();
6 }
7
8 if (substr($this->extensionConfiguration['baseUrl'], -1) === '/') {
9     $this->extensionConfiguration['baseUrl'] = rtrim($this->
10         extensionConfiguration['baseUrl'], '/');
11     $this->saveConfiguration();
12 }
13 if (empty($this->extensionConfiguration['accessToken']))
```



```

14     || empty($this->extensionConfiguration['accessTokenSecret'])
15 ) {
16     return $this->showAuthorizeButton();
17 }

```

The first statement checks if all required configuration elements are present. If one or more are absent, an error message is returned to the user telling them that crucial information is missing. This message is handled in the 'showCredentialsInformation()' method.

The second statement checks if the base URL of the Mautic installation given by the user ends with a forward slash. If it does, the forward slash at the end is stripped from the URL. This is done to maintain consistency within the application. All URLs will not have a closing forward slash.

Next it will check if the access tokens are set. Access tokens are variables set by the system after the authorization process has been completed. If they are not present, a button will be shown to the user which will allow them to fetch the access tokens from Mautic. If the access tokens are present this action is not needed, and thus the button stays hidden.

We shall assume a first install, in which the access tokens have not been set yet. The authorize with Mautic button is shown. The user clicks this button and is forwarded to Mautic. This request will include the earlier provided information. The user is now prompted to log into Mautic and give access to the TYPO3 application. Once the user does this, he or she is redirected back into the TYPO3 backend. This redirect request will also contain the access tokens, and they will now be saved.

At this point all the information is present to try and initiate contact with the Mautic API. This is also immediately done to ensure everything is working as intended. The code below shows the process.

```

1 $api = new MauticApi();
2 $api = $api->newApi('contacts', $this->authorization, $this->authorization->
    getBaseUrl());
3 $api->getList('', 0, 1);
4 $version = $api->getMauticVersion();
5 if ($version === null) {
6     return $this->showErrorMessage();
7 }
8 if (version_compare($version, $this->minimumMauticVersion, '<')) {
9     return $this->showIncorrectVersionInformation($version);
10 }
11 if (0 === strpos($this->extensionConfiguration['baseUrl'], 'http:')) {
12     return $this->showUnsecureConnectionInformation();
13 }

```

A new MauticApi object is created, and it will execute a simple request. Every request sent to Mautic should also return the version of Mautic. If the version is empty, or not present we know something went wrong. In this case an error is presented to the user telling them that the API of Mautic is not working as expected.

If it is working, the version will be checked against an internal parameter called 'minimumMauticVersion'. The extension depends on certain functionality that has not always been present in Mautic, so we have to set a minimum version. In case the version requirements are not met, the user is presented with an error telling them to upgrade their Mautic installation to the latest version.

Lastly, if there were no critical errors, the system will check if the connection between Mautic and TYPO3 is secure (using SSL, HTTPS). If it is not, a warning will be presented to the user informing them that everything is working as expected, but highly recommending them to secure the connection. This however, does not impede functionality.

If all of the previous checks passed without issue, the user is presented with a green message informing them that the connection to the API of Mautic was successful.

```
1 return $this->showSuccessMessage();
```

The authorization process has now been completed, and all functionality of the extension should be available to the user at this point.

5.2 Tracking

Tracking can be enabled in the extension manager (this is in a separate tab). The user has two options here. The first one is a checkbox to enable tracking on the website. If this checkbox is checked, the default Mautic tracking script (JavaScript) will be injected on every page on the TYPO3 frontend. The second option given here is optional. It gives the user the option to override the default tracking script. There are use cases for this. By default the tracking script only passes a select list of parameters to Mautic, if the user wants to pass additional data, he or she can do so by overriding the script with this option.

Figure 5.2: Tracking form in the TYPO3 extension manager

Below you can see the default tracking script as provided by Mautic (JavaScript).

```
1 <script>
2 (function(w,d,t,u,n,a,m){w['MauticTrackingObject']=n;
3   w[n]=w[n]||function(){(w[n].q=w[n].q||[]).push(arguments)},a=d.
4     createElement(t),
5     m=d.getElementsByTagName(t)[0];a.async=1;a.src=u;m.parentNode.insertBefore(
6       a,m)
7   })(window,document,'script','https://mautic.example.com/mtc.js','mt');
8   mt('send','pageview');
```

The method that renders the tracking script is registered with TYPO3 in the configuration files of the extension as seen below.

```
1 $GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['tslib/class.tslib_fe.php']['
2   configArrayPostProc']['mautic'] =
3   \Bitmotion\Mautic\Hooks\MauticTrackingHook::class . '->addTrackingCode';
```

The method it calls is defined in the 'MauticTrackingHook' class. This class will then fetch the tracking script from the 'MauticTrackingService' class. Responsibilities have been separated in this case as the hook should only have the responsibility of listening to the system and responding accordingly. The service class holds the responsibility of giving out the actual script. This is done to make sure other classes can have access to the script without going into the hook.

```

1 public function addTrackingCode()
2 {
3     if ($this->mauticTrackingService->isTrackingEnabled()) {
4         $pageRenderer = GeneralUtility::makeInstance(PageRenderer::class);
5         $pageRenderer->addJsInlineCode('Mautic', $this->mauticTrackingService->
            getTrackingCode());
6     }
7 }

```

If tracking is disabled in the extension configuration, no action is taken. However if it is enabled the tracking code will be fetched from the 'MauticTrackingService' class.

This class will check if the tracking script override configuration option has been set. If it is set, it will return the value provided there. If it has not been set, the default tracking script will be returned. The URL of the Mautic installation will be injected into the default script before it is returned.

```

1 if (!empty($this->extensionConfiguration['trackingScriptOverride'])) {
2     return $this->extensionConfiguration['trackingScriptOverride'];
3 }
4
5 return '(function(w,d,t,u,n,a,m){w[\'MauticTrackingObject\']=n;\'
6     . \'w[n]=w[n]||function(){(w[n].q=w[n].q||[]).push(arguments)},a=d.
        createElement(t),m=d.getElementsByTagName(t)[0];\'
7     . \'a.async=1;a.src=u;m.parentNode.insertBefore(a,m)})(window,document,\'
        script\','
8     . GeneralUtility::quoteJSvalue($this->extensionConfiguration['baseUrl'] . \'
        /mtc.js\')
9     . \',\'mt\');mt(\\'send\',' . \'pageview\');';

```

This whole implementation creates a structure with a clean division of responsibilities among classes while preserving all the checks needed. We allow the frontend to fetch the tracking code via a hook, and we allow other classes to access the tracking code via the service class. The service class also contains the logic needed to determine what tracking code to return, if any at all.

5.3 Forms

The form feature of the extension revolves around synchronizing form structure between TYPO3 and Mautic. This means that a form created in TYPO3 should be created in Mautic as well in the same structure.

5.3.1 Form synchronization

This implementation starts with listening for form actions in TYPO3. This can be done via hooks. There are various hooks that one can use during the process of form creation, editing and deleting. These have names like 'beforeFormSave' or 'beforeFormDelete'. The implementation of hooking into them can be seen below.

```

1 call_user_func(function () {
2     // Assign the hooks for pushing newly created and edited forms to Mautic
3     $GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['ext/form']['beforeFormDuplicate',
        ][1489959059] =
4     \Bitmotion\Mautic\Hooks\MauticFormHook::class;
5 }

```

```

6     $GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['ext/form']['beforeFormDelete'
      ][1489959059] =
7     \Bitmotion\Mautic\Hooks\MauticFormHook::class;
8
9     $GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['ext/form']['beforeFormSave'
      ][1489959059] =
10    \Bitmotion\Mautic\Hooks\MauticFormHook::class;
11 });

```

These hooks will call methods by the same name in the 'MauticFormHook' class. This means that it is now possible to call a method before forms are saved, duplicated or deleted. (Note that there is also a 'beforeFormCreate', but we do not need this one as 'beforeFormSave' is called on the first save anyway)

Upon any of these user actions, we now have a function that will receive the entire structure of the form. This function can now execute logic, and in the end return the form structure to the system. One can see it as a middle-man between the creation of the form and the saving of the form.

Forms all have a variable called 'prototype'. The prototype can be seen as an ancestor configuration. The default prototype is 'standard'. We have decided to add our own prototype called 'mautic'. By doing this we can create a clear differentiation between standard forms and forms meant to be synchronized with Mautic.

The first thing the hook does is check if this prototype is set, and if it is set to 'mautic'. If it is not set to 'mautic', we do not need to take any action on the form, as it isn't a Mautic form. We check if the hook 'is responsible' for this particular form. If it isn't, it will return and stop carrying out any Mautic logic.

```

1 // Form is not a Mautic form
2 if (!$this->isResponsible($formDefinition)) {
3     return $formDefinition;
4 }

```

If it is responsible, it will continue on to apply 'transformation' logic to the form. This transformation concept is translating the TYPO3 form definition into a Mautic form definition, so it can be sent to the Mautic API.

```

1 $this->transformForm($formDefinition);
2 $this->transformFormElements();

```

First it will transform the root element (the form) itself. We separate handling the form and the fields from one another. This is done because a form is a trivial data structure, but fields can be of different types (text fields, radio buttons, checkbox, etc.) and thus need different transformation logic per type.

Each entity has a class that holds responsibility for transforming objects of that particular type. There is one for form, text field, radio buttons, etc. Each one of these is registered in the configuration like so.

```

1 $GLOBALS['TYPO3_CONF_VARS']['EXTCONF']['mautic']['transformation']['formField'
     ]['Checkbox'] = \Bitmotion\Mautic\Transformation\FormField\
      CheckboxTransformation::class;

```

The reason this is done in the configuration, and not simply in a method that matches them up is for extendibility reasons. Another developer might have a custom form field type implemented, but the Mautic extension does not support the transformation element for this field type out of the box. By registering these transformation classes in the configuration other developers can build an extension on top of the Mautic extension and register their own transformation classes. This makes the extension extremely flexible for other developers.

A simple transformation for a text field looks like this. Note that I am giving an example here of text field, every field type has such an implementation. To see those please look at the GitHub repository.

```

1 $fieldData = [
2     'label' => (!empty($this->fieldDefinition['label'])) ? $this->
        fieldDefinition['label'] : $this->fieldDefinition['identifier'],
3     'alias' => str_replace('-', '_', $this->fieldDefinition['identifier']),
4     'type' => $this->type,
5 ];

```

Above you see the Mautic field properties being created for a text field. The 'label' is set to the field's 'label' in TYPO3 (by coincidence, these have the same name). Next the 'alias' in Mautic is set to be the 'identifier' in TYPO3. Lastly, we need to set the 'type', in this case this is a member of the transformation class. Every transformation class has a member called 'type'. This member contains the field type in Mautic, in this case the value will be 'text'.

In the case of form fields with lists, like radio buttons or dropdowns, the logic becomes more complex. However, no matter how complex the transformation logic, as long as the transformation class for that field type is present, syncing with Mautic will work. I could go into detail about how more complex elements are transformed, but the concept is the same, just with more complex logic. It can be found on the GitHub repository.

Now that the form has been completely transformed, it needs to be passed to the Mautic API. The responsibility of communicating with the Mautic API lies in a different class, the 'FormRepository'. This class can execute trivial form actions such as 'create', 'update', 'delete', 'get' and 'submit'. Again this logic was moved to a different class in order to decouple these methods and make them available to other classes as well as centralize all methods that perform actions 'on a form'.

When a new form has been transformed, the 'createForm' method in the repository will be called. This method will do the 'form create' API call to Mautic, and gather the authorization credentials needed to make such a request.

At this point Mautic will process the request and return a JSON object. This JSON object is the result of the logic it carried out. This response needs to be processed as Mautic will assign id's to the form and the fields. We need these id's for future update calls on that same form. The id is the unique identifier of the form (or form field). During an edit action in Mautic all forms and fields that do not contain an id are treated as new elements. This is of course undesired behaviour for the extension. Because of this we need to save the id's in our own form, so that we can later reuse them when we update the form.

This is a tedious process, as we will need to loop through both the TYPO3 form structure as well as the Mautic form structure, which are both nested arrays. The code would become barely readable when pasted in this document. Therefore I suggest you look at the 'updateFormDefinition()' method in the 'AbstractFormTransformation' class.

Once this process has been completed the TYPO3 form structure is passed back to the TYPO3 system (with all the Mautic parameters set). TYPO3 will then handle the rest of the form logic, including saving it to a YAML file.

5.3.2 Form finishers

Finishers are a TYPO3 concept in regards to submitting forms. A Finisher is a class containing logic that will be executed when someone submits a form. These finishers usually carry out one specific action. Users are able to add or remove finishers from forms in the backend of TYPO3.

This extension adds four new finishers. They all extend the 'AbstractFinisher' class that is given by TYPO3.

As long as your own finisher extends this class, TYPO3 will carry out the logic defined within it. The abstract class defines that a method must be implemented called 'executeInternal()'. This method has access to all the form data and should execute the logic to that specific finisher.

The extension adds the following finishers.

- MauticCompanyFinisher: Creates a new company in Mautic from the submitted form data
- MauticContactFinisher: Creates a new contact (lead) in Mautic from the submitted form data
- MauticFinisher: Posts the results of a TYPO3 form to the corresponding form in Mautic
- MauticPointsFinisher: Increases or decreases the submitters points in Mautic by a set amount

Below is an example of the executeInternal() method in the MauticPointsFinisher.

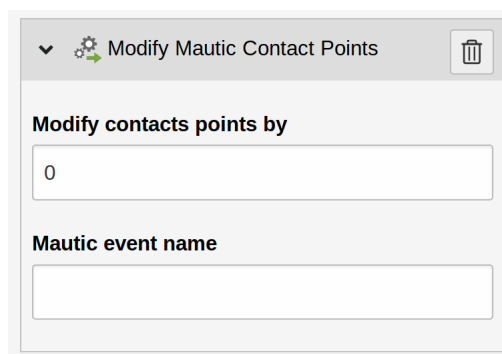
```

1 protected function executeInternal()
2 {
3     $pointsModifier = (int)($this->parseOption('mauticPointsModifier') ?? 0);
4
5     if (0 === $this->mauticId || 0 === $pointsModifier) {
6         return;
7     }
8
9     $data = [];
10    $data['eventName'] = $this->parseOption('mauticEventName') ?? '';
11
12    $this->contactRepository->modifyContactPoints($this->mauticId,
13        $pointsModifier, $data);
14 }

```

It gets the parameters it needs from user input (the data the user has set for the finisher in the backend). This is done via '\$this->parseOption()'. Then, the given data is handed over to the repository class responsible for contacts. The repository in its turn will talk to the Mautic API.

Figure 5.3: MauticPointsFinisher configuration in the backend



▼ ⚙️ Modify Mautic Contact Points

Modify contacts points by

0

Mautic event name

5.3.3 Submitting forms

As described in the previous section, submitting forms is done via a finisher. However, in contrast to all the other finishers, the finisher that submits the actual form data to Mautic has an extra step added in. I would like to elaborate on this step here.

Mautic sets four different cookie values. (*Contact Monitoring* 2018) These are used to track the user around the website. When submitting a form, these cookie values need to be passed along with the request. Otherwise Mautic will not link the submission to the correct lead/contact.

To handle this, submitting a form to Mautic was moved to a separate class called 'MauticSendFormService'. This service class will handle adding the correct meta-data to the request (cookies and the user IP in this case). The relevant part of this logic can be seen below.

```

1  $client = new Client();
2  $multipart = [];
3  $headers = $this->makeHeaders();
4  $cookies = $this->makeCookies();
5
6  try {
7      $result = $client->request('POST', $url, [
8          'cookies' => $cookies,
9          'headers' => $headers,
10         'multipart' => $multipart,
11     ]);
12 } catch (\Exception $e) {
13     /** @var ResponseInterface $result */
14     $result = $e->getResponse();
15     if ($result !== null) {
16         return (int)$result->getStatusCode();
17     }
18
19     return 500;
20 }

```

5.4 Dynamic Content

Dynamic content is in essence showing or hiding content based on certain conditions. In this extension the conditions are based on in which Mautic segments the user is located. With this implementation we can show and hide content based on the Mautic segments of the user.

5.4.1 Fetching Mautic segments

Since the dynamicity of content is based on Mautic segments, those will need to be fetched first in order to use them.

This job is carried out by a repository class called 'SegmentRepository'. This class talks to both the Mautic API as well as the local TYPO3 database. We do not want to fetch segments every time a content element is edited, that would cause a lot of HTTP requests, resulting in performance issues in the backend of TYPO3. Therefore we store segments in a database table.

Segments are fetched once, after the authorization process has been completed. After that the user has the option to fetch them again via a button in the backend of TYPO3. Users can create, remove or edit segments in Mautic, and therefore we need this manual fetch button, in order for them to fetch any updates on demand.

The SegmentRepository will connect to the database and fetch all segments it currently knows. When this query has been executed, it will ask the Mautic API for a list of segments. At this point it will see which segments it already knows (based on id, which is the unique identifier). It will update those it already knows,

create the ones it does not know, and delete the ones that it knows, but didn't receive from Mautic anymore (note that this is a soft-delete process). The code below is a simplification of the process.

```

1 public function synchronizeSegments()
2 {
3     $queryBuilder = GeneralUtility::makeInstance(ConnectionPool::class)
4         ->getQueryBuilderForTable('tx_marketingautomation_segment');
5     $queryBuilder->getRestrictions()->removeAll();
6
7     $result = $queryBuilder->select('*')
8         ->from('tx_marketingautomation_segment')
9         ->execute();
10
11     $availableSegments = [];
12     while ($row = $result->fetch()) {
13         $availableSegments[$row['uid']] = $row;
14     }
15     $result->closeCursor();
16
17     $queryBuilder->update('tx_marketingautomation_segment')
18         ->set('deleted', 1)
19         ->execute();
20
21     $segments = $this->findAll();
22     foreach ($segments as $segment) {
23         // Do matching logic
24         // and write to db
25     }
26 }

```

Since this function is public, any class will be able to call this method if it needs an up-to-date list of segments. This also includes code from other developers. I can imagine other developers will at some point write code that depends on an updated list of segments, therefore this process can be initiated from anywhere in the code.

5.4.2 Mapping segments to personas

A persona is a proxy object between the marketing automation system and TYPO3. A persona is linked to one or multiple Mautic segments, and then this persona is used to hide or show content. So a hypothetical situation would be that a user has a content element called X. The user creates a persona called Y. In the backend this user configures persona Y to encompass Mautic segments A, B and C. Then the user sets content element X to only show for persona Y. This will then make sure that content element X is only shown to people in Mautic segments A, B and C.

The reasoning behind this structure is that other developers can create their own extensions that might talk to a different marketing automation system (that has a similar concept to segments in Mautic) and is able to link these to the persona object as well. This will allow the user to create dynamic content based on more than just one marketing automation system. While this is out of the scope of this project, it is still a good feature to have in regards to extendibility.

5.4.3 Serving dynamic content

In order to serve dynamic content, we need to know during rendertime of a page to which persona(s) the visiting user belongs. For this reason, the personas of a user are set in a cookie on their machine. This way the TYPO3 system always has access to this data when it starts rendering a page.

The lifetime of this cookie can be configured in the backend of TYPO3. When this cookie runs out, it will be refreshed on the next visit of the user. The user will still always have a cookie containing their lead id in Mautic. If the persona cookie has run out, the Mautic id is sent to Mautic with a request to send back all segments this user is a part of. These segments in time will be matched up against the personas defined in TYPO3, and then these will be written to the cookie again.

This is done in a cookie to prevent sending a request to Mautic for every page a user loads. The default cookie lifetime is set to sixty seconds. This means that after sixty seconds the personas of a user will be refreshed. This is a compromise between being always up to date and not overloading the server with too many requests.

As mentioned earlier, this cookie lifetime value can be tweaked. For websites with very high traffic it can be increased to reduce requests (and thus server load), and for testing it can be decreased to always get the latest results.

Below you can see the two methods responsible for this logic.

```

1 public function needsUpdate(Persona $currentPersona, Persona $newPersona): bool
2 {
3     $isValidMauticId = !empty($this->mauticId);
4     $isEmptyPersonaId = empty($currentPersona->getId());
5     $this->languageNeedsUpdate = $isValidMauticId && $currentPersona->
        getLanguage() !== $newPersona->getLanguage();
6
7     return $isValidMauticId && ($isEmptyPersonaId || $this->languageNeedsUpdate
        );
8 }
9
10 public function update(Persona $persona): Persona
11 {
12     $segments = $this->contactRepository->findContactSegments($this->mauticId);
13     $segmentIds = array_map(
14         function ($segment) {
15             return (int)$segment['id'];
16         },
17         $segments
18     );
19     $personaId = $this->personaRepository->findBySegments($segmentIds)['uid']
        ?? 0;
20
21     return $persona->withId($personaId);
22 }

```

At this point we know to which persona a user belongs (or the user belongs to none, which is also a valid case). Now the page can be rendered, as TYPO3 knows to which persona the user belongs and which content elements are specifically (or specifically not) for that persona.

Chapter 6

Conclusion

As this report comes to a closing, the reader should be aware of the concepts and implementation of this project.

Research has been done in terms of which marketing automation would fit best for a TYPO3 integration. The result of this research was that Mautic was the best suitor for this project.

As a result a design and implementation was made, connection TYPO3 to Mautic. This software solution as a whole is now able to serve dynamic content, synchronize forms between the two systems and track users across a TYPO3 website. As the report explains in detail, all the requirements have been met.

Various companies around the world are already using this software solution in production environments. At this point we are only looking forward to how we can extend it even more, and make it even better.

Various ideas still on the drawing board are adding more actions to forms to allow a more direct control. An example of this would be sending a user an email via Mautic upon filling in a form. Another feature to still be added is to automatically fill forms based on user data stored in Mautic.

Finally the extension will have to be adapted to TYPO3 version 9LTS. There are some breaking changes transitioning from version 8 to 9.

As a closing note I would like to say that this project has been a success. At the time of writing, over four hundred installations of the extension have already been carried out.

Chapter 7

Recommendations

This chapter provides the reader with the advice for future development and improvement of features.

7.1 Mautic API

I would recommend always have a good look at the documentation of the Mautic API and testing the endpoints accordingly. Some of them might not work entirely like expected. We also ran into an issue with the OAuth1 protocol that we eventually fixed ourselves. Fixing issues in the Mautic core by yourself is always recommended, as it will help the community as well.

7.2 Automatically filling forms

A feature that could still be developed is the auto filling of forms based on the data gathered on a specific person. This could be done using an API call during runtime and then match the returned contact fields from Mautic to the form field properties. This way auto filling fields with Mautic data is possible.

7.3 Creating more variations of finishers

There are still a lot of actions that could be implemented in a finisher. Some examples of these are adding a contact to a campaign (or removing them), as well as creating unsubscribe actions to unsubscribe a contact from various emails. To start with this, have a look at the finishers already present. The structure of such a class is very straight forward.

Self Reflection

Described here are the competences needed to complete this project along with a description as to how these competences are achieved.

	Manage	Analyze	Advice	Design	Implement	Professional Behaviour	Research Skills
User-Interface					3		
Business Processes		3					
Infrastructure							
Software		3	3	3	3		
Hardware Interfacing							
Professional Skills							

Table 7.1: Project Skills Level

User interface: implement

For the extension to be usable the TYPO3 interface needs to be extended with certain elements. These elements will allow the user to configure as well as use the extension in a straight forward way.

Examples of this are the storage of authorization credentials, but also the creation of dynamic content and marketing automation forms within the existing user interface of TYPO3.

Business Processes: analyze

In order to create a streamlined workflow between TYPO3 and the marketing automation solution, a clear understanding must be formed about how these marketing automation processes take place.

Examples of this are the collection of lead data and the distribution of form results to the correct endpoints.

Software: analyze

As part of the research two marketing automation software solutions are analyzed and compared to one another in order to find the most suitable candidate for the extension. See the research chapter for more details.

Software: advice

The advice part is a natural result of the research part, which will ultimately conclude to one system that will be implemented.

Software: design

The extension needs to be designed before it can be implemented. The flow of data, as well as class diagrams and sequence diagrams will be provided in order to paint a clear picture of the architecture behind the extension.

Software: implement

Lastly, the implementation of the extension will be the ultimate goal of this project. This, in the end, will result in a working project.

Bibliography

Anicas, M. (2018), 'An introduction to oauth 2'.

URL: <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>

Best Marketing Automation Software (2018).

URL: <https://www.g2crowd.com/categories/marketing-automation>

Contact Monitoring (2018).

URL: https://www.mautic.org/docs/en/contacts/contact_monitoring.html

Continuous Integration (2018).

URL: <https://www.thoughtworks.com/de/continuous-integration>

Find a TYPO3 partner (2018).

URL: <https://typo3.com/services/find-a-typo3-partner/official-typo3-partner-finder/>

Hampton, P. (2014), 'Dependency management in php using composer'.

URL: <https://blog.teamtreehouse.com/dependency-management-in-php-using-composer>

Koetsier, J. (2018), 'Fast-growing marketing automation still has only 3

URL: <https://venturebeat.com/2014/01/08/fast-growing-marketing-automation-still-has-only-3-penetration-in-non-tech-companies/>

Mautic Drupal module (2018).

URL: <https://github.com/mautic/mautic-drupal>

Mautic Wordpress plugin (2018).

URL: <https://github.com/mautic/mautic-wordpress>

OAuth: The Authorization Flow (2018).

URL: <https://oauth1.wp-api.org/docs/basics/Auth-Flow.html>

Oracle Eloqua: Drive Dynamic Journeys (2018).

URL: <https://www.oracle.com/marketingcloud/products/marketing-automation/>

PSR-2: Coding Style Guide (2018).

URL: <https://www.php-fig.org/psr/psr-2/>

Süß, D. (2018), 'Mittelstand - definitionen'.

URL: https://www.hk24.de/produktmarken/interessenvertretung/wirtschaftspolitik/mittelstandspolitik/mittelstand_definitionen/1145582

TYPO3 Forms Documentation (2018).

URL: <https://docs.typo3.org/typo3cms/extensions/form/>

Šuflaj, M. (2015), 'Publishing package to packagist'.

URL: <https://github.com/CurrencyCloud/currencycloud-php/wiki/Publishing-package-to-Packagist>

Appendices

Appendix A: Project plan

Marketing Automation with TYPO3

Bachelor Thesis Project Plan

Submitted by Jurian Janssen

In fulfilment of the requirements for the degree
Bachelor of Science
To be awarded by the
Fontys University of Applied Sciences

Düsseldorf, January 2, 2019

Information Page

Fontys Hogeschool Techniek en Logistiek
Postbus 141, 5900 AC Venlo

Project Plan for Bachelor Thesis Project

Name of student:	Jurian Janssen
Student number:	2573644
Course:	Informatics - Business Informatics
Period:	August 2018 - January 2019
Company name:	TYPO3 GmbH
Address:	Am Wehrhahn 41
Postcode, City:	40211, Düsseldorf
Country:	Germany
Company coach:	Mathias Schreiber
Email:	mathias.schreiber@typo3.com
University coach:	Stefan Sobek
Email:	s.sobek@fontys.nl
Examinator:	Geert Monsieur
External domain expert:	TBD
Non-disclosure agreement:	Yes (but not for the project itself)

Statement of Authenticity

I, the undersigned, hereby certify that I have compiled and written the attached document / piece of work and the underlying work without assistance from anyone except the specifically assigned academic supervisors and examiners. This work is solely my own, and I am solely responsible for the content, organization, and making of this document / piece of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document / piece of work will not be accepted for evaluation or for the award of academic credits if it is determined that it has not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document / piece of work and the underlying work will not be accepted for evaluation or for the award of academic credits if it is determined that it embodies plagiarism.

Name: Jurian Janssen
Student Number: 2573644
Place/Date: Düsseldorf, January 2, 2019

Signature:

Chapter 1

Introduction

This document will contains pre-defined set of goals and tasks surrounding the internship of the Author at the TYPO3 GmbH over the period of August 2018 till January 2019. This internship is done in the faculty of Business Informatics.

1.1 TYPO3 GmbH

The TYPO3 GmbH is a company located in the heart of Düsseldorf, Germany. First one must note that despite the name, the GmbH is not a TYPO3 agency. Its main purpose is to market and support TYPO3 as a product. In short this means that the GmbH does not do TYPO3 projects. What it does do however is offer extended support for versions that are no longer supported by the community (eLTS) and offer service level agreements to businesses using TYPO3.

As a second responsibility the GmbH promotes the TYPO3 product as a whole. This is done in numerous ways. Having a stand on events, giving talks on developer conferences and selling merchandise just to name a few. The general intent of this responsibility should be clear; expanding the TYPO3 world beyond the markets it already has. This also includes overseas markets in the America's, where TYPO3 is trying to gain a stronger foothold.

1.2 Team

The TYPO3 GmbH has 15 employees. Their fields of profession vary widely from development to account managing (PR) to finances to video editing. The company is lead by Mathias Schreiber, the CEO of the GmbH. Not all employees work in Düsseldorf. Some also work remotely. Either from home, or from an office space elsewhere.

Many other companies are involved with the GmbH. They are known as TYPO3 partners. Partners are companies that posses qualified expertise withing the TYPO3 project. In short, this means that the company has an X amount of TYPO3 certified developers, integrators, editors and consultants, as well as that the company has applied for partnership with the GmbH. Certifications can be obtained from the TYPO3 association (note that this entity is entirely separate from the GmbH) through the certification committee. To get certified an individual must pass a written exam.

1.3 Infrastructure and Organization

The GmbH uses multiple tools to assure a streamlined workflow. Peer-to-peer communication is done via Slack. Slack is a channel based communication tool that supports text, voice and video chat. For ticketing and development Jira and Bitbucket are used. Where Jira is the ticketing (issues) system and Bitbucket is the version control system. However, collaborative projects may differ in this regard and use other systems such as Git or Forge. This depends on the other parties involved.

Planning and organizing is done via Scrum. Every morning at 10:00 there is a daily scrum meeting on Slack. All employees are involved in this meeting, this means also the non-developers. During this meeting each person will present what they have done the day before and what they will do today. Short discussions are allowed, but are usually done after the 'daily'. Next to that there is a conference room where more general issues are discussed if it need be.

Chapter 2

The Project

The project in general is to create a well thought out connection between the TYPO3 content management system and a (or multiple) marketing automation solution(s).

This chapter will list certain aspects of the project such as requirements, tools, planning etc. More in depth explanation about these various topics can be found in Chapter 3 of the main report.

2.1 Stakeholders

TYPO3 GmbH: Has employees working on the project (in this case the author). Will also benefit from the resulting implementation, both in terms of development and marketing.

Bitmotion GmbH: A partner involved with the development of a marketing automation extension for TYPO3. The author will work together with various employees of Bitmotion GmbH in order to deliver the end product.

The author: Developer of the project in the name of TYPO3 GmbH. The integration is the end deliverable for the bachelor internship of the author.

Marketing Automation solution(s): The solutions that will be supported in the extension(s). A good integration between them and TYPO3 can also possibly increase their market share as they can then seamlessly expand their market to TYPO3 users/developers.

2.2 Workflow

The research concerning the different marketing automation systems will be conducted by the author internally. This document will be delivered in the form of a pdf and will also be part of the final thesis report.

Communication between the stakeholders will mostly take place on Slack. A private channel has been created in which one or multiple persons from each stakeholder are present. Company internal communication is either done via Slack, the daily scrum meeting or in person at the office during the day.

Development for this particular project is done via Github. This has multiple reasons. The first one being that this project will be entirely open source, meaning that in the end it will be free software that anyone can use and/or improve. The second reason for this is that not only the TYPO3 GmbH is involved, but also partners like Bitmotion GmbH. Github is a common ground where we can all work together in more or less the same

workflow. Github will be used as version control system as well as ticketing system (Github issues). These are naturally all accessible to the public at any time.

2.3 Risk Management

Various risks have been identified, analyzed, and where needed countered or suppressed. They are outlined below.

Description	The software does not function, or does not function as desired at the end of the project lifespan.
Occurrence chance	Low
Severity	Critical
Action	Employ quality management measures such as continuous integration solutions, code standard checkers and unit tests.

Description	Stakeholder Marketing Automation solution does not merge fixes to the Marketing Automation solution software made by the author in order for the extension (this product) to function properly.
Occurrence chance	Low
Severity	High
Action	Have steady lines of communication between the author and Marketing Automation solution developers.

Description	The TYPO3 version 9 LTS core release is postponed to a date outside of the lifetime of this project.
Occurrence chance	Low
Severity	High
Action	Ensure that the extension code can be backported to TYPO3 version 8 LTS with little to no effort.

Description	Loss of work due to hardware defects and/or damages caused by any factors outside the author's control.
Occurrence chance	Low
Severity	Critical
Action	Use a versioning system, and regularly pushing work to it.

2.4 Quality management

In order to maintain a certain quality standard for the endproduct a handful of tools are being used to guarantee the quality of the application code.

2.4.1 Testing

The entire application shall contain feature and unit tests. This may be extended to various protected methods if the situation of testing it is deemed reasonable. All tests are written using PHPUnit and the TYPO3 Testing Framework. To assure the quality of the product one hundred percent of the tests must pass without errors or warnings. The product will not be deemed finished until this requirement is met.

Testing code coverage is aimed at sixty percent or higher.

2.4.2 Code standards

All code within the application shall be checked against the 'PSR2' coding standards using php-cs-fixer. All PHP code within the application must comply with PSR2 standards in order to maintain consistency and code quality. The product will not be deemed finished until this requirement is met.

2.4.3 Code reviews

All pull requests to the main repository of the application shall be checked and approved by at least one other repository administrator before code can be merged.

2.4.4 Automated Testing

The application shall be continuously tested using Travis CI, a continuous integration solution. Every code impacting version control action (git push, git merge, pull request) shall be completely tested by Travis CI. Travis will run all aforementioned tests and check if the code standards of PSR2 are met. Merging will automatically be blocked by the version control system if the Travis CI build fails, and will remain blocked until the ongoing issues are resolved.

2.5 Deliverables

2.5.1 Project deliverables

Marketing Automation system analysis This deliverable is a document containing a feature analysis of various different marketing automation solutions.

TYP03 integration This deliverable is a software solution integration (a) marketing automation solution(s) in TYPO3. Written in PHP.

2.5.2 University deliverables

Project plan This document containing a rough outline of the project context and goals.

Mid-term report and presentation A report containing the progress up to that point of the project. This report will eventually be finalized into the thesis report. *Deadline: Oct 9th 2018*

Thesis report and presentation A report containing the progress up to that point of the project. This report will eventually be finalized into the thesis report. *Deadline: Jan 8th 2019*

2.6 Documentation and Coding standards

Documentation for the code will be provided in the doc headers of the code itself. Methods and variables will be explained in a standardized way. This standard is called ‘PSR2’. A library called php-cs-fixer (cs stands for Code Standards) will be used in order to ensure enforcement of this standard. A continuous integration solution called Travis CI will also, among other things, check the code style and will fail if the standards are not met. This essentially means that any changes to the code failing to abide the standards cannot be merged until changes are made to meet the standards.

2.7 Milestones (rough)

Here the rough milestones and their expected completion dates are presented.

First beta release marketing automation extension for TYPO3 A first working version that can be used by the public. Estimated completion date: November 2018.

Presentation marketing automation with TYPO3 Presentation at the TYPO3Camp Rhein-Ruhr. Estimated date: November 2018.

First official stable release of marketing automation extension for TYPO3 A first stable version of the extension is release. Estimated completion date: January 2018.

Gain industry partnerships with marketing automation software Industry partnerships are new to TYPO3. What it comes down to is that both parties promote each other by having an extension that has been marked by both parties as the the official TYPO3 extension for that system. Thought the extension(s) created we will try to gain more industry partners. Estimated completion date: January 2019.

2.8 Planning and phases

Phase 1: Research - Duration: 2 weeks - Aug 27th, Sept 9th (2018)

This phase is aimed at researching the different marketing automation solutions available to the author. Comparing their features and judging their suitability for a TYPO3 extension.

Phase 2: Application design and implementation - Duration: 2 months - Sept 10th, Nov 10th (2018)

This phase is aimed at designing and laying the foundation for the extension. This includes most of the programming work and actively discussing with collaborators and other parties about features and design choices.

Phase 3: User testing and beta release - Duration: 1 month - Nov 11th, Dec 20th (2018)

This phase is aimed at gathering test data and feedback from beta users in order to iron out any bugs, optimize the code, optimize the UI. Besides that this phase will also be used to promote the extension in order to gather a reasonable userbase. Lastly this phase will be used to write documentation.

Phase 4: Release - Duration: * - Jan 2nd, * (2019)

This phase will mark the first official release of the extension. The extension shall no longer reside in a beta state, but in a stable state, ready to be used in production environments. Minor adjustments might still be made (minor version releases), but the product is generally regarded as being in a completed state. This phase will also be used by the author to complete his thesis.

2.9 Tools

Various tools will be used throughout this project, below they are all listed together with their purpose.

- Composer - PHP dependency management system
- PHPStorm - An IDE for PHP projects
- Travis CI - Continuous integration solution
- Github - Version control system
- Github Issues - Ticketing system
- LaTeX - Document creation

2.10 Requirements

This section will touch on the requirements for the marketing automation extension. It will determine the needs (or assumed needs) of the customer and/or user.

Functional requirements

- The software shall function as an extension of TYPO3.
- The software shall be installable through Composer.
- The software shall be installable through the TYPO3 extension manager.
- The software shall be able to track leads across a TYPO3 installation.
- The software shall be able to share form data with TYPO3.
- The software shall be able to distinguish lead segments defined by the marketing automation solution.

- The software must connect securely between the two systems, using OAuth.
- The software shall strongly advise the user in case of missing security measures such as a non-SSL connection.

Non-functional requirements

- The code shall adhere to PSR-2 standards.
- The code shall be tested using the TYPO3 testing framework.
- The system functionality shall be documented.

Appendix B: Travis configuration

```

1 language: php
2
3 php:
4 - 7.1
5 - 7.2
6
7 env:
8 matrix:
9 - TYPO3_VERSION=~8.7
10
11 sudo: false
12
13 addons:
14 apt:
15 packages:
16 - parallel
17
18 cache:
19 directories:
20 - $HOME/.composer/cache
21
22 before_install:
23 - composer self-update
24 - composer --version
25
26 before_script:
27 - composer require typo3/cms=$TYPO3_VERSION
28 # Restore composer.json
29 - git checkout composer.json
30 - export TYPO3_PATH_WEB=$PWD/.Build/Web
31
32 script:
33 - >
34 echo;
35 echo "Running unit tests";
36 export typo3DatabaseName="typo3";
37 export typo3DatabaseHost="localhost";
38 export typo3DatabaseUsername="root";
39 export typo3DatabasePassword="";
40 .Build/bin/phpunit --colors -c .Build/vendor/typo3/cms/typo3/sysex/core/Build
    /UnitTests.xml Tests/Unit/
41 - >
42 echo;
43 echo "Running php lint";
44 find . -name \*.php ! -path "./*.Build/*" | parallel --gnu php -d display_errors
    =stderr -l {} > /dev/null \;
45 - >
46 echo;
47 find 'Tests/Functional' -wholename '*Test.php' | parallel --gnu 'echo;_echo_"
    Running_functional_test_suite_{}";_Build/bin/phpunit_--colors_-c_Build/
    vendor/typo3/cms/typo3/sysex/core/Build/FunctionalTests.xml_{}'

```

Appendix C: Narrowing down the list of marketing automation software

This article is meant to shed some light at the process used to narrow down the list of marketing automation software to just four contenders.

At first, the complete list that was taken into account can be found on g2crowd.com. This can also be found in the references under (*Best Marketing Automation Software* 2018).

Five criteria were defined to narrow down the list.

Features: This criteria is straight forward. Does the particular system have all the needed functionality to satisfy the requirements of our application.

Popularity: This criteria was chosen because the extension needs at least a decent user base. It makes no sense to build an extension for software nobody uses. This is measured by the amount of reviews on g2crowd and the positivity of those.

Costs: Given that the TYPO3 user base is very diverse. Ranging from freelancers to agencies all the way to enterprise, we must take into account that we build an extension for software that can be used by all of these groups.

Open/closed source: Sharing the same philosophy about open source software is a plus, as this can strengthen ties between TYPO3 and the open source marketing automation world. It also gives us the option to fix any bugs that we might find in the software that are crucial to the development of our extension.

Vendor reputation: This criteria was chosen to make sure we are dealing with software that does not die or disappear within the next years. A reliable and stable vendor should suffice in insuring this is not going to be the case.

The 'features' criteria is a must. Without it the extension cannot function as desired. The other criteria are variable. They are not critical, but definitely a plus.

These criteria have been assessed by looking at the documentation available for the various systems. They were then narrowed down to four systems, based on having a good mix of criteria. No system was found to satisfy all needs completely.

In the end we have one system widely regarded as the market leader, Hubspot. The most popular open source alternative, Mautic. A reputable vendor in the market leader sector, Salesforce. A reputable vendor in the contender sector, Oracle.

These systems will be assessed further and discussed with the stakeholders.

	Oracle Eloqua	SalesForce Pardot	Hubspot	Mautic
Features	Yes	Yes	Yes	Yes
Popularity	Mediocre	High	Very high	Mediocre
Costs	Relatively high	Relatively high	Relatively mediocre	None
Source	Closed	Closed	Closed	Open
Reputable vendor	Yes	Yes	Yes	Community