UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE

Magistrsko delo
(Master Thesis)

Uporabniški vmesnik za prevajanje v Drupal 8

(User Interface for translation in Drupal 8)

Ime in priimek: Saša Nikolić
Študijski program: Računalništvo in informatika, 2. stopnja
Mentor: doc. dr. Jernej Vičič
Somentor: doc. dr. Matthew Schwarzkopf

Koper, marec 2017

Ključna dokumentacijska informacija

Ime in PRIIMEK: Saša NIKOLIĆ

Naslov magistrskega dela: Uporabniški vmesnik za prevajanje v Drupal 8

Kraj: Koper

Leto: 2017

Število listov: 57 Število slik: 14 Število referenc: 31

Mentor: doc. dr. Jernej Vičič

Somentor: doc. dr. Matthew Schwarzkopf

UDK: 004.4'4(043.2)

Ključne besede: Drupal 8, Odprtokodni sistemi, Uporabniški vmesnik, Prevajanje

Izvleček: V delu se osredotočamo na uporabniški vmesnik za prevajanje spletnih vsebin z uporabo modula Translation Management Tool v odprtokodnem sistemu Drupal 8. Implementacija vključuje dve razširitvi spletnega urejevalnika besedil CKEditor ter tri glavna področja za prikaz podrobnosti o prevajani vsebini.

Key words documentation

Name and SURNAME: Saša NIKOLIĆ

Title of the master thesis: User Interface for translation in Drupal 8

Place: Koper

Year: 2017

Number of pages: 57 Number of figures: 14 Number of references: 31

Mentor: Assist. Prof. Jernej Vičič, PhD

Co-mentor: Assist. Prof. Matthew Schwarzkopf, PhD

UDK: 004.4'4(043.2)

Keywords: Drupal 8, Open Source, User Interface, Translation

Abstract: Thesis covers the User Interface for the translation of web content with the module Translation Management Tool in an open source content management system called Drupal 8. The implementation includes two plugins for web-based HTML text editor - CKEditor and three main areas for displaying more details about the content that is being translated.

Acknowledgement

I would like to thank my Google Summer of Code mentors Miro Dietiker and Sascha Grossenbacher for their constant support, engagement, and advice that aided me with this project. I would also like to thank MD Systems, as a leading contributor in open source, for initiating the Translation Management Tool module, and introducing me into Drupal. Furthermore I would like to thank my Google Summer of Code 2016 mentor Matthew Lechleider for the weekly guidance and support during my project, and my thesis mentor doc. dr. Jernej Vičič and co-mentor doc. dr. Matthew Schwarzkopf for helping me throughout the thesis.
My thanks and appreciations also go to Eduard Rene Claramunt for developing the Translation Memory for TMGMT and everyone in the Drupal community for all the sharing of their knowledge and the stimulating discussions that helped me overcome many obstacles.
Special thanks to my family for all the support throughout my studies.
The work was supported by the Google Summer of Code program with the focus on bringing more student developers into open source software development.

Contents

List of Figures

List of Abbreviations

HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
CGI	Common Gateway Interface
ASP	Active Server Pages
JSP	JavaServer Pages
CMS	Content Management System
TMGMT	Translation Management Tool
UI	User Interface
GPL	General Public License
LAMP	Linux, Apache, MySQL, PHP
UK	United Kingdom
WWF	World Wide Fund for Nature
UNICEF	United Nations Children's Fund
NBA	National Basketball Association
API	Application Programming Interface
OOP	Object Oriented Programming
REST	Representational State Transfer
ID	Identifier
WYSIWYG	What You See Is What You Get
CAT	Computer-Assisted Translation
TM	Translation Memory
DOM	Document Object Model
div	Division
npr	Na primer

1 Introduction

1.1 Dynamic webpages - a brief history

In the early 90's dedicated software had to be downloaded and installed to computers in order to buy things, look up for things, and build things. Those days are long gone now and many things have changed. Especially, the internet has significantly improved. Nowadays, it is expected to be able to do everything by simply using a web browser. Or in other words, it is expected that everything is presented in some form of "web technology". But what does this really mean in simple terms?

You might already know that all websites that we nowadays browse are written in the language of HTML - Hyper Text Markup Language.

In order to make them prettier, some styling needs to be applied. To manage colors, typography, layout, etc. we use CSS - Cascading Style Sheets.

With the addition of some JavaScript into the mix, webpages become more interactive with, for example, things popping up and menus dropping down. It makes the overall experience a bit richer.

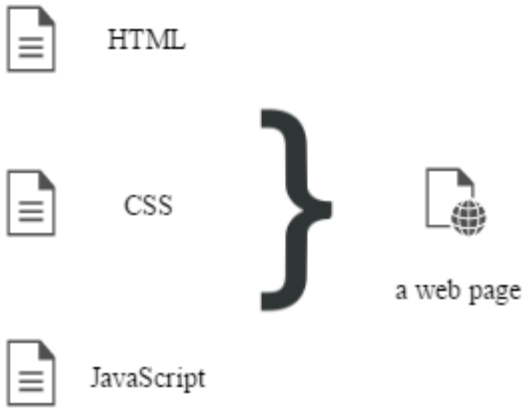

Figure 1: A webpage is nowadays built with HTML, CSS and JavaScript.

Piecing all these technologies together usually requires a very detailed understanding of the three parts - and sometimes it can be too technical for many regular users. In the past, to become fluent in these new "languages", it involved a steep learning curve; you had to be a "coder". But there was another problem - the webpages were only static. There was a big need for a mechanism, that would allow having the actual

content itself separated from the coding part, and come from other dynamic sources to generate websites on the fly.

Therefore, in the late 90's and early 2000's, a whole host of we can refer to as code engines were designed in various programming languages, with the scope to achieve the dynamic approach: CGI, Perl, Python, PHP, Ruby, Cold Fusion, ASP, JSP, and others. They all strived with challenge of separating out the page structure, style and content so that these elements could be organized in a much more efficient way. Instead of the content being embedded in the webpage, it was now stored and retrieved from a database, and the HTML pages were being assembled simultaneously from that data.

This means that our pages can automatically rebuild themselves in response to users' actions; communication became two-way between users and the website. This marks the birth of "Web 2.0" [17].

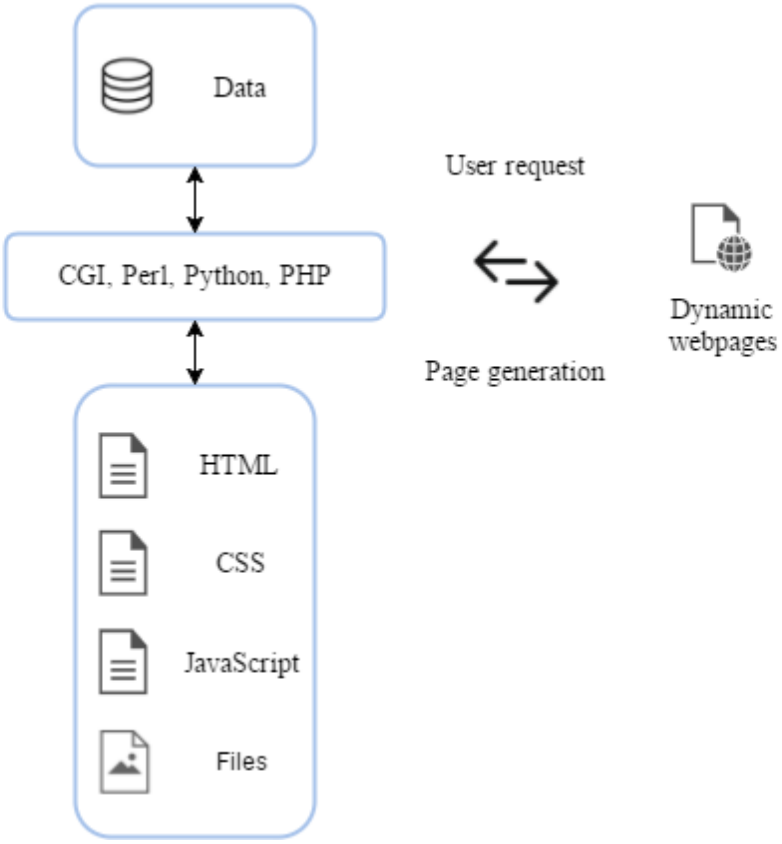

Figure 2: Dynamic webpages are built with programming languages and databases.

1.2 Content management

Contemporary with the growth of the Internet, also the way we understand and use Information Technology within a business context has changed. Research and development were two of the main fields, which advancements have led to technologies such

as; content management, distributed and linear computing, data mining and processing. All of these fulfil a vast range of business needs. Among other factors, the move from localized computing platforms to distributed web technologies has been caused by the mass take-up of commodity computer and network components based on faster hardware and sophisticated software [29].

Content management practices were already established through other fields of management within businesses since the 1980's. Between the late 1995 and 1996 the enormous market potential of the internet became very evident to everyone in the Information Technology sector. As a result, companies started building websites massively and aggressively in order to compete on the market and the need for web content management arose [26].

In the present time, open source Content Management Systems have gained a big market. There are many definitions of what this is as there are many Web CMS analysts and vendors. In simple words, let's define it to be a LEGO-like toolkit for piecing together HTML, CSS, and JavaScript to build state-of-the-art websites for everyone, regardless of their level of experience with web technologies - with the focus on applying management principles to the content. As they are open source, it means that the software is licensed, so that it can be freely used, changed, and shared (in modified or unmodified form) by anyone. It generally supports creation, management, distribution, publishing, and archiving, therefore covering the complete lifecycle of a website. It usually also provides the ability to manage the structure of the site, and the appearance of the published pages with the main purpose of allowing the users with little to no knowledge of web programming languages to create and manage website content with relative ease, as it is based on templates. Therefore, content is separate from presentation - unlike a normal hard-coded site. Content can be edited online and goes live immediately. A CMS can be extended with extra functions and features by adding plugins, as and when the requirement changes. As described in the previous section, this means that CMS is a dynamic system.

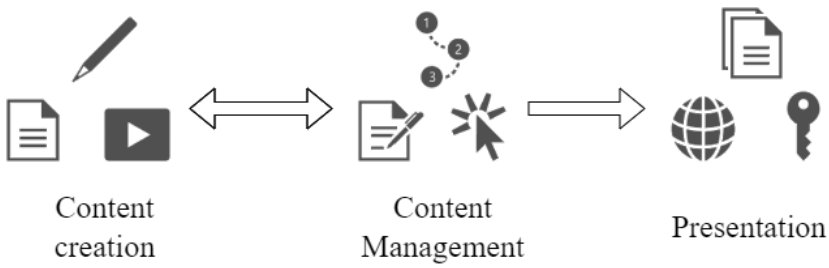

Figure 3: General structure of a content management system.

Content management can be crucial for e-businesses with enhancing e-business technologies and maintaining web recognition on the internet, while being the fundamental

part of the solution to information and data overload. It also contributes to understanding information better, thus adding value and substance to data [19].

The dynamic system clearly has many advantages over the usual static websites that were so popular in the 90s. With the use of a CMS to create a website, a business can greatly benefit in many aspects. Having a streamlined and decentralized authoring process, faster turnaround time for new changes and webpages, improved site navigation, increased site flexibility and security, reduced duplication of information, and above all, reduced site maintenance costs are just some of the key points.

There are lots of CMS available in the market, based on different functionality and platform. Open source solutions are the most popular, since they generally provide good security, quality, customizability, documentation and great support by the community of users and developers. No CMS is best for everyone, but when content management principles are concerned, a few CMS names like Drupal, Joomla and Wordpress stick in mind. There have been many studies made to determine which outperforms the others, but this is really difficult to state, as each system has it's own pros and cons [28] [22].

This paper is based on the Google Summer of Code 2016 project, with the goal of implementing a more functional UI for the Translation Management Tool (TMGMT) in Drupal 8 [11] [10]. Therefore, we will focus on the newest version of Drupal, it's content translation functionality and the extension of the Translation Management Tool module.

To note, Google Summer of Code is a global program focused on bringing more student developers into open source software development. During the summer break from university, students work on a project with one of many open source organizations supported by Google. Since it started in 2005, the program has seen over 12,000 student participants and over 22,000 mentors from 118 countries from all over the world. Google Summer of Code has produced over 30 million lines of code for 567 open source organizations [8].

1.3 Structure of the thesis

Chapter 2 describes in more detail what Drupal is, why and when to choose Drupal over the other available content management systems on the market. Chapter 3 describes the newest version of this program called Drupal 8 in more details. Chapter 4 presents one of the current built-in solutions for content translation and one extension to facilitate translation in Drupal 8 - the Translation Management Tool module. The idea is to make the Translation Management Tool module for Drupal 8 a professional CAT tool by extending the Translation Editor and implementing new features, like

segmentation, tag validation and translation memory. Chapter 5 describes the motivation and implications of a new UI for content translation. The problem is presented and defined as a long term project and only a part of it was implemented in this thesis. In Chapter 6 the tools and languages used are outlined, together with the methods used throughout the implementation process. Thereby in Chapter 7 the main results are presented and divided into two parts, functional (CKEditor plugins) and visible (UI sections). Chapter 8 is dedicated to a discussion of what was achieved and some improvements for future work are proposed.

2 Drupal

2.1 What is Drupal

Drupal is an open source content management software, that allows community-shared code to be assembled to quickly make web sites for any purpose [18].

Originally written by Dries Buytaert as a message board, Drupal became an open source project in 2001, released under the open GNU General Public License (GPL). This guarantees end users the freedom to run, study, share and modify the software. There are no licensing fees, and there will never be. It is build on principles like collaboration, globalism and innovation.

It runs on any computing platform that supports a web server capable of running PHP version 5.4.5+ (e.g. with Apache, Nginx, Litespeed, etc.) and a database (e.g., MySQL, SQLite, or PostgreSQL) to store content and configurations [31]. PHP enables us to build data-driven pages, and as such, provides a manageable split between content, configuration, user accounts, and different media files. It is a very popular open source scripting language that is especially suited for web development because it can be embedded into HTML pages. PHP "pages" are essentially HTML pages that contain embedded code, which is processed on the server dynamically, before it is displayed in the browser. Drupal, in this case, is the smart PHP-based system that brings together different assets from the database, to build the actual visible pages. These are then made available across the web using a web server.

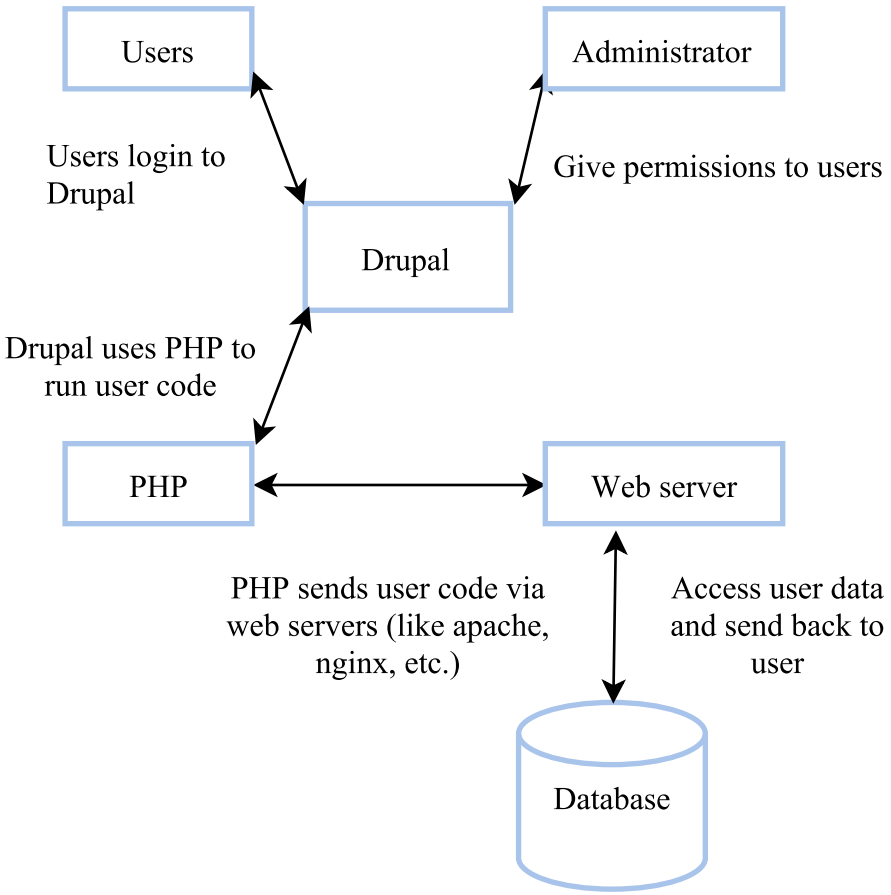

Figure 4: A simple Drupal representation.

As mentioned in Section 1.2, we can think of Drupal as a big LEGO construction kit for "site-builders". Drupal site-builders build web sites by piecing together Drupal modules, which are built on top of Drupal core. Each module is designed to solve a specific problem but in a Drupal-compliant way, so that integration with every other community-written module is still possible. This makes Drupal very extensible and scalable, so that it can grow over time as user's needs expand. More about it in Section 3.2.5.

Drupal is an "out of the box" web content management tool that can drastically accelerate any technology project, website, or startup to get up and running fast and without much IT knowledge. It is used by hundreds of thousands of web developers around the world, and serves as a backend framework for more than $2,2\%$ of all websites worldwide.

Drupal is proven, secure CMS and application framework that stands up to the most critical Internet vulnerabilities in the world to prevent the worst from happening. Drupal is mature, stable and designed with robust security in mind. A dedicated security team, together with a large professional service provider ecosystem, and one of the largest developer communities in the world ensure rapid response to issues [6].

Strong coding standards and rigorous community code review process prevent many security problems that can nowadays happen on the web.

Drupal can be used to build anything, from personal blogs, wikis to bigger corporate home pages, forums, and e-commerce sites. Many big corporate, political and government sites are built using this solution. The United States, Australia, France, London and more use Drupal to communicate with citizens. Media companies like BBC, NBC, The Economist, Al Jazeera, Warner Music Group and MTV UK rely on it to inform and entertain the world. It's also part of how organizations and universities like Amnesty International, WWF, UNICEF, University of Oxford work to make the world a better place.

Drupal core, and the thousands of contributed modules that build on it, requires an initial investment to learn, but mastering the Drupal way can be truly rewarding; the passionate community is a testament to Drupals power to liberate site builders from the simplicity/flexibility dilemma [14].

2.2 Drupal community

Technology and business leaders can transform content management into powerful digital solutions with Drupal, backed by one of the world's most innovative open source communities. The Drupal community is one of the largest open source communities in the world, counting more than 1 million passionate developers, designers, trainers, strategists, coordinators, editors and sponsors working together to build Drupal, provide support, create documentation, share networking opportunities, and more. The shared commitment to the open source spirit from these people is is the driving force of the Drupal project.

One great way to get involved, learn more and get support is groups.drupal.org[1]. This provides a place for groups to organize, meet, and work on projects based on interest or geographic location. There are various Drupal events like Drupal camps and Drupal cons all around the world, where community members can meet face to face, have presentations, swap tips, and get inspiration for new projects, while making friends along the way. The two main channels to communicate and interact with others over the Internet are IRC[2] and Slack[3]. Planet Drupal[4] is a great place to find blog posts, written by the Drupal community, for the Drupal community. It contains code, advocacy, marketing, infrastructure and many other useful information [4]. On the

[1] `https://groups.drupal.org/`
[2] `https://www.drupal.org/irc`
[3] `https://drupalslack.herokuapp.com/`
[4] `https://www.drupal.org/planet`

Community Page[5] we can find the highlighted Drupal community members and teams, which in one way or another contribute to this project.

The main part of the Drupal community is the "issue queue"[6]. This is where all Drupal code is being contributed. Each project, whether a theme or a module, has its own "issue queue". It works as a listing of categorized posts related to the project, which allows any project user to post an "issue" to a specific project version and categorize it (e.g., bug report, feature request) and suggest a priority such as "critical" or "normal". Afterwards, the project maintainer can assign open tasks to colleagues and track the status as the work on it progresses (e.g., active, fixed) [7]. This helps specific project to be more efficiently tracked. Components such as documentation and code are parts of each project and can be set by its maintainer. Some projects, like Drupal core, have many components like "taxonomy.module", "user system" and more. Maintaining an issue queue requires a serious commitment. Verifying reported bugs are still actual bugs, marking duplicate issues, answering support requests, responding to issues constructively and creating and rolling patches are just some of the main things that it involves.

2.2.1 Getting involved

> "It's really the Drupal community and not so much the software that makes the Drupal project what it is. So fostering the Drupal community is actually more important than just managing the code base." (Dries Buytaert)

As an open source project, Drupal does not have any employees to provide improvements and support. The whole system depends on diverse community of volunteers to move the project forward, not by working only on the development, but also on marketing, organizing user groups and camps, speaking at events, helping to review and solve issues in the issue queue, maintaining documentation and more.

There are many reasons why contribute to Drupal. First of all, the fact that more contributions mean less work - for others and for you, is a known fact. For example, when writing documentation or answering questions, it triggers others to respond with their helpful comments and it serves as a steering wheel for the contributed Drupal project. Contributing code to Drupal helps to involve also others to test, repair, improve, maintain and document the written code. It also makes sense, that those who contribute more, have a stronger voice in the project and greater influence on the future of Drupal. The hands-on approach with Drupal, as in every other case, helps to learn about it much faster. For example, taking pieces apart and fixing them or just

[5] `https://www.drupal.org/community-spotlight`
[6] `https://www.drupal.org/project/issues`

discussing on some issues can significantly improve the understanding of the project. Last, but not least, more contribution means more business opportunities. All personal contribution is public and visible on your Drupal profile page. Clients, customers and potential employers around the world can learn your name and see your skills. On the other hand, as Drupal grows, new possibilities may open among companies for new projects with newer functionality.

New Contributor Task section[7] has many great tips and step-by-step guides for new contributors to get started. Everyone has different interests and is free to contribute to Drupal on various fields. There are many areas that one can focus, such as; user support, documentation, translations, testing, design and usability, donations, development, themes, marketing, etc.

All in all, contributing is a great way to show your skills and your interests to the world, meet new people, learn new things on a daily basis and be part of this big open source community.

2.3 Basic concepts

When starting to learn Drupal, there are some important terms, which are used to define the components of the system. These are:

- Modules - software (code) that extends Drupal functionality
- Nodes - any piece of content on the website
- Fields - a reusable piece of content or in technical terms, a primitive data type, with custom validators and widgets for editing and formatters for display (e.g., Title, Body, Comment body, Tags, Image)
- Content Types - predefined collection of data types (Fields) which relate to one another by an informational context
- Entity types - an abstraction to group together fields (e.g., Nodes, Comments, Files, Taxonomy terms and vocabularies, Users)
- Bundles - implementation of an entity type to which fields can be attached
- Entity - instance of a particular entity type (e.g., a comment, taxonomy term, user profile) or of a bundle (e.g., a blog post, article)
- Taxonomy - classified (grouped) content

[7] `https://www.drupal.org/contributor-tasks`

- Regions and blocks - pages are laid out in regions, which are formed by blocks of information
- Views - allows people to choose a list of nodes or other entities and present them as pages, blocks, RSS feeds, or other formats
- Theme - controls the appearance of the website

We will not go in much details about what each of these terms mean (except for Modules in the next chapter - 3.2.5) but understanding these terms and how they relate to one another and using the correct terminology plays a crucial role especially when reporting issues and communicating with the community.

3 Drupal 8

Drupal 8 is definitely the most significant update in Drupal history with more than 200 new features and improvements. Easier customizations of data structures, listings and pages are the first noticeable changes. Countless new possibilities for displaying data on mobile devices, building APIs and adapting the website to multilingual needs are also some of the other things that improve the overall usability and diversity of this platform. With a much more efficient core, easier migration from earlier versions and in-line content editing tools it is a cutting-edge platform that is setting new standards for other CMS [27].

Since its creation in 2001, Drupal has grown and developed to meet new changing demands of all its global users. To achieve that, new forward-looking changes were made with every update. As a result, Drupal stayed relevant to new technologies, unlike nearly every other open source CMS. As the founder Dries Buytaert already noted in 2006, with every major release, developers have gone through a lot of pain in order to adopt these changes. Here are his words on this matter:

> "So let's capture that thought for future reference. Sweeping changes are required to make major advances in technology, and often times there is a lot of pain before the pay-off." (Dries Buytaert)

Although Drupal 7 is still a very popular CMS among users, it has some important limitations. The following are just some of them [23]:

1. Incomplete Entity API.
2. Lack of separation between content and configuration in the Database.
3. Lack of separation between logic and presentation in the theme layer.
4. It is hard to use for many individuals, thus hard to find Drupal talent (developers, themers, site builders, etc.) for organizations.

Contributed modules tried to solve many of these problems, but they were mostly incomplete and difficult to manage when building complex websites. Thus with the launch of Drupal 8 these limitations were taken into consideration and a significant amount of work has been done to make all the new features available. Together with the

community it has been decided to take a head-on approach - through the Configuration Management Initiative, Twig templating layer and a completely new Entity API. At the same time the above mentioned issue of Drupal 7 being complex for many individuals was taken care with the launch of Drupal 8. The code is more abstract, verbose and yet, more maintainable and accessible to non Drupal developers.

One of the biggest challenges with Drupal, is the difficulty for organizations to find Drupal developers. Drupal 7 did not address this issue. In fact, it made it even worse with the use of procedural programming with even more Drupal-specific development concepts (e.g., excessive use of structured arrays). The most effective way to solve the Drupal talent issue, as well as complexity issue, is to update Drupal with modern frameworks and platforms, in order to minimize the Drupal-specific knowledge and become more proficient. For that matter, modern PHP 7 concepts and standards like PSR-4[1], and some external libraries like Composer, Guzzle, Zend Feed Component and Assetic were adopted.

Drupal 8 is now based on some core Symfony components which is a drastic change from the previous version. Symfony, a PHP framework developed by SensioLabs, enforces the use of the namespaced PHP classes, instead of global functions - a development methodology, known as **Object Oriented Programming (OOP)**. This allows developers to create dynamic relationships between these objects and advantage of not having to change modules when a new type of object is added.

The advantages and disadvantages of object oriented programming are well-known. Verbosity, size, slower performance and the amount it takes to write code are just some of the downsides. For people that are new to object oriented programming, the learning curve to absorb object oriented concepts can be steep as some of the key programming techniques, such as inheritance and polymorphism, can be initially quite challenging. On the other hand, it has many benefits. Mainly, it allows to break software code into smaller parts. Those can be then designed separately, tested and re-used. Bigger problems are broken into smaller ones, which are then easier to solve. This results in:

- cleaner code,
- better architecture,
- abstraction layers,
- modular solutions,
- less bugs,
- easier refactoring.

[1] `http://www.php-fig.org/psr/psr-4/`

For Drupal 8 this means that the code will be more abstract, more verbose and slower, but also more maintainable, modular and on-boarding for new Drupal developers.

Another really helpful feature for newcomers is the "guided tour". The descriptive text is now placed right under the help link. After the user clicks on it, explanations are provided on how things should be done and how things work in a pop-up window. This user-friendly boost is well-received and it's making the CMS easier for everyone to understand [25].

As already mentioned above, there are also some news regarding the theming part of Drupal. In Drupal 8 **Twig** replaces PHPTemplate as the default templating engine. It is PHP-based, flexible, fast, and secure. It's much easier to create beautiful and more functional Drupal websites using Twig, as its templates are written in a syntax that's less complex than a PHP template or others while being more secure. Drupal allows overriding all of the templates that are used to produce HTML markup in order to allow full control of the markup that is being output within a custom theme. There are templates for each page element ranging from smaller fields to HTML elements on higher levels.

Responsive designs are a must these days, and for Drupal 8 it was a big priority. Drupal 8 is **mobile first** in its approach. All the default themes that come with Drupal 8 are responsive, along with an administration theme that adapts to different screen sizes, and a *'Back To Site'* button to go back to the front page. Tables fit into any screen size without a hitch, and the new administration toolbar was redesigned to be more extensible, concise and to work well on all mobile devices. Toolbar's top-level items include: Home, Menu, Shortcuts and Users. For mobile, icons replace textual labels to save on screen size and to provide a cue for usage.

One of the very useful features that also come in Drupal 8 is the new **configuration management system**. While it strives to make the process of exporting and importing site configuration feel almost effortless, immensely complex logic facilitates this process. Over the past five years, the entire configuration system code was written and rewritten multiple times in order to find a proper solution. As a result of this work, it is now possible to store configuration data in a consistent manner and to manage changes to configuration [30]. The configuration system is designed to optimize the process of moving configuration elements between instances of the same site (e.g., from local development to the server). It is not intended to allow exporting the configuration from one site to another. The file storage format for configuration information is defined as YAML files and stored in the *'sites/name/files/config_HASH/active'* directory. What is configured in the file is defined by the file name. For example, *'views.view.content.yml'* contains a complete definition of the view with machine name

'content'. Drupal does this by using configuration prefixes. When the user creates a new field, node type or a view via the Drupal UI a new file is created. These files can be easily tracked by use a version-control system like Git, for example.

Content editing is also a field that was redefined. Content authoring experience was vastly improved thanks to the Spark initiative. Content editors can now use the rich functionality of the **WYSIWYG editor - CKEditor**. The configuration for text formats and editors are now located in one place. The default installation profile of Drupal 8 configures your site for Basic HTML and Full HTML text formats. Customization of the toolbar is also very easy with a drag-and-drop interface. The initial release of Drupal 8 (8.0.0) did not support any browser's built-in spell checker with CKeditor to check the written text. This function was added with the Drupal 8.1.0 update. Another great improvement is the addition of the optional language markup button in CKEditor. When configured to appear in your editing toolbar, it allows you to assign language information to parts of the text, which is useful for accessibility and machine processing.

In addition to the WYSIWYG functionality, the other most touted improvement is **in-line editing**. In-line editing is the ability to edit content directly on the page the content is displayed, without leaving the page, opening new tabs or overlays; you can now click a link provided in the contextual links menu (now displayed on hover as a pencil) to edit the page directly. You just hover over the section you want to edit and you are able to modify the content with a WYSWYG tool-bar hovering above the content.

Figure 5: It's really easy to edit content with the new in-line editing feature also on mobile phones.

Views are high up in the Drupal module hierarchy, as it is used in almost every Drupal 7 website, and a lot is is quite impossible to accomplish without it. Site builders and designers have used it to display galleries, maps, graphs, lists, tables, menus, blocks and more. With Drupal 8, Views is firmly integrated in Drupal core. Several administration pages and the front page are now Views. Users will now be able to quickly create pages and modify existing ones effortlessly, without installing any extra modules.

The fact that Drupal 8 ships with a big variety of **field types in core**, takes its content structure capabilities up a notch. New field types like link, date, e-mail, telephone, entity reference, etc., help with content creation. And now there are many new possibilities, such as attaching fields to more content types and creating custom contact forms by attaching fields to them.

Also the overall performance of the Drupal website was improved. Drupal 8 now caches all entities and only loads JavaScript when necessary. This means that the previously viewed content is quickly loaded from the cache instead of reloading it from the server. The inclusion of **BigPipe**[2] **in core** enables developers to improve the front end/perceived performance for end-users, by marking personalized parts of the

[2] `https://www.drupal.org/project/big_pipe`

website as cacheable or uncacheable parts (placeholders). By default, Drupal 8 uses the "Single Flush" strategy for replacing the placeholders when the initial page loads. In this case the response isn't send until all placeholders are replaced. Facebook's BigPipe introduces a new rendering strategy, that allows the initial page to be flushed first, and then the placeholders are replaced in a stream.

As the goal is to have a stable platform with as few bugs as possible, the support for **automated testing of JavaScript** was added. Developers can now test the JavaScript front end automatically, saving time and making continuous integration much faster. There are also other improvements to the testing system, including improved reporting of PHPUnit and other test results [3].

One of the most important pieces of this mosaic are built-in web services. This is a major sea change for the platform. Drupal content entities can now be interacted via the **RESTful Web Services API**, allowing for the design of more tightly networked web applications [1]. In other words, this makes it possible to use Drupal itself as a data source, and output content as JSON or XML. Consequently, this means we can completely separate backend from the front end part of the project and have different teams work on them independently. With this approach, Drupal can become a completely "*decoupled*" or "*headless*" CMS. This leverages client-side frameworks and enables the use of many new technologies; a powerful combination in this matter is ReactJS, Relay and GraphQL.

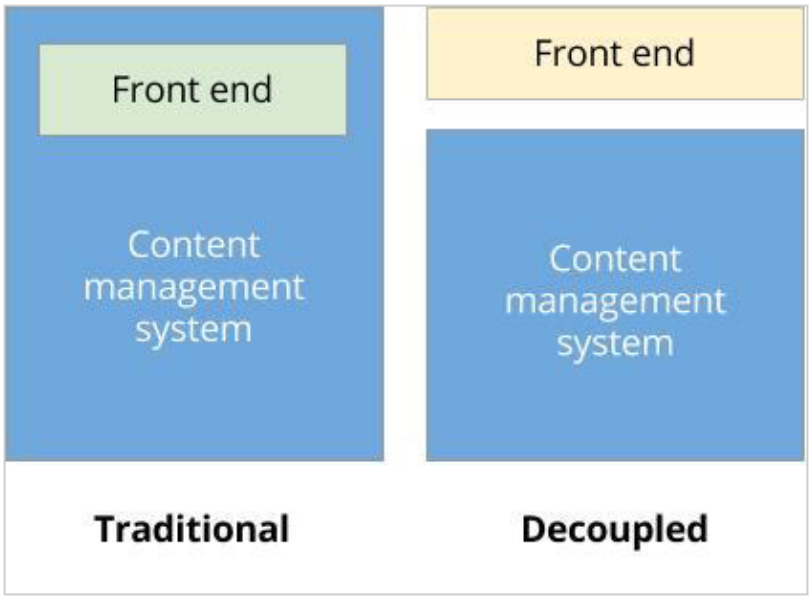

Figure 6: Traditional ("monolithic") versus fully decoupled ("headless") architectural paradigm. [20].

Last, but definitely not least, Drupal 8 provides extensive **multilingual** features

right out of the box. The administration interface has built-in translations. Also, pages can be created with specific language-based Views filtering and block visibility. Translation updates from the community are being constantly reviewed and added. We will present translation in Drupal 8 in more details in Chapter 4.

3.1 Drupal 8 core release cycle

The Drupal community was working hard since March, 2011 on Drupal 8 and the progress was constantly documented[3]. The first stable version of Drupal 8 was released on 19th November 2015. The date was chosen based on the drastic slow-down of incoming critical issues which ensured a stable release. It was followed by many release parties around the globe, organized by contributors themselves. This key milestone was achieved thanks to more than 2,300 people. There have been 15 alpha releases with more than 11,500 committed patches.

Starting with Drupal 8.0.0, Drupal core releases moved to a new release cycle schedule using semantic versioning numbering system. *Patch releases* (8.0.1, 8.0.2, etc.) are released monthly and contain bug fixes. *Minor releases* (8.1.0, 8.2.0, etc.) are released approximately every six months and incorporate new features. The final minor release in the 8.x series will be a long-term support (LTS) release. Drupal 9.0.x will be branched around the same time as the 8.x LTS or possibly beforehand, depending on the level of remaining work in both branches [5]. The same release cycle will be used for Drupal 9.

[3] `https://www.drupal.org/core/dev-cycle`

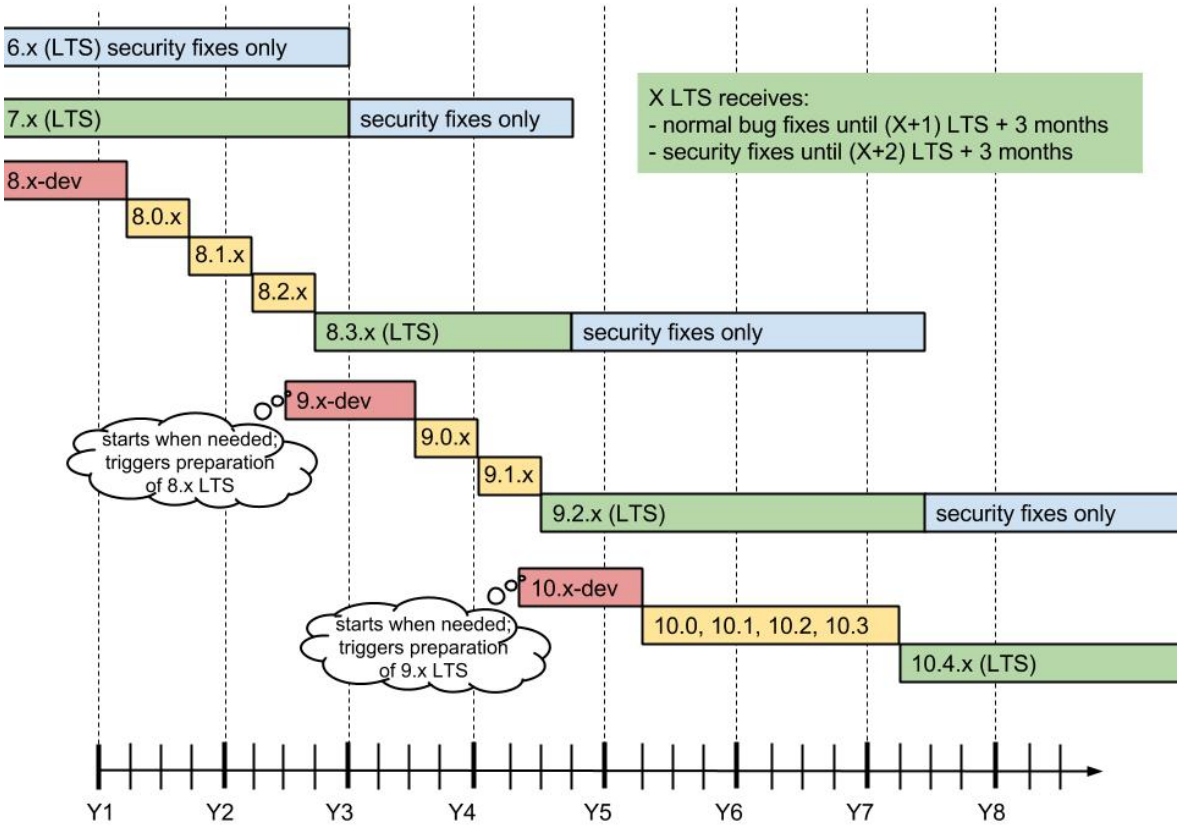

Figure 7: Drupal timeline plan for specific version releases. The plan is well defined, but the actual schedule may vary.

3.2 Architecture

Drupal 8 has seen a big revolution in terms of architecture, unlike Drupal 7 compared to the previous version. These big changes were necessary in order to professionalize and modernize the CMS. We already discussed the shift towards object oriented programming and Symfony in Chapter 3. Let's dig deeper into this topic and how Drupal 8 is actually built.

3.2.1 Symfony2

The Symfony2 framework is designed to build custom web-based applications. It does not function as a CMS, as it can't be used to administer a site. It provides an efficient way to create an application solely by writing code. In theory, Drupal could be build entirely on top of Symfony, but in practice, the default approach of Symfony was not found flexible enough. That is why we emphasized before that Drupal 8 is using just the core layers and extends them to provide support for Drupal modules. The result

is the best of both worlds; a well-balanced system, that uses a large and unmodified part of Symfony core, but extends it with a very flexible CMS layer.

The Symfony framework is based on several components. Some are crucial to the system, like the HttpFoundation component. This understands HTTP and offers a structured request and response object that is used by other components. Others, like Validator, are just helper components. As the name already says, the heart of the system is the Kernel component, which is basically the "main class" that manages the environment (services and bundles) and is responsible for handling a http request. By extending the Kernel, we can build bundles. These can be used to create "coherent" pieces of functionality, like modules in Drupal [15].

Drupal 8 is built on top of **DrupalKernel**, which is the heart of the CMS, and some major Symfony components, such as `HttpFoundation, HttpKernel, Routing, EventDispatcher, DependencyInjection`, and `ClassLoader`. Other than that, it also includes various Drupal-specific and third-party components. Drupal does not extend *Kernel*, like other Symfony web applications, but offers the same type of functionality by implementing the `Kernel` Interface. Because bundles are not very flexible and extensible, the bundle approach could not be used in this situation. The available services and modules are loaded by the `DrupalKernel` in a slightly different way than Kernel, but as Symfony's `Kernel` does, it delegates requests handling to the `HttpKernel`. By adopting the `HttpKernel`, Drupal and Symfony projects will become more interoperable. It means that integration of your custom Syphony application with Drupal and vice-versa is much simplified. This is a great benefit for both communities.

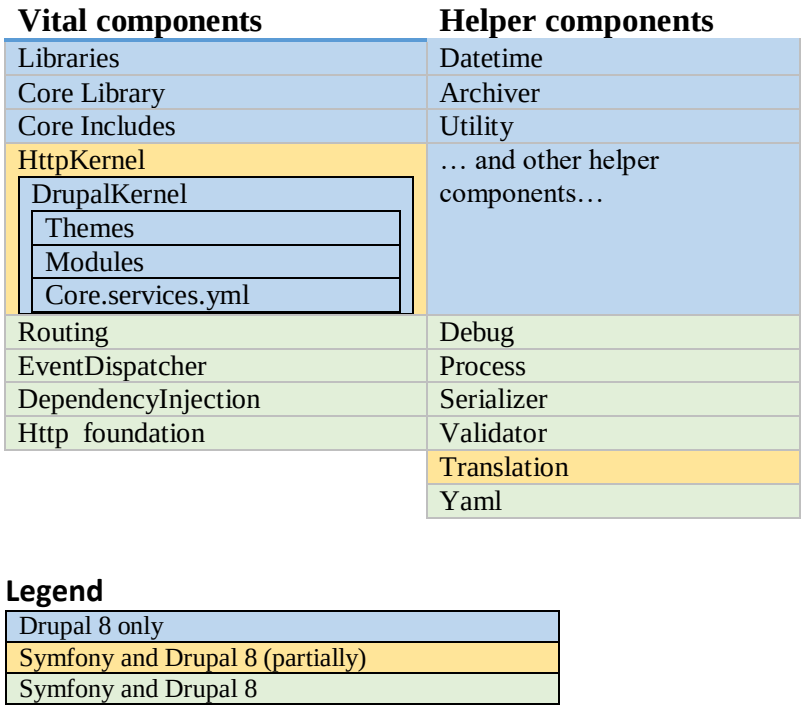

Figure 8: Drupal 8 fundamental layer is based on components.

3.2.2 Service container

The backbone of Drupal 8 is considered to be the service container. It is used to efficiently manage services in the application. This concept is also known as **Dependency Injection**. Services are global objects that can be used to do specific tasks, for example Mailer service, or a database connector. A service corresponds to exactly one class. The service container is very important, because it contains the available services, knows about their relations (dependencies) and configurations, and even constructs them. Interfaces are used to define which methods the dependent services should provide, in order to swap the service implementation with another one if necessary.

As previously noted, also service configuration is based on YAML files. Drupal 8 uses `services.core.yml` for everything core-related. The configuration can be extended by modules and `ServiceProvider` classes and their loading is handled by the Kernel.

YAML provides a flexible and readable syntax, but users need to be very careful with the indentation. An example in Drupal 8, taken from `services.core.yml` is displayed below:

```
uuid: ''
name: ''
```

```
mail: ''
slogan: ''
page:
  403: ''
  404: ''
  front: /user/login
admin_compact_mode: false
weight_select_max: 100
langcode: en
default_langcode: en
```

code/services.yml

Here the `router_listener` service is defined. To load the listener, it is required to specify the corresponding class. The arguments property defines that the first argument of the RouterListener constructor should be `'@router'`. This also defines the service with the service id `'router'`.

3.2.3 Flow of control

In Section 3.2.1, we stated that the basic workflow in Drupal 8 is based on requests and responses. The general flow is the following [16]:

1. Bootstrap configuration:
   - Read the `settings.php` file, generate some other settings dynamically, and store them in global variables in the `Drupal\Component\Utility\Settings` singleton object
   - Start the class loader, which takes care of loading classes.
   - Set the Drupal error handler.
   - Detect if Drupal is installed. If not, redirect to the installer script.
2. Create the Drupal kernel.
3. Initialize the service container (either from cache or from rebuild).
4. Add the container to the Drupal static class.
5. Try to serve page from the static page cache.
6. Load all variables.
7. Load other necessary include files.

8. Register stream wrappers (public://, private://, temp:// and custom wrappers).
9. Create the HTTP Request object (using the Symfony `HttpFoundation` component).
10. Let the `DrupalKernel` handle it and return a response.
11. Send the response.
12. Terminate the request (modules can act upon this event).

When a request enters Drupal, the system is bootstrapped and the `DrupalKernel` is booted. Symfony's `HttpKernel` handles the request, delegated from the handle method of the `DrupalKernel`. The `HttpKernel` dispatches the '`kernel.request`' event with several listening subscribers, each with a specific function:

- loading the session and setting the global user (`AuthenticationSubscriber`)
- detecting the current language (`LanguageRequestSubscriber`)
- converting the url to a system path (`PathSubscriber`)
- allowing setting the custom theme and initialising it (`LegacyRequestSubscriber`)
- showing the maintenance page, if in maintenance mode (`MaintenanceModeSubscriber`)
- getting the fully loaded router object (`RouteListener`)
- checking if the client has access to the router object (`AccessSubscriber`)

The `RouteListener` is where the routing work happens. In Drupal 7, `hook_menu()` was used to register page callbacks along with their titles, arguments and access requirements. Drupal 8 instead, opts for the routing process of Symfony, which is much more flexible, but very complex at the same time. Page callbacks are no longer functions. Instead, they are methods in controller classes. Routes configuration are now stored in `{module}.routing.yml` files in the respective module folder.

As an example, the user logout page route configuration looks like this:

```
user.logout:
  path: '/user/logout'
 defaults:
    _controller: '\Drupal\user\Controller\UserController::logout'
  requirements:
    _user_is_logged_in: 'TRUE'
```

code/user_logout.routing.yml

Every route needs to have an id and a path. The *'defaults'* section is very important as it is used to control what needs to be done in case the request matches the path. The *'requirements'* section defines if the request should be handled at all. This is checked through access checks, the Symfony alternative to *'access arguments'* and *access callback*, found in Drupal 7.

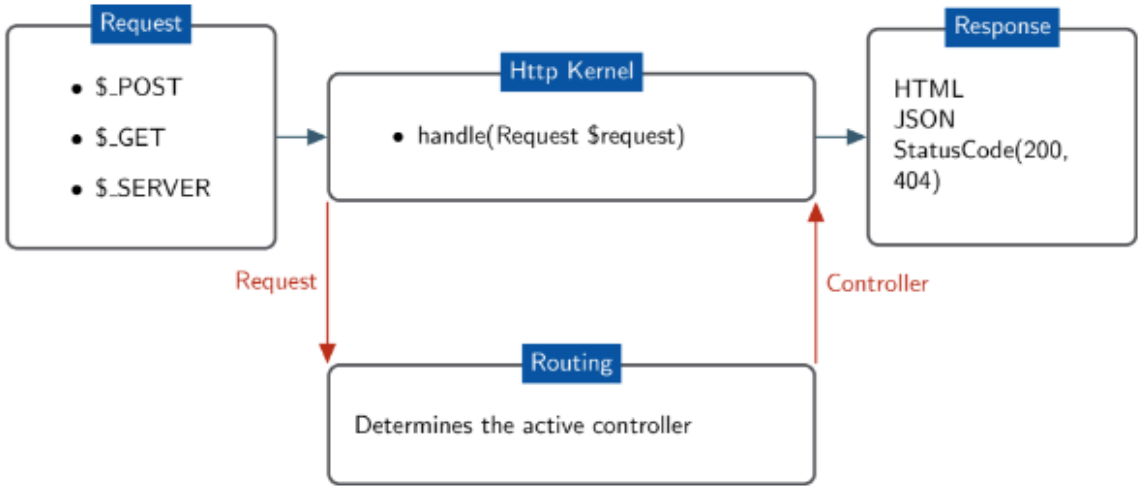

Figure 9: Drupal's routing system works with the Symfony HTTP Kernel and is responsible for matching paths to controllers,

3.2.4 Entities

Entities were introduced in Drupal 7 as generalized objects with the same functionality. For example, fields could be attached, access methods were generalized and referenced in views, etc. This concept was found to be extremely powerful. In fact, it got extended and it has become a even more central part in Drupal 8. All content data, like nodes, taxonomy terms and users are now entities. Also non-fieldable data, such as views, actions, menus, image styles, etc. and bundles extend the `Entity` class. Even though these object types are not fieldable, functionality like generalized storage in configuration files or automatic route parameter conversion based on the entity name is very handy.

3.2.5 Plugin system

Another important aspect of Drupal 8 new architecture is the plugin system. As we discussed in Chapter 2.1, Drupal 8 is highly modular in its design, thanks to this

feature. It is used to provide a set of guidelines and reusable code components to allow developers to expose pluggable components - plugins, within their code and (as needed) support managing these components through the user interface [9]. In software engineering we call this a design pattern. Furthermore, Plugins that perform similar functionality are of the same plugin type. Drupal 8 contains many different plugins, of different types. For example, each different field widget type is a plugin, and all together form a `Field widget` plugin type.

Plugins are defined by modules: a module may provide plugins of different types, and different modules may provide their own plugins of a particular type. By enabling or disabling modules, you can switch on or off various bits of functionality. The system can also be extended by adding new ones.

The plugin system has three base elements:

- Plugin Type - created by the `Plugin Manager`, this is the central controlling class that defines how the plugins of this type will be discovered and instantiated. The type describes the main purpose of all plugins of that type; e.g. cache backends, blocks, image actions, etc.
- Plugin Discovery - the process of finding plugins within the available code base that can be used within this particular plugin type's use case.
- Plugin Factory - is responsible for instantiating the specific plugins chosen for a given use case.

Other than the base elements, the plugin system includes several situationally useful components:

- Plugin Derivatives - allow a single plugin to act in place of many. This is useful when user entered data might have an impact on available plugins.
- Discovery Decorators - another available discovery method that wraps an existing discovery method.
- Plugin Mappers - allow the mapping something (most often a string) to a specific plugin instance.

Plugins are much like PHP native interfaces with some additions: the plugin system can discover every implementation of an interface (the default is magic namespacing), deals with metadata (by default this is provided by annotations) and provides a factory for the plugin classes.

Plugins can sometimes be confused with services. The main difference is that plugins implement different behaviors via a common interface, while services provide the same functionality, and are interchangeable, differing only in their internal implementation. A good example of when plugins are used, are image transformations. The most commonly used image transformations are scale, crop, desaturate, and so on. All of these act in the same way on the same data - it accepts an image file, performs a transformation, and then returns the altered image. However, each effect does something different.

On the other hand, for caches we use services, as the user just expects the caching to be done and cache can be cleared and replaced without any functional difference.

4 Translation in Drupal 8

While English is the lingua franca for business, only about five percent of the world's population speaks it as their first language, and more than 70 percent don't speak it at all. Even in Europe, where English is widely known, people value tools in their own language. Translation in Drupal 7 was very difficult and complex. It involved enabling a number of modules to do all the translations properly.

In Drupal 8, big steps forward were done in terms of multilingual support with the Multilingual Initiative. Since its introduction in May, 2011, huge efforts by everyone involved resulted in hundreds of issues resolved and many great improvements that will simplify site-builder's process.

Drupal 8 makes language selection occupy the first step in the installer. Compared to Drupal 7, where you were presented with a long page with instructions on how to locate and download a translation file, place into a specific directory and reload the page, Drupal 8 comes with the realization that all these tasks can be automated. The users are greeted with about 100 languages to choose from when installing Drupal 8. It also comes with a highly improved browser based language detection capabilities, so it will attempt to automatically identify users preferred language for the installation process. Also, the translations for the system are downloaded and imported when the language is selected, instead of doing that manually like before. The installer can now also fully display languages that are written from right to left.

Once Drupal 8 is installed in a foreign language, the Language and Interface translation modules will be enabled with the chosen language configured. Drupal 8 has more core modules handling language related features, yet less requirement for contributed modules to be installed for the most important tasks. In fact, the 4 modules mentioned below, cover 20+ modules from Drupal 7 and in much better ways [21]. The question still remains, why do we need 4 modules when a multilingual site just needs all the features? The answer is straightforward; there are technological reasons to organize the modules by the main features they provide - for better maintenance and support. What is more, this kind of separation also provides better support for foreign language (not multilingual) sites.

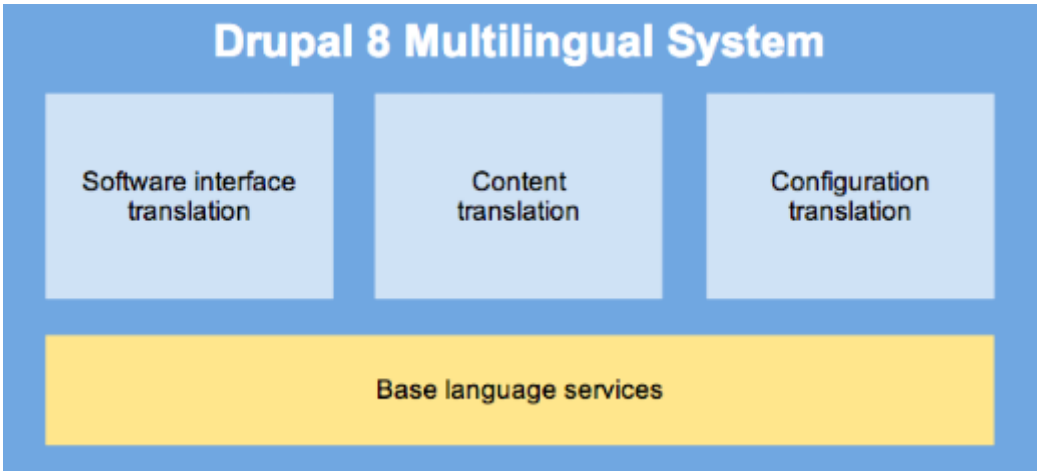

Figure 10: The multilingual system is based on four modules; Interface Translation, Content Translation, Configuration Translation and Language.

The **Language** module serves as the base layer for language support. The purpose of this module is to provide language configuration, assignment and detection functionality. The language overview is straightforward. It lets users reorder existing languages, remove them (except the site default language) and add new languages. It is not anymore possible to have enabled and disabled languages on your website - this forces the use of proven content staging techniques for new language content. Another change from Drupal 7 is that the removal feature applies also to English, even if it is not set as the default language. In prior Drupal versions this wasn't possible, because the built-in interface relied on it. This restriction was now lifted, as many of single language sites will not need English configured at all - they will be completely installed in a foreign language at first. A simple example; if you have a multilingual blog with an English only interface but posts in different languages, enabling the Language module will be the only thing you need. It will simply let you assign languages to content.

Once you need your website interface to "speak" different languages, you'll need the **Interface Translation** module. The combination of Language and Interface Translation modules is what was Locale module in Drupal 7, but the two new modules have a lot more useful features and expanded on their reach a great deal in providing support for your work. This module lets you configure a different language for the built-in user interface of the page (such as registration forms, content submission and administration interfaces) from the content.

The **Configuration Translation** module's only purpose is to provide a user interface to the existing backend support for configuration translation that is native to Drupal 8. Views, contact module categories, vocabularies, menus, and so on are all stored withing the unified system and can be translated with this module. Furthermore, as everything is in blocks now, from page title, to branding, breadcrumbs, menus,

navigation tabs, etc., it is possible to configure them as "interface blocks" or "content blocks", in order to determine which negotiated language will apply. As for contents of the blocks themselves, the built in Views module can be used to do listings to pull data from entities with versatile language options. Drupal now provides a full set of tools for site builders to dynamically apply language based conditions to all parts of the website in detail. To note: configuration translation relies on having a correct configuration schema to provide translations. So, every module must provide a correct schema[1] [12].

We already saw that the configuration is treated very differently in Drupal 8 compared to Drupal 7 in Chapter 3. YAML files are really simple text based files that have an internal nested-tree data format to them. From the Drupal perspective, they are just a simple way to store a nested array of data. For language support, the granularity of these files is very important, because languages are stored/supported on the file level. The top level 'langcode' key sets up the boundaries of language assignment granularity for Drupal 8 configuration. In other words, each configuration file holds configurations that are in the same language. This enables easy multilingual site configurations. You would think that this ends up in a mess of configuration files in all kinds of languages, but this is not the case. Drupal keeps track of configuration language information on each file, which lets us translate them to other languages. The way we translate them is not to duplicate the file but instead to use overrides which are loaded on top of the base file when needed. So our configuration translation system uses the same base configuration files but allows to replace textual elements in the configuration to ones in other languages.

All the originally shipped configuration is considered part of the software and is translated with the community translation system on the official Drupal 8 localization page [2]. The system is integrated with the translation download and update system in Drupal core, so all community translations for shipped views, user emails, contact categories, etc. are downloaded in the installer and translation updates for them are available on the site.

Finally, Drupal core also ships with a **Content Translation** module as the main method for translating content. It has an approach very similar to the Drupal 7 *Entity Translation module*. With this process, only a single node or entity is created, where the entity is language-independent and only the associated fields are flagged with a language. Content Translation module is fairly customizable. It lets users decide whether each type of content entity (node pages, comments, custom blocks, taxonomy terms, etc.) should be translatable or not. Also the sub-types within each entity type, like content types for node page content or terms in particular vocabularies for

[1] `https://www.drupal.org/docs/8/multilingual/translating-configuration`
[2] `https://localize.drupal.org/`

taxonomy, can be translated. And within each translatable entity sub-type, users can decide which fields should be translatable.

The translation process is quite simple. The Content Translation module needs to be enabled and properly configured. Once done, at least two languages on the website should be enabled. Users with translate permission can then see links to "Translate", alongside other tabs like "View", "Edit" and "Delete" and in the "Operations" drop-down menu.

Translations share the same ID (such as the node ID) with the original node. In the Drupal database, there is a specific table that stores information about each translation with the corresponding language and the node ID. Each field stores its translated values in a separate table.

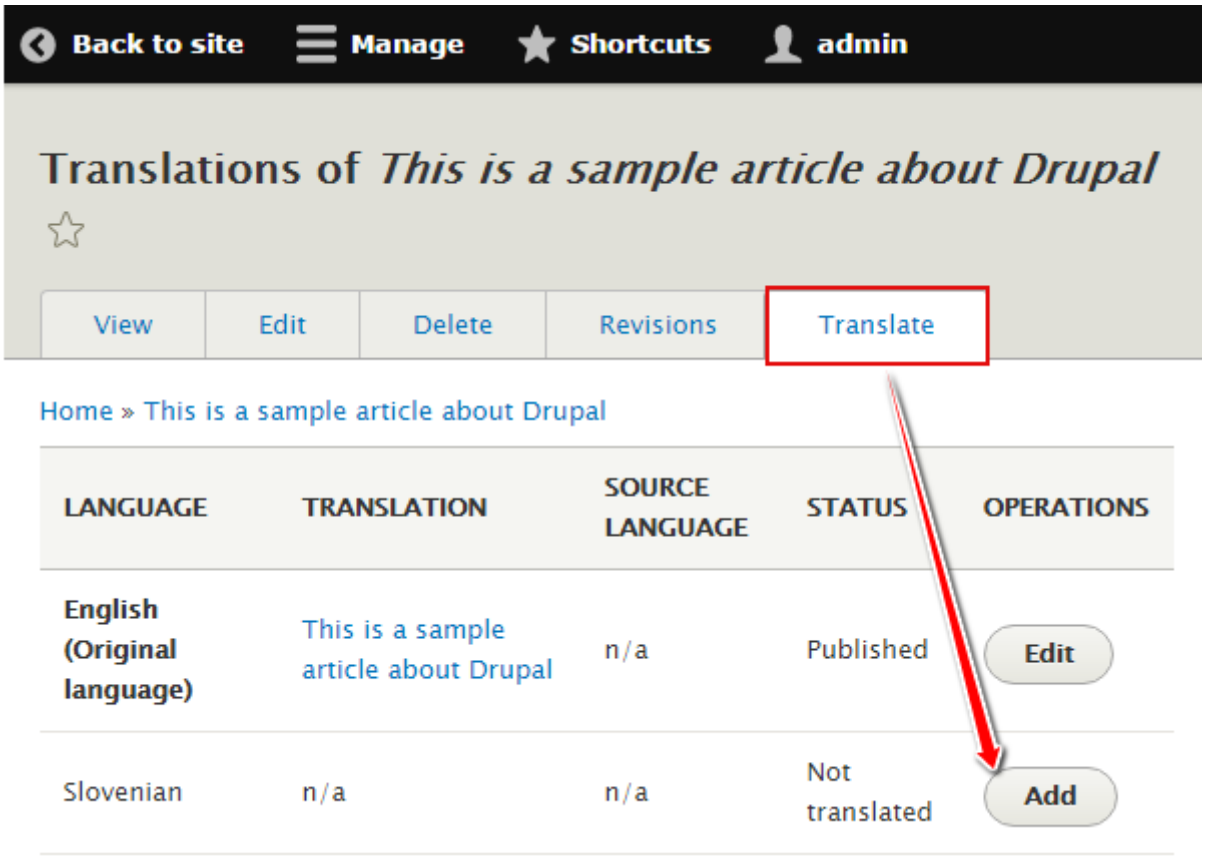

Figure 11: Translation with the Content Translation module found in Drupal core is possible in just a few clicks.

These four modules cover a wide range of translation use cases and conveniently make numerous contributed modules obsolete, but there are some exceptions!

4.1 Translation Management Tool

The Translation Management Tool (TMGMT) module provides a tool set for translating content from different sources [13]. The idea behind TMGMT was born in 2011 and the project started in 2012. Initial development of this module was sponsored by MD Systems, Amazee Labs, S.W.I.S. Group, Microsoft, Acquia and Supertext. The plan was to build an extension to support editors, publishers, translators and project managers during their process of content translation with the goal to solve all the confusion and problems that were arising while doing translation in Drupal 7.

An example in Drupal 7 would be the following. Let's say there were 100 content nodes on the page and we needed to translate it to 5 different languages. To do this, we would end up with a huge number of nodes, which all contain the same content, but in different languages. To maintain all the nodes and translations was a real struggle and it was impossible to see and manage the status of the translations. Also, there was no proficient workflow - external services were not supported and the translator had to log into the site configuration to do the job. With Translation Management Tool most of these problems are solved and with it, the translation with Drupal is streamlined and user-friendly.

Note that this module does not make the default built-in translation module in Drupal 8, i18n or any other language module obsolete. In fact, it facilitates the translation process. It builds on and uses existing language tools and data structures in Drupal and can be used in automated workflows.

4.1.1 Architecture

The architecture of the module consists of 3 major parts. The sources expose translatable content of any type, may it be content, configuration and interface texts or other entities. In TMGMT these sources are named source plugins and are added to the translation job, as seen in Figure 13. Each of the source plugins in a job is called a job item.

On the other side are the so called translators, which are responsible for getting the requested sources translated. The translation of the job can be done by local or remote translators (also called translation plugins) of different kinds and the translation process can be totally automated. This means that external services can be used for creating a foreign language version of the source, but also the users themselves can translate the text via the *Local translator* or the *File translator* - both are included in the module by default. The Local translator allows managing translations on the site so that the content is translated in a central place with defined workflows. The File translator enables exporting jobs into files and importing them once they have

been translated. It contains a pluggable system to support various file formats, such as XLIFF and HTML.

The core system combine these two main parts and provide the ability to create, manage and review translation jobs. The main features of the core system include:

- Creation of translations and managing their progress.
- Review of returned translations, ability to request revisions and communicate with the translator if supported.
- The same information is provided on the translate tab of the supported sources.
- A suggestions system that makes recommendations about related content that could be translated with the same job.
- Sources can declare which parts of a source text should not be translated, for example placeholders for user interface strings.

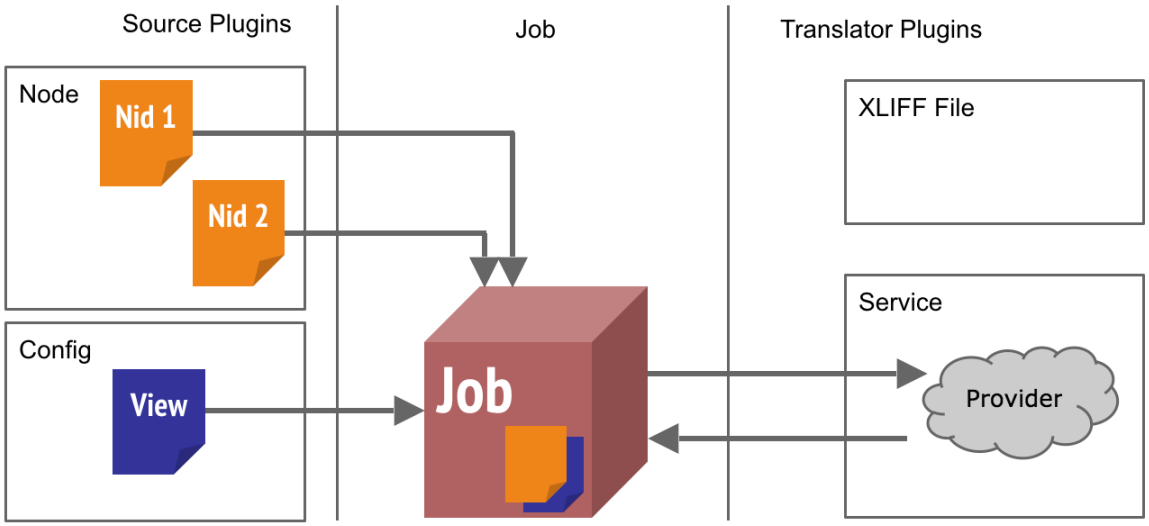

Figure 12: TMGMT architecture is made out of three basic parts; Source Plugins, Jobs - as TMGMT core functionality and Translator Plugins.

4.1.2 Translation workflow

To get started, users need to make sure to have downloaded all of the listed dependencies on the module page, and enable one or more languages on the site. The content type that will be translated needs to be set as multilingual in the "Content language and translation" menu.

Having TMGMT installed, gives users new possibilities on the translate tab of any node on a multilingual site. Translations can be requested for single or multiple nodes

and for various languages at once. Once selected, the user is prompted to a window, where translator providers can be configured. Not only translations can be added manually by different users, but accordingly to the chosen language, users can also request a translation to external translation providers. Configuring this settings properly results in a translation job being created. If a machine translator was selected, the job gets submitted to the service and it comes back in a translated fashion. The status changes accordingly on the translation overview page - the first status is obviously always "needs review". On the review page, users can check the translations, correct them if needed, mark each field as completed and accept the translation. Content and HTML tag validation are also two options that "local translators" can consider in this step. Translation revisions are often helpful when looking back at the changes made during the translation process.

The module also provides overviews for the supported sources that allow translating multiple pieces of content (job items) in a single job and see the current translation status for the site content.

Beside default jobs, TMGMT additionally provides "Continuous Jobs" for supported translators. Continuous jobs' function is to automatically and continuously submit all new and updated content to the configured translator. Each job has a specific language pair while multiple jobs can be created to translate into and from different languages. Continuous job items can be submitted in groups on a cron run.

4.1.3 Current state

The Translation Management Tool module reached a major milestone in the middle of the year 2016 and became the standard for multilingual workflows. More than 3000 sites are currently using this module and it has been downloaded more than 29000 times [13]. The Drupal 8 port started in 2013 and up until now, all Drupal 7 features were successfully ported with strict test coverage, review cycles and driven by the *drupal.org* issue queue. It also maturely evolved with many new features and more than 10 new service providers. As described above, the supported providers can be human or machine-driven. What is more, it is based on a plugin architecture that allows additional sources and translation services to be added by everyone. The currently available translators are the following:

- Microsoft Translator (machine)
- Google Translate (machine)
- Nativy (human)

- Gengo (human, machine)
- Supertext (human)
- One Hour Translation (human)
- Translations.com GlobalLink Connect (human, machine)
- SDL BeGlobal (human)
- GlobalSight (human)
- REST Translator (human)
- Euroscript Global Content Management X-Connect (human)
- Smartling Global Fluency (human, machine)
- LiveWords (human, machine)
- thebigword (human, machine)

One of the most notable new features are the icons to display the job status, source status and the translator. Consistent colors are used in order to maintain a clear understanding of the translation status; orange is for the state of "needs action", blue means "waiting" and green clearly symbolizes the "done" state.

The new global job item overview is another area with noticeable changes. Job items are filtered by the "Needs review" state because they are the ones that need users attention. The same applies for the job overview page, where only the open jobs are displayed. The translation progress is clearly marked with colors and state numbers and all extra information.

Another new addition is the connection to providers with the purpose to check credentials and update language mappings. This is very useful for remote machine translators, such as Google Translate and Microsoft Translator.

The ability to detect conflicts while translating with the "Show changes" option greatly improves the translation process. This helps users to revise the translated content and resolve any possible mistakes.

We already wrote about the inclusion of the WYSIWYG editor in Chapter 3. The preservation of text formats is the main benefit that content editors and translators get while using it. This means that the same content seen in the editor will be displayed on the page - nonetheless, WYSIWYG literally means "what you see is what you get".

Live preview functionality also greatly increases the user experience for the user by providing full awareness of the resulting appearance of the translating content before accepting the translation.

All those improvements make TMGMT a great translation tool, but there are more than 300 open issues on *drupal.org* that need to be worked on. Most importantly, new ideas and visions are being proposed by the community every day which makes the project evolve over time.

5 Problem

5.1 Problem Definition

The Translation Management Tool (TMGMT) module is an extension to support editors, publishers, translators and project managers during their process of content translation. It is currently well progressed for Drupal 8 with all previous features ported, the internal API was cleaned and many new features were added, as described in the previous Chapter 4.1. TMGMT can be seen as a very large initiative with multiple sub-projects. The Local Translator is definitely a very important and useful one. It allows a translation manager to delegate translation jobs to local Drupal users. The tool was working in Drupal 7, but it was rather limited in functionality and by far not fulfilling expectations of professional translators. To better support the professional audience, the XLIFF exchange feature was developed in order to support available external CAT tools. The next goal is to make the Local Translator a professional CAT Tool directly inside the Drupal UI. We are taking part of this initiative by working on implementing a UI for the translation editor (CKEditor in this case).

In the long run, TMGMT wants to use the CKEditor libraries to build an alternative non-WYSIWYG translation editor with special linguistic actions. We can think about the Google Translate service, for example, with many new features on top of it. The main features would be the following:

- Iterate through predefined segments
- Encode and decode HTML tags inside segments
- Get alternative suggestions from the translation memory
- Indicate translation quality of a segment
- Enable repositioning of encoded tags
- Mark active segments in both corresponding editors
- Mark translated segments as completed
- Display a counter of completed segments

- Warn the users of missing tags and segments

All the features and many more, would serve as a better translation experience, especially when translating longer chunks of text. Currently with TMGMT, if a content item changes in one field, it needs to be resubmitted as a whole to the translator. Without a translation memory, a translator is asked to perform the same translations all over again. If the user uses a non free translator this means that each time he would have to pay for it, leading to a very high maintenance cost of a multilingual site.

The translation memory will play a big role in this project, as it serves as a database, which stores "segments". These can be sentences, paragraphs or sentence-like units (headings, titles or elements in a list) that have previously been translated, in order to aid human translators. The translation memory stores the source text and its corresponding translation in language pairs called "translation units". Individual words are handled by terminology bases and are not within the domain of the TM. Research indicates that many companies producing multilingual documentation are using translation memory systems. In a survey of language professionals in 2006, 82.5% out of 874 replies confirmed the use of a TM [24]. The usage of TM is correlated with text type characterized by technical terms and simple sentence structure (technical, to a lesser degree marketing and financial), computing skills, and repetitiveness of content.

The translation memory programs generally break the source text (the text to be translated) into segments, look for matches between segments and the source half of previously translated source-target pairs stored in a translation memory, and present such matching pairs as translation candidates. The translator can accept a candidate, replace it with a fresh translation, or modify it to match the source. In the last two cases, the new or modified translation goes into the database.

Some translation memories systems search for perfect, 100% matches only. That is to say that they can only retrieve segments of text that match entries in the database exactly, while others employ fuzzy matching algorithms to retrieve similar segments, which are presented to the translator with differences flagged. It is important to note that typical translation memory systems only search for text in the source segment. The flexibility and robustness of the matching algorithm largely determine the performance of the translation memory, although for some applications the recall rate of exact matches can be high enough to justify the 100%-match approach.

In the case where no matching segments are found, they will have to be translated by the translator manually. These newly translated segments are stored in the database where they can be used for future translations as well as repetitions of that segment in the current text.

Translation memories work best on texts which are highly repetitive, such as technical manuals. They are also helpful for translating incremental changes in a previously

translated document, corresponding, for example, to minor changes in a new version of a user manual. Translation memories have not been considered appropriate for literary or creative texts, for the simple reason that there is so little repetition in the language used. However, others find them of value even for non-repetitive texts, because the database resources created have value for concordance searches to determine appropriate usage of terms, for quality assurance (no empty segments), and the simplification of the review process (source and target segment are always displayed together while translators have to work with two documents in a traditional review environment).

If a translation memory system is used consistently on appropriate texts over a period of time, it can save translators considerable work.

Translation memory managers are most suitable for translating technical documentation and documents containing specialized vocabularies, but might be also really useful on various websites.

Ensuring that the document is completely translated (translation memories do not accept empty target segments) and ensuring that the translated documents are consistent, including common definitions, phrasings and terminology are two of the main benefits of using translation memories. This is very important when different translators are working on a single project. Since translation memories "remember" previously translated material, translators have to do their job only once. This accelerates the overall translation process drastically and can lead to big cost reduction for long-term translation projects from the customer perspective.

The translation memory for TMGMT is still in an early phase and available as a sandbox project on *drupal.org*, but the core functionality works. In order to use it, we will provide the UI for the local translators as a separate module. We will therefore implement two new CKEditor plugins; one for the displaying the segments and another for displaying the tags. The separation is mandatory for a better visual representation of the two parts and for code maintenance.

6 Implementation

6.1 Implementation plan

For this project, we followed the Unified Software Development Process or Unified Process. The development lifecycle consists of the following four main phases:

1. Inception
2. Elaboration (milestone)
3. Construction (release)
4. Transition (final production release)

In the *inception phase* we established a business case for this project. We prepare a preliminary project schedule defined potential risks and problems. This is also where we established the project scope and boundary conditions.

In the *elaboration phase* we discussed the architecture, did a research of the field and came up with potential mockups for the new UI. We started studying the development process for CKEditor plugins and the general Drupal plugin development process. Before the coding process started, we created a Trello board [1] separated by weeks and with the following columns; to do, in progress, needs review, done and needs help/discussion. We also had different tags for different kinds of tickets, like weekly task, functional, cleanup, critical, UI, blog post and evaluation. This separation was great in order to focus on important tasks and have a visual representation of our work.

As the project was carefully planned, the *construction phase* consisted of 12 full weeks of coding. During this time, we wrote weekly reports of our progress in the form of blog posts (available on my personal website [2]) and had weekly meetings with our project mentors and Drupal representatives for the Google Summer of Code 2016 program. This methodology helped us stick with our initial plan and keep track of our progress.

The *transition phase* is out of scope for this paper, as we consider the production release to happen when the module will be fully functional, tested and merged into the TMGMT module.

[1] `https://trello.com/b/s74pBYi6`
[2] `http://sasanikolic.com`

6.2 Specification

The project specification consists of comprehensive guidelines regarding the product, organizations involved and the legislative part.

1. Product
   (a) Usability
      (i) The UI must follow the Drupal UI standards. [3]
      (ii) The UI must follow the Drupal interface patterns. [4]
      (iii) The UI must be simple, but the user might need some knowledge about Drupal.
   (b) Efficiency
      (i) The operations should function smoothly and in less than 1 second.
   (c) Portability
      (i) The module must work all modern browsers (except IE).
2. Organization
   (a) Standards
      (i) The module must follow the coding standards of Drupal. [5]
      (ii) The module must follow the "Best practices" of Drupal. [6]
3. External
   (a) Legislative
      (i) The module must be under the license GPL 2.0. [7]

6.3 Module structure

CKEditor is one of the best, if not the best, editor for website content. Drupal 8 in its basic version has built-in CKEditor with minimal basic functionality to edit the text. For most of the sites, such functionality is enough to edit website content, but very often, especially on large, sophisticated projects, in order to add new functionality, a user may require additional features of the editor. [2]

[3] `https://www.drupal.org/docs/develop/user-interface-standards`
[4] `https://www.drupal.org/node/1087090`
[5] `https://www.drupal.org/docs/develop/standards`
[6] `https://www.drupal.org/docs/develop/standards`
[7] `https://www.drupal.org/about/licensing`

To extend the functionality CKEditor uses plugins. For example, you may need to add Youtube video and display it on the page; or you want to have some extra functionality when importing images and various widgets, for all these purposes we are looking for a plugin and add it to the editor. Today there are more than 500 plugins on the official website of the editor. [8]

Drupal 8 offers a very user-friendly method to add a new plugin to CKEditor. Compared to version 7, with version 8 it is much easier to add new plugins. Now the site administrator has a more usable interface to work with the editor and its extensions. To add a new plugin, we need to create a new Drupal module.

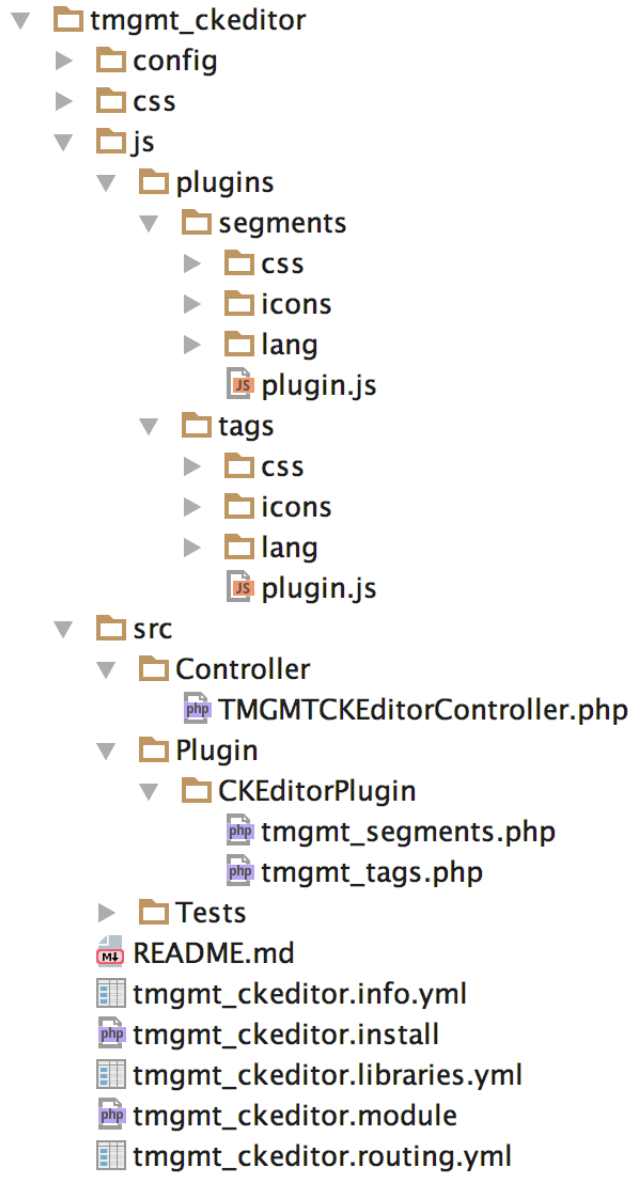

Figure 13: Every basic Drupal 8 module consists of .info.yml and .module files, all other files include extra features.

[8] `http://ckeditor.com/addons/plugins/all`

`tmgmt_ckeditor.info.yml` is responsible for the description of the module. It contains key-value pairs with the module name, its description and dependencies. In this case, our module is depending on *tmgmt_memory*, *tmgmt_demo* and *paragraphs_demo*. The later two are for testing purposes only.

The`tmgmt_ckeditor.install` file contains the code that is triggered when the module is installed (as the file extension suggests). Here we define the initial testing node named *Test for CKEditor plugins* with 5 different segments. The first segment is a normal sentence, while the others are defined to test inner tags functionality. The second node has two inner self-closing tags (`<br /> and <hr />`. The third one defines a properly closed, while the fourth segment has an inner tag that is not properly closed. The fifth segment contains a simple image tag. We also define some paragraphs and a few hardcoded translations that we therefore put in the translation memory when the module first loads. This helps us with keeping the development of the module constant - without the need to translate the content and save it in the memory manually.

`tmgmt_ckeditor.module` implements various hooks to alter the default behavior of the module. First, we define a hook to alter the CKEditor css, in order to tell Drupal where to find the new CSS files for our plugins. Most importantly, it contains the hooks to alter the default form behavior to allow the usage of a newly defined *translation_html* text format inside the editor. The main purpose of this is to have a better visual representation of a clean translation editor with only the two new plugins attached. Here, we are also altering the text output by masking the text and unmasking it after it has been saved. More about it in Section 6.3.2.

`js` is a folder for storing data about the plugin. In our case, the module actually contains our two plugins, named *segments* and *tags* respectively. This is the main directory, as it contains the main functionality of the plugins written in plain JavaScript programming language, as the folder name implies - *js* stands for JavaScript.

In order for Drupal to know that we have created these CKEditor plugins, it is necessary to create a new folder called *Plugin* inside the `src` folder and define them in separate files. Each module is a class that extends the *CKEditorPluginBase* and contains various functions:

- *getLibraries* describes additional libraries
- *getDependencies* indicates the dependence of this plugin
- *getFile* indicates the location of the plugin
- *getButtons* adds new buttons to the editor
- *getConfig* describes the configuration

6.3.1 Segments plugin

The CKEditor segments plugin defined in `js/plugins/segments/plugin.js` is where most of the module's functionality is implemented. It is written following strict coding convention in order to have a final product with good code readability and make the code maintenance easier for the community.

As the segmentation of the content is already done in the preprocess function, this is where we are implementing the defined features in Chapter 5.1 regarding the segments.

As the first step in the development process we wanted to iterate through the predefiend segments and display them cleanly in the editor when the translation page is loaded. The initial version is using user-defined `<tmgmt-segment>` tags in the source code. With them we define specific segments (e.g. divisions and paragraphs), that are then easier to translate. As by our goals, when the user clicks on the *'Display segments'* button in the CKEditor toolbar, a simple search is performed on our tags. They are then replaced by a pair of triangularly shaped icons, showing where the segments start and end. This enables the translator to distinguish the segments clearly and perform specific actions. The segments are toggled in both, source and translation editors for a better visual representation.

The next step was to mark the active segment, so that the user easily knows which part of text is being translated. We implemented this by making the segments icons orange, as this color is generally used in TMGMT for the active state. We believe it is also a color that quickly gets user's attention. The active segment is also displayed in the area below the editor for clean text separation.

As we also hard-coded some translation proposals for each segment that will end up being returned as a string from translation memory after the user clicks on a segment. In the area below the editor, apart from the active segment, we display a table with translation suggestions from the TM. Each of them has a specified quality and source. The quality is set in percentage, while the source can be of *human* or *machine*. Both values are passed from the TM through a HTTP request. More details about how that is done is explained in Chapter 6.3.3. By clicking the "Use suggestion" button, the user can confirm the suggestion is correct and replace the content of a segment with it.

After the translation is performed - either via the TM suggestion or manual translation, the user has the option to easily mark the segment as completed. This can be done by right clicking on a segment and selecting the *Set status completed* option. Again, this action triggers the change of a segment's indication. In this case, we display the segment with a green background.

We also added a new sticky area on the right side of the browser with the completed segments counter, so that the user always has an idea of how many segments were

already translated and how many translations are missing. This feature might increase the translation consistency and accuracy.

We believe that classes are pseudo application states and usually done for very simple indication so that you can use CSS for applying styles. In other words, since we are building a complex application with various states, we are using data-attributes to address them. We therefore defined data-attributes for the active state of the segment, its source, quality, and others. All of them should be preserved during the translation process.

Another important thing to mention, is that the plugin works for multiple editors on the same page. For example, if we are translating *Paragraphs*, which contain multiple fields - the segments plugin can be toggled in each of the editors and corresponding segments will be marked on both sides. This is a very useful feature to have, since the Paragraphs module is becoming more and more popular. To do this, a lot of refactoring was needed, which resulted in more than 300 lines of code changed. A new node with paragraphs was created, which contains segments with specific identificators. This was done in `tmgmt_ckeditor.install`, so that it happens when the module is enabled - for the purpose of making the usage and the development process of the plugin available right out of the box. After that, we needed an initial for loop over all source editors to populate the translation editors with data. This may be the only for loop in code that can exist. We therefore removed the for loops that were present when searching for same segments and mark them as active, since this is very time consuming and might result in bad performance when having many editors on page. We pair the editors according to their id and name (in regex: `idvalue-source-value$` and `id` × `value-translation-value$`. This change also affected the way that the areas below the editors are displayed and how the plugin works in general. We store all the data regarding the editor pairs in a JavaScript Object Prototype, such as:

- ID of the editor pair
- the selected editor name
- the DOM element of the area below the editor (in which we display selected segments, words and suggestions)
- the selected word
- the selected segment's id
- the counter of segments, that are marked as completed

6.3.2 Tags plugin

The idea behind masking the HTML tags is to get a clear understanding of the structure and display the opening/closing tags in the editor. It also helps with finding missing tags and their closures inside a segment, which can definitely improve the translation experience. As per HTML5 standards, there are three types of tags; *pair tags* - which consist of a start tag and end tag, with the content inserted in between, *empty elements* or *self-closing tags* - which are without content, and the image tag - which can be considered as a special case, since it can have translatable "alt" and "title" attributes.

We do not and will not support masked tag pairs. Instead, every masked tag will consist of two parts - their opening and closing element:

- element - the name of the masked HTML tag
- raw - contains the encoded tag, together with attributes

This is is our definition of the masked tags structure:

```
<tmgmt-tag element="br" raw="&lt;br /&gt;" />
<tmgmt-tag element="b" raw="&lt;b&gt;" /> ... <tmgmt-tag element="/b"
    raw="&lt;/b&gt;" />
<tmgmt-tag element="img" raw="&lt;img src=&quot;path&quot;
    alt=&quot;test&quot; title=&quot;Image title&quot; /&gt;" />
```

It is important that the editor displays the masked tags properly with their respective names. This is why we have the element attribute inside the tag. The raw attribute contains the encoded tag. In the case when the tag has some attributes like src, alt and title, we will easily get them from here, decode them, and display them to the user when the tag is clicked. The downside is, this makes these attributes untranslatable and they will be placed back untranslated. When unmasking, the process will simply replace `<tmgmt-tag>` with its raw property. Last, but not least, this structure helps us when looking for open and closed tags inside segments.

The workflow of masking the tags is a rather simple. Initially tags are masked with TMGMT when the editor loads. When the user clicks on a segment containing masked tags, we encode the masked tag before sending it through the HTTP request to the service controller. There we need to unmask the tag and query for the translation in the memory. If we have a match, we need to do the masking process again and return it to the frontend.

Because the segments in the memory are not masked, we need to properly invoke the mask and unmask function from the service controller. The problem here was that we cannot call alter hooks from the controller itself. As a quick solution, we provided them as simple external functions, which can then be called in the `hook_tmgmt_data_item_text_output_alter()` and `hook_tmgmt_data_item_text_input_alter()` hooks respectively to mask the tags on load and unmask on save, but also in the controller.

In order to maintain a consistent translation experience, the validation of missing masked tags is necessary. This consists of looping over every segment, displaying the counter of found missing tags in the translation editor based on the source editor, and the array of missing tags. Both, the counter and the exact missing tags are displayed in the area below the editor, as soon as a missing tag is detected.

This means, that we do the validation at many levels:

- when the editors are loaded
- when we add a suggestion from the memory
- when we detect that the content was changed with the `CKEDITOR.on('change')` event

The benefit of displaying the missing tags is crucial here, so that the translator gets warned about the possibility to have an incorrect DOM structure of the translated text. The user then has an option to click on the missing tag and place it in the cursor position of the active editor.

Based on the fact that languages structure varies a lot and there is no strict rule, that the tags should stay in the same position during the translation process, we need to be able to drag them around the content. For example, a bold text that is present at the beginning of a sentence in English, might be bolded at the end of the sentence in German. To do that, we converted the segment to be a *div element* and the tag to be an *inline widget* through the `CKEditor.dtd` object.

6.3.3 Communication with TMGMT Memory

Another important part of the module is the communication between our frontend part and the TMGMT Memory. This is done using simple HTTP requests. We define a route *tmgmt_ckeditor.get* in the `tmgmt_ckeditor.routing.yml` file and callback function for that route in the `TMGMTCKEditorController.php` file. Whenever a user clicks a segment, we are sending an HTTP request and check if the active segment is in the translation memory. To test our API responses we used Postman. This is an application that allows users to send POST/GET/PUT/DELETE etc. requests and see API

responses. Our first iteration consisted of a synchronous HTTP request. This possibility was deprecated because of its detrimental effects to the end user's experience. In fact, synchronous requests block the execution of the code and waits until we get a valid response from our server. This can create "freezing" on the screen and an unresponsive user experience. Therefore, we fixed the HTTP request to be asynchronous.

```
/**
 * Get the suggested translations from the tmgmt-memory.
 *
 * @param {Array} selectedContent
 *   Array of data about the selected segment.
 */
function getDataFromMemory(selectedContent) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function () {
      // Translations in the memory are found.
    if (xmlhttp.readyState === 4 && xmlhttp.status === 200) {
      displayActiveSegmentText();

      var jsonData = JSON.parse(xmlhttp.responseText);
      createTable(jsonData);
    }
    // No translations in the memory.
    else if (xmlhttp.readyState === 4 && xmlhttp.status === 204) {
      var noSuggestionsWrapper = document.createElement('div');
      noSuggestionsWrapper.className =
   constants.class.noSuggestedTranslationsWrapper;

  editorPairs[activeEditorId].areaBelow.appendChild(noSuggestionsWrapper);

      var text = document.createElement('P');
      text.className = constants.class.noSuggestedTranslations;

   text.appendChild(document.createTextNode(languageFile.noTranslationsInMemory));
      noSuggestionsWrapper.appendChild(text);
    }
  };
  sendHttpRequest(xmlhttp, selectedContent);
}
```

6.3.4 UI sections/areas

During the implementation process, we came up with many new ideas and features that we added to the initial mockups and design that we defined in the inception phase.

The main area is obviously the CKEditor itself. Here we defined two new plugins with specific icons in its toolbar. When toggling them, the segments and tags are displayed inside the text editor. The editor also contains a new context menu item for marking the segments as completed.

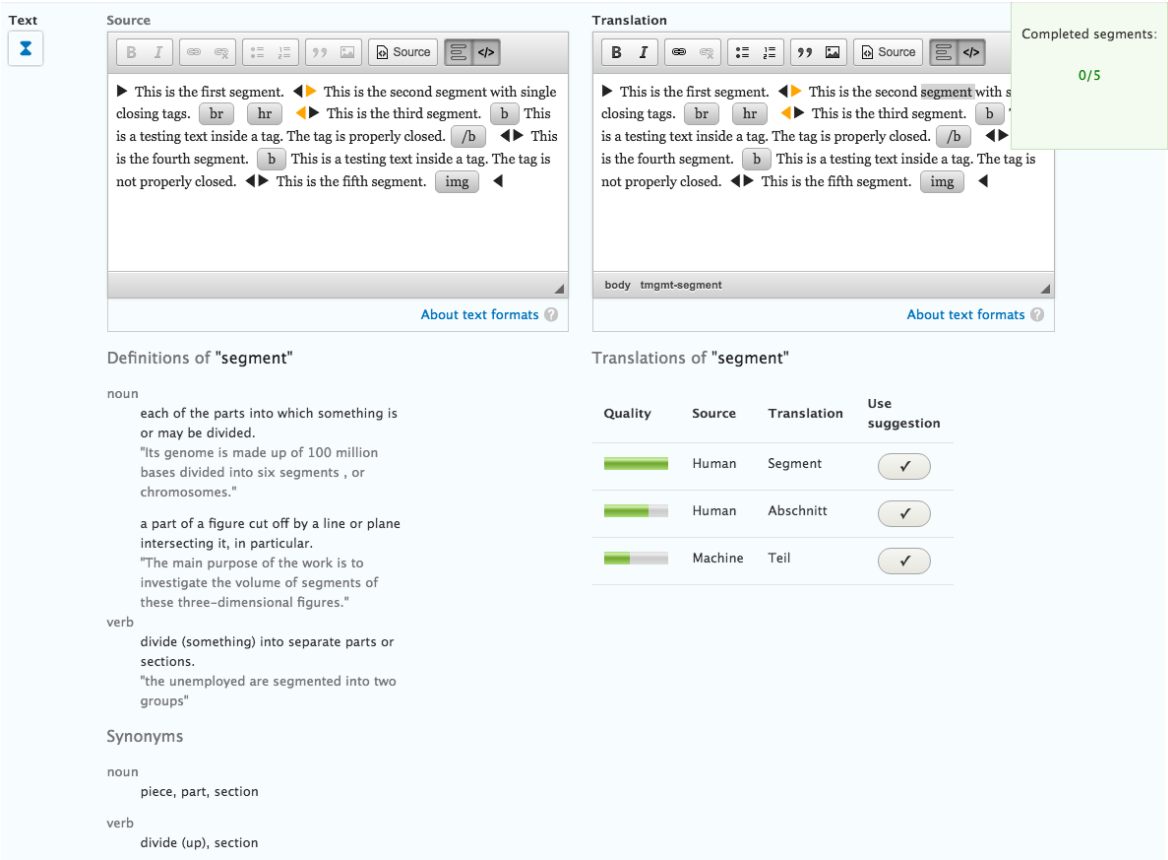

Figure 14: Full mockup of the UI improvements.

The ”sticky area” on the right side of the screen would hold some important information, such as the counter of the completed segments and the validation counters. This seemed logical, because they are both numerical counters and should be easy to spot at first sight. The background of this element would adapt to the tag validation. If all translations are valid, it has a green background. In case we have some invalid translations with missing tags, it is set to red. After some discussions about the initial idea, we agreed to have the validation part in the area below the editor. The reason for this is understandable, as the area below the editor holds the parts that the user can interact with - and the validation part that displays the missing tags is clickable, so that the user can place them in the editor at the cursor position. Note; we call it

the "sticky area" for a reason - its location persists when users scroll on the webpage, so that it remains always visible.

As already seen above, the area below the editor contains all the relevant information about the currently selected word and segment. Below the source editor we will be displaying the selected word definitions and synonyms in a later phase. Currently, we display just the segments suggestions on the right side in a table view.

We believe that the UI seen in the Figure 14 is not final yet and there are still a lot of things to be redefined and restyled. For the later, we still have an issue opened on drupal.org[9], where everyone is welcomed to show and discuss their suggestions.

[9] `https://www.drupal.org/node/2787689`

7 Results

We were working hard for roughly three months during the summer of 2016 on developing a rich and usable user experience for translators using the Translation Management Tool module in Drupal 8. The main goal was to create two CKEditor plugins. The first plugin converts text parts into segments to easily segment the content into smaller bits and enables easier translation management. It is also connected with the Translation Memory, to provide saved translation suggestions for requested segments. The second plugin is about masking HTML tags inside segments. This really helps to understand the structure better and cleanly show which opening/closing tags are missing inside a segment.

We believe the initial goal to have a prototype ready was clearly reached after $4-5$ weeks of work. After that, we started implementing more functionality around these two main "pillars". Users can now click through segments, which are clearly shown in both, the source and the translation editors, and check in the area below the editor for the translation suggestions. They can be used to replace the selected segment. Each segment can contain HTML tags, which are masked in the pre-process, before populating the content. They are displayed as widgets, so they can be clickable and draggable inside the editor area. If the translation is missing some tags, a warning message is displayed for awareness. When the translator thinks the translation is good enough, it can be marked as completed. This increases the counter of completed segments in the sticky division in the top right corner and colors the segment with a green background.

We created a shortlist of what is currently working as expected. Most of the following lines are crucial to the module, while others are extra features that we implemented along the way:

- dummy nodes for testing with new text format
- translation of nodes with one or more editor pairs (paragraphs)
- masking and unmasking the HTML tags
- displaying the active segment in both editors
- translation memory querying

- displaying suggested translations from the memory
- using suggested translations - placing the selected one in the active editor
- validation of missing tags (globally and per segment)
- adding of missing tag
- set segments as completed
- masked tag dragging

On the other hand, there are quite a few issues opened on the drupal.org projects page [1] which still need to be implemented, are in the state of "need feedback" or need more discussions:

- text segmentation (in tmgmt_memory)
- saving segmented content properly, so that accepting translation does not save segments but initial content
- responsiveness and mobile friendly
- better UI features and proposals
- make the area below the editor a class (prototype) with related methods

One important thing that we should be aware of: this project provides a number of puzzle pieces towards providing translators and reviewers better tools to work with TMGMT. Many other things are needed until this fully works together with real content.

7.1 How to test the module

In order to test the module, we are providing some basic guidance in the projects sandbox page. [2] A sandbox project is a module or theme that contains experimental code that is not yet ready for general use. Same information can be also found the module's `README` file. To use this module, the user needs to have a Drupal 8 website set up. Some beginner knowledge is needed in order to download the module and its dependencies. Once that is done, the remaining steps can be easily done through the Drupal interface. Installing the module is as easy as going to the *Extend* tab

[1] `https://www.drupal.org/project/issues/2737249`
[2] `https://www.drupal.org/sandbox/sasanikolic/2737249`

and selecting the *TMGMT CKEditor* module. As for now, only the predefined nodes from our module are working as expected - "Test for CKEditor plugins" and "Test for CKEditor plugins with paragraphs", because they contain the segmented content. After requesting a translation from a *Local translator*, which can be the user itself, all that is left to do is to go to the translation page. Any testing feedback is greatly appreciated.

8 Future work

The module is currently in an early alpha phase, but we are optimistic that we will develop more features in the near future and fix existing issues and many more described in Chapter 7.

Some of our future plans involve improving the Translation Memory module and extending it with the "fuzzy matching" functionality. Fuzzy matches are segments in the TM that are similar to the one being translated, but it does not match its translation 100%. If there is no exact match, but there are segments in the TM that are similar to the one being translated, then these are presented as fuzzy matches.

What we also want to focus on, is extending the TMGMT functionality by implementing the usage of the segments or separate words, their definitions, context and alternatives.

In order to eliminate uncertainty in the translation process, we would also implement a glossary. This can enforce consistency, shorten the time it takes to translate a document, and reduce the overall cost of translation over time. Glossaries are also referred to as a lexicon, term base, and terminology collection. It contains key terminology used on the site in a source language (typically English) and approved translations for that terminology in all other target languages. The glossary may also contain other metadata such as the definition, context, part of speech, and approval/review date. It is one of the key tools, along with a style guide and Translation Memory, to assure that all translated materials meet the defined quality requirements.

The project was also presented on Drupal Mountain Camp [1] in Davos in February, 2017, and in various Drupal meetings in Switzerland, where it gained a lot of interest among the community.

[1] `https://drupalmountaincamp.ch`

9 Povzetek

Sistemi za upravljanje spletnih vsebin je sistem, ki omogoča enostavno urejanje in vzdrževanje vsebine spletnih strani brez kakršnegakoli znanja spletnih tehnologij. Urednik spletne strani tako lahko samostojno spreminja besedila, slike in druge elemente spletne strani brez pomoči podjetja ali osebe, ki je stran izdelalo. Osveževanje spletne strani s CMS sistemom je zelo preprosto, podjetja in posamezniki pa želijo redno ažurirane strani, zato so CMS sistemi vedno bolj priljubljeni. V magistrski nalogi se v uvodu posvečamo kratki zgodovini le-teh ter njihove glavne lastnosti.

Med CMS sistemi so trenutno najbolj razširjene odprtokodne rešitve, kot npr. Wordpress, Joomla in Drupal. V nalogi se osredotočamo na slednjo rešitev. V 2. poglavju je na splošno predstavljen Drupal, v naslednjem poglavju pa opisujemo najnovejšo različico tega odprtokodnega sistema poimenovano Drupal 8 ter njegove glavne značilnosti in novosti.

Drupal 8 prinaša mnoge izboljšave na področju prevajanja spletnih vsebin. To je na predstavljeno v 4. poglavju. Ker pa mislimo, da uporabniška izkušnja in funkcionalnosti, ki jih Drupal 8 ponuja, niso dovolj na nivoju za nekatere uporabnike, v podpoglavju 4.1 predstavljamo modul *Translation Management Tool* kot alternativo za prevajanje spletnih vsebin. S to razširitvijo lahko zelo poenostavimo proces prevajanja že s tem, da prepustimo to delo profesionalnim prevajalcem. V 5. poglavju opisujemo problem, s katerim se prevajalci, ki uporabljajo Drupal kot ogrodje za prevajanje, srečujejo ter našo vizijo, kako bi ta proces poenostavili. Da bi olajšali delo "lokalnim prevajalcem", v 6. poglavju predstavljamo izboljšave na področju uporabniškega vmesnika za prevajanje s pomočjo razširitve TMGMT v Drupal. V 6. poglavju opisujemo specifikacijo ter dejansko implementacijo naše rešitve. V 7. poglavju predstavimo končni produkt dosedajšnjega dela, delujoče ter manjkajoče funkcionalnosti. Tu na kratko tudi opišemo, kako testirati našo rešitev. V zadnjem, 8. poglavju navedemo še nove ideje in predstavimo delo v prihodnje.

10 Literature

[1] 9 changes to be excited about in drupal 8. hhttps://dev.acquia.com/blog/9-changes-be-excited-about-drupal-8. Accessed: 2016-12-11. *(Cited on page 17.)*

[2] Aleksey razumov - creation of extended functionality for the basic ckeditor in drupal 8. https://medium.com/@CayugaSoft/creation-of-extended-functionality-for-the-basic-ckeditor-in-drupal-8-f9e6990aa48f. Accessed: 2017-25-02. *(Cited on page 40.)*

[3] Drupal 8.1.0 is now available. https://www.drupal.org/blog/drupal-8-1-0. Accessed: 2016-12-11. *(Cited on page 17.)*

[4] Drupal community. https://www.drupal.org/community. Accessed: 2016-12-10. *(Cited on page 8.)*

[5] Drupal core release cycle: major, minor, and patch releases. https://www.drupal.org/core/release-cycle-overview. Accessed: 2016-12-12. *(Cited on page 18.)*

[6] Drupal feature - security. https://drupal.com/feature/security. Accessed: 2016-12-12. *(Cited on page 7.)*

[7] Drupal issue queue. https://www.drupal.org/issue-queue. Accessed: 2016-12-10. *(Cited on page 9.)*

[8] Google summer of code 2016 information page. https://summerofcode.withgoogle.com/archive/https://summerofcode.withgoogle.com/archive. Accessed: 2016-12-08. *(Cited on page 4.)*

[9] Plugin api overview. https://www.drupal.org/docs/8/api/plugin-api/plugin-api-overview. Accessed: 2016-12-12. *(Cited on page 25.)*

[10] Saša nikolić - google summer of code 2016 blog. http://sasanikolic.com/gsoc-2016/gsoc/. Accessed: 2016-12-08. *(Cited on page 4.)*

[11] Saša nikolić - google summer of code 2016 project page. https://summerofcode.withgoogle.com/archive/2016/projects/6157951773442048/. Accessed: 2016-12-08. *(Cited on page 4.)*

[12] Translating configuration. https://www.drupal.org/docs/8/multilingual/translating-configuration. Accessed: 2016-12-15. *(Cited on page 29.)*

[13] Translation management tool. https://www.drupal.org/project/tmgmt. Accessed: 2016-12-16. *(Cited on pages 31 in 33.)*

[14] Understanding drupal. https://www.drupal.org/docs/8/understanding-drupal/overview. Accessed: 2016-12-11. *(Cited on page 8.)*

[15] Understanding drupal 8, part 1: the general structure of the framework. https://cipix.nl/understanding-drupal-8-part-1-general-structure-framework. Accessed: 2016-12-12. *(Cited on page 20.)*

[16] Understanding drupal 8, part 3: Routing. https://cipix.nl/understanding-drupal-8-part-3-routing. Accessed: 2016-12-12. *(Cited on page 22.)*

[17] N. Abbott and R. Jones. *Learning Drupal 8*. Packt Publishing, 2016. *(Cited on page 2.)*

[18] J. Barnett. *Drupal 8 for Absolute Beginners*. Apress, 2015. *(Cited on page 6.)*

[19] B. Boiko. *Content Management Bible*. Bible. Wiley, 2005. *(Cited on page 4.)*

[20] D. Buytaert. The future of decoupled drupal. http://buytaert.net/the-future-of-decoupled-drupal. Accessed: 2016-12-11. *(Cited on pages VII in 17.)*

[21] G. Hojtsy. Drupal 8 multilingual tidbits 2: more core modules. http://hojtsy.hu/blog/2013-jun-12/drupal-8-multilingual-tidbits-2-more-core-modules. Accessed: 2016-12-12. *(Cited on page 27.)*

[22] S. K.Patel, Dr.V.R.Rathod, and J. B. Prajapati. Article: Performance analysis of content management systems- joomla, drupal and wordpress. *International Journal of Computer Applications*, 21(4):39–43, May 2011. Full text available. *(Cited on page 4.)*

[23] R. Lagger. Major differences between drupal 7 and drupal 8 drupal. https://www.linkedin.com/pulse/major-differences-between-drupal-7-8-ryna-lagger. Accessed: 2016-12-11. *(Cited on page 12.)*

[24] E. Lagoudaki. *Translation memory systems: enlightening users' perspective; key findings of the TM survey 2006 carried out during July and August 2006*. na, 2006. *(Cited on page 37.)*

[25] J. Levi. 16 drupal 8 features you should know. `https://axelerant.com/drupal-8-features-need-know/`. Accessed: 2016-12-11. *(Cited on page 14.)*

[26] R. Nakano. *Web Content Management: A Collaborative Approach*. Addison-Wesley Professional, Sept. 2001. *(Cited on page 3.)*

[27] S. Nikolić and J. Šilc. Drupal 8 modules: Translation management tool and paragraphs. *Informatica*, 40(1), 2016. *(Cited on page 12.)*

[28] S. K. Patel, V. R. Rathod, and S. Parikh. Joomla, drupal and wordpress - a statistical comparison of open source cms. In *3rd International Conference on Trendz in Information Sciences Computing (TISC2011)*, pages 182–187, Dec 2011. *(Cited on page 4.)*

[29] D. D. Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt. The evolution of the grid. Technical report, University of Southampton, 2004. *(Cited on page 3.)*

[30] M. Tift. Configuration management in drupal 8: The key concepts8. `https://www.lullabot.com/articles/configuration-management-in-drupal-8-the-key-concepts`. Accessed: 2016-12-11. *(Cited on page 14.)*

[31] T. Tomlinson. *Beginning Drupal 8*. Apress, 2015. *(Cited on page 6.)*