Bachelor's thesis
Degree programme in Information Technology
2018

Veneta Stefanova

WEBSITE ARCHITECTURE
PLANNING, TECHNICAL
IMPLEMENTATION, AND
PERFORMANCE
IMPROVEMENTS

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES
The aim of this thesis project was to plan and develop a website for the commissioning company, ID BBN, whose main expertise consists of developing modern marketing solutions.

One of the purposes of this thesis was to solve the performance and content inconsistency issues of commissioner’s existing website. The website requirements set by the commissioner included precise technical implementation planning, code modularity, and friendly UI so that the employees would be able to produce and distribute content in an accessible way.

A content marketing-oriented Content Management System was used as a development platform where the website database was set up. The developed website layout was integrated together with the content management system to produce the desired outcome.

As a result, the newly developed website is modular, has significantly improved performance and follows the latest design trends. The content is consistent and follows similar visual and technical implementation patterns. All requirements set up by the commission were met through the course of this project.

The implementation of the new website has achieved layout consistency by following similar design patterns on each page. The usage of template files results in code modularity and hence, improved performance. The employees with no technical background can produce and distribute content in an accessible way by using the UI provided by OCM.

KEYWORDS:

website development, performance improvements, templates, CMS, front-end
CONTENTS

1 INTRODUCTION 6

2 CONTENT MANAGEMENT SYSTEMS IN CONTENT MARKETING 9
   2.1 Content Marketing 9
   2.2 Oracle Content Marketing (OCM) 9
      2.2.1 Publishers and domain names 10
      2.2.2 Content types, inbuilt and additional custom fields 10
      2.2.3 Static and dynamic contents and their usage on a template level 10

3 TECHNICAL IMPLEMENTATION 12
   3.1 Final URL structure and mapping domain name to publishers 12
   3.2 Templates and publishers planning and implementation 13
   3.3 Individual pages planning and setup 15
      3.3.1 Overall website and page structure 15
      3.3.2 “Main” page 15
      3.3.3 “Company/ about us” page 17
      3.3.4 “Our clients” page 18
      3.3.5 “Services” page 21
      3.3.6 “Events” page 22
      3.3.7 “Resources” page 27
      3.3.8 “Recruitment” page 28
      3.3.9 “Blogs” page 31
      3.3.10 “Contact” page 33
      3.3.11 Static sliding contact menu 34

4 CHALLENGES AND USED TOOLS 35
   4.1 Challenges 35
      4.1.1 OCM as a CMS 35
      4.1.2 Usage of API 35
   4.2 Tools 36
      4.2.1 Flexbox, CSS animations, and browsers’ support 36
      4.2.2 jQuery 36
      4.2.3 Testing 37
PICTURES

Picture 1. Lighthouse 2018, example of ID BBN’s current pages performance.
Picture 2. Implementation process.
Picture 3. Connections and dependencies between OCM’s components.
Picture 4. Dependencies between individual pages and components of the website.
Picture 5. Dependencies between pages, publishers and content types.
Picture 12. Video/image/quote widget code.
Picture 14. bxSlider initial functions calls.
Picture 24. Performance comparison between existing website’s pages and the newly-developed pages.
Picture 25. Partial UI derived from "Events” content type.

TABLES

Table 1. Reasons behind full website renewal.
Table 2. Technical requirements given by the commissioner.
Table 3. Final website URL structure.
Table 4. Content type "Events" fields.
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>BBN</td>
<td>Business-to-Business Network</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>CTA</td>
<td>Call-to-Action</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>GDPR</td>
<td>General Data Protection Regulation</td>
</tr>
<tr>
<td>ID BBN</td>
<td>The company’s name</td>
</tr>
<tr>
<td>jQuery</td>
<td>a JavaScript library</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>OCM</td>
<td>Oracle Content Marketing</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP: Hypertext Preprocessor</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UX</td>
<td>User Experience</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

The commissioning company, ID BBN, is a Finnish marketing automation agency situated both in Turku and in Helsinki. The company takes pride in providing high-level marketing and technical expertise, which has lead to being recognized not only in the Finnish marketing business but also internationally. The company is a mixture of creative and technical specialists (ID BBN – Modern Marketing, 2018).

Table 1. Reasons behind full website renewal.

<table>
<thead>
<tr>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The current existing website is hosted in different CMS.</td>
</tr>
<tr>
<td>2. The website is built with different web development technologies.</td>
</tr>
<tr>
<td>3. There are existing pieces of redundant code left from previous version changes.</td>
</tr>
<tr>
<td>4. The website code lacks modularity.</td>
</tr>
<tr>
<td>5. Website is hard to maintain in both technical and visual aspects.</td>
</tr>
</tbody>
</table>

ID BBN’s website follows the latest design trends. However, it has come to the commissioner’s attention that the website performance has reduced (Picture 1.), as being one of the reasons (Table 1.) for the overall website renewal.

Picture 1. Lighthouse 2018, example of ID BBN's current pages performance.
The subject of this thesis is the planning and the new implementation of the already existing website of ID BBN. The thesis follows through the process of planning the website architecture and its implementation (Picture 2.).

![Diagram](image)

**Picture 2. Implementation process.**

The implementation process follows the specific technical requirements given by the commissioner (Table 2.). The requirements per each individual page vary, as well as their visuals. As a result, a completely new technical implementation is required with precise planning.

**Table 2. Technical requirements given by the commissioner.**

<table>
<thead>
<tr>
<th>Technical requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. All content pages and assets located in one CMS.</td>
</tr>
<tr>
<td>2. Improved overall performance.</td>
</tr>
<tr>
<td>3. Achieved code modularity.</td>
</tr>
<tr>
<td>4. A non-technical team has the ability to create and distribute content in an accessible way.</td>
</tr>
</tbody>
</table>

The usage of Content Marketing as a strategy in the commissioning company has an influence on the choice of CMS, such as Oracle Content Marketing (OCM) to be used for the development of this project. The final result aims to be a modern, modular, easy-to-undergo-modifications website with significantly improved performance.
In this thesis, Chapters 2 introduces a short theoretical background about the subject and the CMS used. Chapter 3 follows the step-by-step implementation of each component of the website. Chapter 4 discusses the challenges and how they were overcome. Chapter 5 includes final thoughts and a summary of the outcomes of the project.
2 CONTENT MANAGEMENT SYSTEMS IN CONTENT MARKETING

Marketing representatives need to follow the latest trends and distribute relevant marketing content accordingly to the target audience. That is achieved by using Content Management Systems. A choice of an unsuitable Content Management System (CMS) could result in not meeting this goal and causing content inconsistency in the long run (Sukhraj, 2018).

2.1 Content Marketing

Content Marketing is a strategy which aims to distribute relevant content to a specific target audience. It is a core approach which allows companies to attract leads and promote their services (Content Marketing Institute, 2018). It is essentially a “storytelling” which aims to keep the user’s attention for a prolonged period of time (Patel, 2018).

2.2 Oracle Content Marketing (OCM)

Oracle Content Marketing is a CMS designed to make the distribution of marketing content easy for every organization using the platform. It uses profiles to target a specific group of people and contributes to providing exceptional customer experience by showing relevant content. The platform is built on the 5 principles of Content Marketing (Oracle Content Marketing - User Guide, 2018, pp.13-14), which results in creating the right content for the right target group of people.

OCM gives the opportunity to build a website by using templates, static and dynamic contents. OCM uses publishers which are connected to domains on the server side. Publishers are mapped to specific content types (Picture 3.) which makes it easy to manage and present contents to the user via templates on the back-end of the platform (Oracle Content Marketing - User Guide, 2018). OCM’s template content hub uses Twig as a template engine. Specific syntax (Twig.symfony.com, 2018) coming from Twig is taken into consideration during development. Syntax errors can result in the complete unavailability of the web pages and showing error messages in the browser.
2.2.1 Publishers and domain names

In OCM the publisher is a service that is used to publish and distribute content (Oracle Content Marketing - User Guide, 2018, pp.106-107). The publishers are mapped to a development domain name owned by the commission (Picture 3.). They can be connected to specific content types in order to distribute content (Oracle Content Marketing - User Guide, 2018).

2.2.2 Content types, inbuilt and additional custom fields

Content types help to organize the content. They can be added according to the needs of the users (Oracle Content Marketing - User Guide, 2018, pp.85-86). OCM provides basic inbuilt input fields for content types, such as “Rich Text Editor”, “Title”, and “Featured Image”. Custom created fields can additionally added to meet the specific needs of the user (Picture 3.).

2.2.3 Static and dynamic contents and their usage on a template level

Both static and dynamic contents can be created in OCM. Static contents are delivered to the user without being generated or modified. The same static files are delivered to all users on a request. They are not personalized and they do not undergo changes unless

Picture 3. Connections and dependencies between OCM’s components.
modifications are done manually. The static files are not generated per-request and that makes them efficient and fast (Maxcdn.com, 2018). Dynamic contents are generated on a request by the user. In OCM they are personalized and derived from the content types. The server loads them according to the user’s request. Dynamic contents are flexible to changes due to their identical structure. Once the request is sent to the server, a server-side programming language such as PHP builds an HTML page and delivers it to the user (Newman, 2018). The server delivers static and dynamic content rendered in a template hub.

Templates are HTML pages which have specific markup into their syntax (Web.archive.org, 2018). Web template systems are used in Content Management Systems. Templates operate by reusing static components for each page, while the dynamic components are generated by the user’s request (Newman, 2018). In OCM the template hub is a folder full of files which are interacting with each other. There are main files which include other smaller files. All files are rendered according to specific programming logic which results in efficiency and modularity.
3 TECHNICAL IMPLEMENTATION

A website map has been created to visualize the planned implementation process (Picture 4.).

![Website map]

Picture 4. Dependencies between individual pages and components of the website.

3.1 Final URL structure and mapping domain name to publishers

Existing domain name is linked to publishers. The publishers are connected to a template file in order to render content in the URLs. The URL structure (Table 3.) is defined to include both Finnish and English language versions.

Table 3. Final website URL structure.

<table>
<thead>
<tr>
<th>Finnish language version</th>
<th>English language version</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.idbbn.fi">www.idbbn.fi</a></td>
<td><a href="http://www.idbbn.com">www.idbbn.com</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/yritys">www.idbbn.fi/yritys</a></td>
<td><a href="http://www.idbbn.com/company">www.idbbn.com/company</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/palvelut">www.idbbn.fi/palvelut</a></td>
<td><a href="http://www.idbbn.com/services">www.idbbn.com/services</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/tietopankki">www.idbbn.fi/tietopankki</a></td>
<td><a href="http://www.idbbn.com/resources">www.idbbn.com/resources</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/tapahtumat">www.idbbn.fi/tapahtumat</a></td>
<td><a href="http://www.idbbn.com/blog">www.idbbn.com/blog</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/blogi">www.idbbn.fi/blogi</a></td>
<td><a href="http://www.idbbn.com/events">www.idbbn.com/events</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/rekry">www.idbbn.fi/rekry</a></td>
<td><a href="http://www.idbbn.com/careers">www.idbbn.com/careers</a></td>
</tr>
<tr>
<td><a href="http://www.idbbn.fi/yhteys">www.idbbn.fi/yhteys</a></td>
<td><a href="http://www.idbbn.com/contact">www.idbbn.com/contact</a></td>
</tr>
</tbody>
</table>
3.2 Templates and publishers planning and implementation

Publishers are used to render content in the template content hub. Templates give the opportunity to reuse existing files in other files. Applying changes to globally shared components requires a change to a single file and makes the code modular (Table 2, requirement 3). Modularity is a key concept when using templates. It gives flexibility and a sense of better control and organization of the code structure.

The website uses only one template content hub, consisting of big and small template files, CSS files and widget files. These files serve the purpose of rendering files according to publishers. Content rendering is done on a code-level by checking the publisher-specific id variable provided by the system which corresponds to a string of numbers. Each web page has a corresponding publisher connected to URL and a template file (Picture 5.). The template file holds the content being rendered as HTML in the URL. The template file for each individual page is a large file filled in with other small imported files called components. Components are widget files, template files, and CSS files.
Picture 5. Dependencies between pages, publishers and content types.
Content types are defined for each dynamic component that uses data from CMS’ database. Widgets are used to hold specific logic of rendering contents from OCM’s database. For example, if the goal is to show a specific number of blog posts by category, widgets code is rendered into the template files.

3.3 Individual pages planning and setup

In the following sub-sections, the thesis follows the technical implementation of each individual web page by visualizing technical requirements, followed by an explanation of the implementation approach.

3.3.1 Overall website and page structure

Pages share common logical implementation and structure. The visuals show (Picture 6., Picture 9.) modern and plain layout and the majority of the content sections are box-shaped content blocks. Animations are embedded on top of the content blocks to create smooth transitions.

3.3.2 “Main” page

When developing a non-static website, the developer writes the implementation logic initially on a “dummy” page, meaning -to build the required functionalities, and in a later stage to build the layout with front-end development tools. The “dummy” page provides a playground for implementing functionalities’ logic and excludes the possibility of visually damaging the layout. The implementation of this thesis subject follows that development approach.

The main page holds two rows of dynamic sections Row 2 and Row 4 (Picture 6.). Row 2 consists of information regarding the commission’s latest marketing strategies, Row 4 renders posts by specific category from the “Blogs” page. A content type “The latest news” is created to show dynamically the latest news of the company. The content type consists of a “Title”, “Short description”, “CTA text”, and a “CTA link” fields and they are rendered on Row 2. OCM’s UI provides a variety of custom fields which are set up so
users who lack technical knowledge can edit content easily (Table 2, requirement 4.). These system field names are used to render content into the template files.

![Image](image1.png)


Widgets are used to show the contents generated from a specific content type. Retrieving information from the “The latest news” content type is done by looping through the content type’s database (Picture 7.).

![Image](image2.png)


Rendering blog posts from ID BBN's Blog is already possible because the “Blogs” page had been implemented previously in OCM. The widget “Blog posts by category(Main page FI)” holds the logic of rendering blog posts per specific category from the Blog (Picture 8.).
The previously existing main page of ID BBN had an API fetching the blog posts from OCM. The page being rebuilt in OCM makes the use of API redundant - the contents are coming directly from OCM, which results in a better overall performance (Table 2, requirement 2).

3.3.3 “Company/ about us” page

The “Company/ about us” page is static, consists of information about the company.

The page starts with Row 1 (Picture 9.) having a hero image and a text, holding information regarding the company. Row 2 has cover background images. Addresses and maps are added into boxes to the two locations where the company resides. The information is shown to the user once a hover effect is triggered.

Row 3 (Picture 9.) holds information about the variety of the additional services which the company provides. The content is built with a jQuery slider plugin -bxSlider (Bxslider.com, 2018).
3.3.4 “Our clients” page

The “Our clients” page consists of short descriptions of clients’ business cases implementations. The cases are listed under specific categories which the user can choose. The cases are visualized in boxes with visible titles and toggle smooth hover effect which shows or hides additional short description and a CTA (Picture 10.). The CTA leads to more information regarding the individual case (Picture 11.). A content type “Cases” is created to achieve the setup.
The cases are filtered by category and they are loaded with a “Load more” button. The loading functionality is achieved with AJAX and API because the particular additional loading uses asynchronous methods and creates a request to the server on every click - fetches the requested N-amount of cases and shows them to the user.

The implementation of AJAX calls is done on the front-end. The API endpoint is created through a reversed proxy because API access tokens are publicly available.

The user is redirected to a subpage when a CTA is clicked. The subpage is generated by OCM and displays additional information for the requested content (Picture 11.).
The content type “Cases” holds three groups (Picture 11, Row 2-4) of fields which are built on the concept of “challenge-solution-result”. The page is meant to be used by people with no technical knowledge in an accessible and efficient way through OCM’s UI (Table 2, requirement 4.). The content type consists of three groups of two inputs - description and visual representations. Each pair of fields is similar. The difference between them is that each group is used for either problem, challenge or solution content rendering (Picture 11, Row 2-4). All fields are used to render content in the individual case subpage (Picture 11.).

The text rendering is done by rendering content from specific text fields. A different implementation approach is used in rendering an image, a video or a quote. On the back-end in a widget code, for video or image, an if-else statement checks if the URL contains specific string based on domain name. Depending on the result it renders accordingly either an image, a Youtube embedded iframe or a plain text (Picture 12.).
3.3.5 “Services” page

The “Services” page is static and it is implemented by using the bxSlider (Picture 13.).

Two different functions are used to initialize the slider. The first function call (Picture 14.) initializes the desktop slider, the latter is to set up the mobile sliders. All function parameters are markup defined in the official documentation of bxSlider (Bxslider.com, 2018).
The layout for the mobile uses five small sliders instead of one to achieve the desired functionality. They are implemented in a separate code section. The browser switches the layout from a single slider to five individual ones by checking the screen size using CSS media queries.

3.3.6 “Events” page

The “Events” page consists of information about the events hosted by the company. The page holds both static and dynamic contents (Picture 15.).

Row 2 (Picture 15.) consists of static content. The boxes hold information for past or upcoming events. They have a brief description and a CTA that leads to external landing pages with information for the particular event.
The content on Row 3 (Picture 15.) is dynamic and it is derived from OCM’s content type. A content type “Events” is created and holds additional fields (Table 4.).
Table 4. Content type "Events" fields.

<table>
<thead>
<tr>
<th>Group name</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>1. Event title&lt;br&gt;2. Event image&lt;br&gt;3. Time and date&lt;br&gt;4. CTA to the particular event page&lt;br&gt;5. Location&lt;br&gt;6. Description&lt;br&gt;7. Agenda description&lt;br&gt;8. Event timetable</td>
</tr>
<tr>
<td>Keynote</td>
<td>1. Title&lt;br&gt;2. Description</td>
</tr>
<tr>
<td>Agenda</td>
<td>1. Agenda</td>
</tr>
<tr>
<td>Speakers</td>
<td>1. 1 field for each individual speaker’s name&lt;br&gt;2. 1 field for each individual speaker’s photo</td>
</tr>
<tr>
<td>Live stream video</td>
<td>1. Video link&lt;br&gt;2. CTA to Outlook event</td>
</tr>
<tr>
<td>After event</td>
<td>1. After-event PDF material 1&lt;br&gt;2. After-event PDF material 2</td>
</tr>
<tr>
<td>Upcoming event</td>
<td>1. Title&lt;br&gt;2. Description&lt;br&gt;3. CTA to the upcoming event</td>
</tr>
<tr>
<td>Concept</td>
<td>1. Title&lt;br&gt;2. Description</td>
</tr>
</tbody>
</table>

All fields under content type “Events” are used in different ways and pages. On Row 3 (Picture 15.) particular “Event” fields are used (Table 4, Event, Fields 1,3,4,5) for displaying information. Event -Field 4 (Table 4.) controls where the user is redirected to. OCM checks on a code-level whether the particular field is empty or not. If it is empty, the CTA links to the subpages generated by OCM. If it is not empty, it redirects to an external page provided from the OCM’s UI.
Row 3 (Picture 15.) is implemented by filtering all existing events created under “Events” content type and ordering them by a date - the nearest upcoming event comes first on the list. An additional checking is done to disable the visibility of already past events.

Row 4 (Picture 15.) holds six boxes of blog posts. A similar approach is used in “Main” page - using a widget that filters the blog posts by the desired category and displays them to the user. Programming logic is being reused (Picture 6, Row 4) and adjusted to meet the requirements in the particular case.

Row 5 (Picture 15.) is a section with Instagram posts. The user can get acquainted with photos from the events. The functionality is implemented by using Instagram API and AJAX - it shows the first four posts under a specific hashtag.

The defined fields make additional content rendering possible. In Row 2 (Picture 16.) are rendered fields for event’s title and description, in Row 3 - the “Keynote” fields (Table 4, Keynote). These fields are rendered texts from the database.

![Image of webpage layout](image.png)

Row 4 (Picture 16.) holds information about the speakers of the event. The code is built to display the correct number of speakers in a user-friendly manner. The minimum amount of speakers is one and the maximum is six.

Row 5 (Picture 16.) displays the agenda for the event and rich-text editor field “Agenda” (Table 4.) is used to render the information.

The sign-up form on Row 6 (Picture 16.) is generated with an additional JavaScript code on the front-end due to OCM not providing the functionality to build HTML forms. JavaScript picks up the data from rendered form fields (Table 4, Sign-up form) and generates a modular form. The fields are rendered via Twig and hidden in the DOM. A custom generated event form is displayed to the user. The form action submits to Eloqua where it can be followed who have signed up for the particular event.

Row 7 (Picture 16.) consists of live stream video. Live stream fields (Table 4, Live stream video) are used to show the content regarding the particular stream. “Video link” is a text field where a video link is pasted. A checking in the back-end is done -if the field is not empty and consists of a Youtube live stream link, it creates a styled iframe on the front-end. If it is empty, it hides the section.

Live stream section (Picture 16, Row 7) gives the user the opportunity to save the event’s information to one’s calendar. The “CTA to Outlook event” (Table 4.) consists of a link to an Outlook calendar file that holds the event’s information. The user can save the link locally to one’s computer by clicking the CTA and add the event to Outlook.

To the participants are delivered particular materials once the event has passed. The after-event sections (Picture 16, Row 8-9) are included in the layout but they substitute the initial sign-up page once the event has passed. Row 8 provides materials from the event. The fields from “After event” (Table 4.) are used to display the materials to the user.

Row 9 (Picture 16.) holds information about the upcoming events. The content is displayed by rendering the fields under “Upcoming event” field group (Table 4.).

The event’s creator can show or hide sections by using checkboxes in OCM’s UI. Only relevant information is shown to the user in the different time frames of the event -before, during and after-event. OCM provides the developer with the ability to integrate checkboxes to achieve the setup. The checkboxes help a non-technical team to create and modify content in an accessible way (Table 2, requirement 4).
3.3.7 “Resources” page

The “Resources” page holds resource files in PDF format which are available to the users who visit the website.

The layout (Picture 17.) has similar design patterns to the layout of “Our clients” page (Picture 10.). The “Our clients” page code and styles are being reused and integrated to work for the particular page. For the development of the page, a new content type “Resources” is created. It holds the “Title”, a “Description” and a “Link” of the materials. The resources are filtered by the categories assigned by the creators of contents. Initially, the page shows N-amount of posts but the user can load more materials asynchronously by clicking the “Load more” button.

![Picture 17. Selander, T 2018, “Resources page” layout, Turku.](image)

The resources are PDF files which can be downloaded on a CTA click. To the user appears a pop-up on the screen requesting an email address to allow downloading the particular materials. A single form is being reused for all files. The form holds fields for an email address and additional hidden fields for tracking of the downloaded content.
3.3.8 “Recruitment” page

“Recruitment” page holds information regarding the working environment, the open job positions and the benefits of becoming an employee of ID BBN.

![Recruitment page layout](image)


The working environment information is introduced in static boxes with hover effect which shows a brief description and a CTA (Picture 18, Row 2). A CTA leads to recruitment’s subpage (Picture 19.) which holds detailed information about the atmosphere of work.

Row 4 (Picture 18.) presents the open job positions. The open positions boxes are derived from content type “Recruitment” but they do not redirect to a subpage. The content type holds fields “Title”, “Image” and a “Job description”, derived from a text
editor. The content type is mapped to “Recruitment FI” publisher to show the needed HTML code to the corresponding URL. The title scales-in and becomes clickable on a hover effect. A clicked title shows the specific job description below and animation scrolls down to the beginning of Row 5.

There are two for-each loops on the back-end which print the job descriptions (Picture 18, Row 5) in a different way. The first loop renders the job title and the id of the content as a CSS class (Picture 18, Row 4). The latter loop renders the job description and the same id as a CSS class (Picture 18, Row 5). On the front-end JavaScript checks whether the clicked job position has a corresponding class name with the job descriptions section. If there is a match - a smooth fade-in effect is triggered and scroll animation leads the user to the particular description.

In Row 5 (Picture 18.) is also situated a static contact form. The form holds basic contact information about the applicant, additional social media non-mandatory fields and its back-end handling is in Eloqua. An additional form input validation is implemented because Eloqua does not support functionality for validating free user input text fields.

The “Recruitment” page has a subpage (Picture 19.). According to one (Table 2, requirement 4) of technical requirements, the subpage should be easily editable by a non-technical person. The content type “Recruitment” is already used on the main “Recruitment” page, hence, it is not possible to use the same content type to derive the desired page. Instead, a new additional “Employer Branding” content type is added.

The subpage structure is identical to “Our clients” subpage but with an additional widget to filter blog posts per given category (Picture 19, Row 7).
The layout (Picture 19.) follows a similar design to the individual subpage of “Our clients” page (Picture 11.). Code structures (Picture 19, Row 2-5) are reused and integrated to work with the particular content type.
3.3.9 “Blogs” page

The “Blogs” page already exists in OCM as a result of a previous project. The new page is redesigned (Picture 20.) to follow similar layout patterns and overall structure of the website.

![Blog page layout](image)


The way the content is displayed to the user is the main difference between the existing “Blogs” page and the newly implemented page. The existing page renders content directly from OCM by using widgets. The newly created “Blogs” page content rendering is dependable on API.

On an initial page load, the content is presented to the user as a pyramidal structure (Picture 20.). The posts are displayed to the user differently according to their array index in the API response. On each page load, on Row 2 the first post returned from the API is highlighted and more information about it is shown. The post lacks a hover effect. Row 3 holds two blocks and Row 4 holds three blocks, both rows having a hover effect. When triggering the hover effect, part of the description of the blog and a “Read more” CTA appear, which initially are not visible to the user.
The user can load more posts if one is interested. The functionality is implemented with an API and AJAX. For every call to the server, the page shows the next three posts in the array in a row of three boxes similarly to Row 4 (Picture 20.). If the person selects a particular category, the rendered content is shown in three blog posts per row. The dropdown menu and the layout structure are an identifier that one is navigating through content under a specific category.

The user can read further a particular blog post by clicking on the corresponding CTA. The CTA redirects to the subpage derived from the content type “Blog post”. The structure of the individual blog post page holds a full-width header image and the blog post’s content displayed in a column (Picture 21.).

3.3.10 “Contact” page

The “Contact” page is static. It holds the contact information and introduces part of the people who work in high positions in the company – the Managing Director, the Vice Presidents and the Customer Leads (Picture 22.).

![Contact page layout](image)


The page displays the names, images, and links to different means of contact. Filtering by “Location” and “Title” is presented by using dropdown menus. The employees’ information is stored in a JavaScript object. People’s information is rendered in boxes (Picture 22.) by looping through the object.
To implement filtering by “Title” a title is defined for each person as CSS classes - managingDirector, vicePresident, customerLead.

The same approach is used for “Location” - Helsinki, Turku. The CSS classes are captured with JavaScript. When the user clicks on a particular "Title", the JavaScript gets the string, compares it to the defined CSS classes and shows only the ones which match the criteria. The same logic is present for filtering by “Location”. To use both filtering dropdown menus simultaneously, an additional JavaScript variable is created to store the data of the previously selected filter. It creates a comparison between the selected values of “Location” and “Title”. As a result, it displays the people matching the criteria.

3.3.11 Static sliding contact menu

The contact menu layout feature is implemented to all pages. Once clicked, it slides to the left and opens a three column content (Picture 23.).

![Static contact menu layout](image)


In the first column is placed a form which requires an email address to subscribe for the monthly newsletter of ID BBN. Above the form is a placeholder text which will be replaced with information regarding GDPR.

A contact form is located in the second column with required input fields. It is a static form which the user can fill in.

A chat functionality is implemented in the third column - a third-party service implemented by a plugin, integrated between the UI and ID BBN's recipient’s email address.
4 CHALLENGES AND USED TOOLS

Obstacles were present due to the used tools through the implementation. Some of them derived from the fact that specific tools are used, others from the fact that particular tools were not used. This section follows up more detailed explanation what came into use and the benefits and drawbacks of each.

4.1 Challenges

4.1.1 OCM as a CMS

OCM has its own limitations which made particular implementation parts challenging. An example of such a limitation is the “Employer Branding” subpage of “Recruitment” page which needed an additional content type to be created in order non-technical team to easily modify contents.

Other challenges that OCM has introduced is the lack of inbuilt form functionalities and that it generates only custom text fields. A back-end code checking was created to filter these fields’ values. The values were captured with JavaScript to present the content to the user without damaging the layout and causing an unpleasant user experience.

Template-level limitations were present due to the only file’s formats possible to be added are “.tpl” and “.css”. OCM is built on top of PHP and Twig templating engine but it does not support importing custom written PHP functions in the template which could have significantly speed the implementation process.

4.1.2 Usage of API

A reason behind using CMS for the implementation of the project was that it provides a database and the information can be retrieved directly from it. On the other hand, an API is used across particular pages as a different way of retrieving data. Both ways of rendering content have their benefits and drawbacks.
On one hand, retrieving content straight from OCM results in fast performance due to requesting directly to the database. On the other hand, using the inbuilt functionality given by OCM does not provide a modern UX.

While API gives the benefits of asynchronous loading of content and gives the user an impression of modern design, the requests to the server can be significantly time-consuming. Additionally, the server can be down and the page becomes fully-dysfunctional.

4.2 Tools

4.2.1 Flexbox, CSS animations, and browsers’ support

Flexbox and CSS animations were used for the implementation due to their inbuilt functionalities.

Flexbox was used to achieve a layout consisting of boxes. It provides an easy way to create flexible boxed-shaped designs by using the inbuilt features. It allows equal sized boxes and equal spacing between them, as well as reordering them in the desired way (CSS-Tricks, 2018). In particular cases, Flexbox boxes were a challenge to adjust visually due to the uncertain amount of the content in them (Picture 17, Picture 20). As a result, the Flexbox spacing between boxes was overwritten by plain CSS in order to achieve the desired layout.

CSS animations implemented together with Flexbox resulted in providing modern hover effects and sliding animations.

Making the website compatible across all OS, browsers, and devices, such as boxes size, content size, and animations behavior was a challenge. Internet Explorer and Edge required more testing than other browsers due to their partial non-support of Flexbox (Caniuse, 2018).

4.2.2 jQuery

The main library used for implementing the website is jQuery. Although its motto is “Write less, do more.” (js.foundation, 2018) it has its drawbacks. jQuery is lacking the
functionalities to support particular behavior defined in the pages’ layouts. It lacks the ability to save a state and to keep track of the constant changes in the UI, resulting in some filtering options being excluded from the layout.

Modern frameworks such as React.js would have made possible the initially designed features to be implemented due to providing state (Reactjs.org, 2018). ES6 JavaScript gives more flexibility and implementation options than jQuery. It has inbuilt arrow functions and array methods (Reactpwa.com, 2018). During this project, it was not possible to be used because modern JavaScript frameworks can not be used together with OCM.

4.2.3 Testing

The testing tools used through the project have been the browsers’ Developer Tools and Lighthouse -Google Chrome extension.

The Developer Tools provides options for testing such as simulating devices screen views, “Console” for logging API data responses, and “Network” -following the browser’s requests.

Lighthouse extension gave better test results for the overall performance and best practices of web pages. It provides audits for Performance, Accessibility, Progressive Web Apps, Best Practices and SEO (Google Developers, 2018). Lighthouse has been beneficial in the scope of this project due to the fact that it generates a detailed report according to the audits. It provides solution examples and further reading material in order to improve the results of the audit.

“Performance” audit checks the code loading speed and efficiency (Google Developers, Lighthouse, 2018). In the scope of this project, the audit for “Progressive Web App” is not taken into account, due to not being part of the technical requirements (Table 2.). The “Accessibility” audit concerns the UI -page contrast and HTML markup that could improve the UX (Google Developers, Accessibility, 2018). Lighthouse provides the developers with audit “Best Practices”. The audit reports whether features such as usage of HTTPS, particular JavaScript function calls and deprecated markup have been used or implemented. The “SEO” audit reports regarding the page search engine results.
5 CONCLUSION

The commissioner had the need for a new, fast, modular website which meets the given technical requirements (Table 2.). This thesis followed up the implementation process of ID BBN’s newly developed website.

According to the given technical requirements (Table 2.), it was possible to conduct website architecture planning and to fulfill all given requirements.

![Performance comparison between existing website's pages and the newly-developed pages.](image)

Picture 24. Performance comparison between existing website's pages and the newly-developed pages.

One of the issues addressed by the commissioner was that the website content assets were stored in different Content Management Systems. This resulted in an inconsistency of the layouts, styles, code and caused chaotic environment when something needed to undergo changes. By implementing the website in OCM, the content is stored in one CMS – OCM (Table 2, requirement 1).

Another requirement set up was code modularity (Table 2, requirement 3.). The newly developed website consists of small blocks of codes which are reused on different pages during the implementation. Code-splitting improved significantly the performance (Picture 24; Table 2, requirement 2) by delivering to the user only the needed content at the moment of a request.
The newly developed website’s components are easy to create, distribute and modify by the non-technical team (Table 2, requirement 4.) by using the UI provided by OCM (Picture 25.).

Currently, the outcome of the project satisfies the needs of the commissioner. The following up step will be to create an English language version of the website.
REFERENCES


