Themes in Totara

Creating and cloning

Fredrik Österdahl
Ilias Hendo
Themes in Totara - Creating and cloning

Fredrik Österdahl
Ilias Hendo
Abstract

The purpose of this paper is to give the reader an insight into the graphical interface of a Learning Management system, the theme, how it functions and its importance. The main point is how the creation of new themes can be done, and two different approaches are examined and compared.

The use of learning management systems have been growing rapidly over the past years. Modern learning, be it basic school, higher education or professional training, is almost always supported by some learning management system where both instructors and participators share learning material, assignments, discussion, among other things.

The theme of a learning management system is of great importance and can have a substantial impact on the efficiency of the actual learning process. Themes might also differ in their purpose, thus depending on the target audience, creating a specific theme might be desirable.

The work done in this paper examines the structure and function of themes in the web based Totara learning management system, and looks at two different approaches to creating new themes. These approaches involve creating something from scratch, and cloning an already existing theme.

The conclusion drawn from the work done in this paper is that the most efficient way of creating a new theme, is through cloning an already existing theme. Only when time and experience is plentiful, and a full control over the theme structure is desirable, is creating a theme from scratch a viable option.
# Table of contents

Abstract .................................................................................................................. VI

Table of contents .......................................................................................... IX

Table of figures ............................................................................................ XII

Table of tables ............................................................................................... XV

1. Introduction ................................................................................................. 1
   1.1. Outline .................................................................................................. 2

2. Background .................................................................................................. 4
   2.1. Introduction ......................................................................................... 4
   2.2. E-learning history ............................................................................... 4
   2.3. Learning Management System ......................................................... 5
   2.4. The use of LMS’s ............................................................................. 6
   2.5. Moodle ............................................................................................... 7
   2.6. Kineo .................................................................................................. 8
   2.7. Totara .................................................................................................. 8
      2.7.1. Totara functionality ..................................................................... 8
   2.8. Themes ................................................................................................. 10
   2.9. Analysis and Creation of a Totara theme ....................................... 10
   2.10. Chapter Summary ................................................................. 11

3. Themes in Totara ....................................................................................... 14
   3.1. Introduction ......................................................................................... 14
   3.2. Examples of LMS themes ............................................................... 14
      3.2.1. Samsung Backstage ................................................................ 14
      3.2.2. Fronter ....................................................................................... 15
      3.2.3. Standard Totara Responsive .................................................... 16
   3.3. Programming languages and libraries ........................................... 17
      3.3.1. PHP .............................................................................................. 17
      3.3.2. HTML .......................................................................................... 17
      3.3.3. CSS .............................................................................................. 17
         3.3.3.1. Less......................................................................................... 18
      3.3.4. JavaScript ................................................................................... 18
         3.3.4.1. YUI ....................................................................................... 18
         3.3.4.2. JQuery .................................................................................. 18
   3.4. Software ............................................................................................... 18
      3.4.1. FileZilla ....................................................................................... 19
3.4.2. Notepad ++ ................................................................. 19
3.4.3. Brackets ........................................................................ 19
3.5. The STRT - a case study .................................................... 19
  3.5.1. Config.php ................................................................. 20
    3.5.1.1. Name ..................................................................... 21
    3.5.1.2. Parents .................................................................... 21
    3.5.1.3. Sheets ..................................................................... 21
    3.5.1.4. Layouts .................................................................... 21
    3.5.1.5. enable_dock ............................................................. 22
    3.5.1.6. rendererfactory ....................................................... 22
    3.5.1.7. javascript_footer ................................................... 22
  3.5.2. renderers.php ................................................................. 23
  3.5.3. version.php .................................................................. 23
  3.5.4. Theme Directories ........................................................ 23
    3.5.4.1. JavaScript ............................................................... 23
    3.5.4.2. Style ..................................................................... 24
    3.5.4.3. Lang ..................................................................... 24
    3.5.4.4. Layouts ................................................................... 25
    3.5.4.5. Images ................................................................... 29
      Pix .................................................................................. 29
      Pix_plugins ....................................................................... 30
      Pix_core .......................................................................... 30
  3.6. The Bootstrapbase Theme .................................................. 30
    3.6.1. CSS ....................................................................... 30
    3.6.2. JavaScript ................................................................. 30
    3.6.3. Layouts ..................................................................... 31
  3.7. Chapter Summary .............................................................. 31
4. Project Implementation ............................................................ 33
  4.1. Introduction ..................................................................... 33
  4.2. Creating a Totara theme .................................................... 33
    4.2.1. Other folders ............................................................. 33
    4.2.2. The first layout .......................................................... 34
    4.2.3. Start styling ............................................................... 35
    4.2.4. Renderer .................................................................... 37
    4.2.5. Continue Styling ......................................................... 43
    4.2.6. Functionality .............................................................. 44
4.3. Short on the method of cloning a theme ................................................................. 45
  4.3.1. Cloning..................................................................................................................... 45
  4.3.2. Cloning the folder .................................................................................................. 46
  4.3.3. Rename .................................................................................................................. 46
  4.3.4. CSS ....................................................................................................................... 47
  4.3.5. The layout files ....................................................................................................... 48
  4.3.5.1. Custom renderer ................................................................................................. 49
5. Results.......................................................................................................................... 51
  5.1. Introduction ................................................................................................................ 51
  5.2. The created theme ..................................................................................................... 51
    5.2.1. Comparison to the STRT ...................................................................................... 52
    5.2.2. The resulting structure ......................................................................................... 53
    5.2.3. Problems ............................................................................................................... 53
  5.3. Difference in approaches .......................................................................................... 54
    5.3.1. Product .................................................................................................................. 54
    5.3.2. Time ....................................................................................................................... 55
  5.4. The Totara framework .............................................................................................. 55
6. Conclusion .................................................................................................................... 58
  6.1. Introduction ................................................................................................................ 58
  6.2. Future development ................................................................................................... 58
  6.3. Project Evaluation ..................................................................................................... 59
    6.3.1. General problems ................................................................................................. 59
    6.3.2. Time ....................................................................................................................... 59
  6.4. Closing words ............................................................................................................ 60
References .......................................................................................................................... 62
A. Appendices .................................................................................................................... 65
   Abbreviation list ............................................................................................................... 65
   Page Types ....................................................................................................................... 65
   Theme options .................................................................................................................. 66
   Source code for created theme ....................................................................................... 68
# Table of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Samsung Backstage LMS. From the left: front page, user page, and course selection page.</td>
</tr>
<tr>
<td>3-2</td>
<td>Fronter LMS used by Högskolan Dalarna. From the left: front page and course page.</td>
</tr>
<tr>
<td>3-3</td>
<td>The Standard Totara Responsive Theme login menu.</td>
</tr>
<tr>
<td>3-4</td>
<td>The Standard Totara Responsive Theme front page.</td>
</tr>
<tr>
<td>3-5</td>
<td>STRT directory.</td>
</tr>
<tr>
<td>3-6</td>
<td>CSS files in the style folder.</td>
</tr>
<tr>
<td>3-7</td>
<td>The queen is welcome, and Mario is benevenuto.</td>
</tr>
<tr>
<td>3-8</td>
<td>The layout files in the layout folder.</td>
</tr>
<tr>
<td>3-9</td>
<td>The columns3 layout file with the headers’, columns and footer sections marked.</td>
</tr>
<tr>
<td>3-10</td>
<td>The page layout of the STRT. The header, columns and footer are marked.</td>
</tr>
<tr>
<td>3-11</td>
<td>The STRT in a smaller screen.</td>
</tr>
<tr>
<td>4-1</td>
<td>The new config.php file.</td>
</tr>
<tr>
<td>4-2</td>
<td>The general layout file.</td>
</tr>
<tr>
<td>4-3</td>
<td>Part of the config file where the layout option is filled with every page type.</td>
</tr>
<tr>
<td>4-4</td>
<td>The current login page of the theme.</td>
</tr>
<tr>
<td>4-5</td>
<td>CSS rule affecting all elements.</td>
</tr>
<tr>
<td>4-6</td>
<td>The CSS option with the general CSS file in the config.php file.</td>
</tr>
<tr>
<td>4-7</td>
<td>Part of the core_renderer class public method user_menu.</td>
</tr>
<tr>
<td>4-8</td>
<td>Part of the HTML output from the user_menu method.</td>
</tr>
<tr>
<td>4-9</td>
<td>The general layout file extended with more output methods.</td>
</tr>
<tr>
<td>4-10</td>
<td>The breadcrumb method.</td>
</tr>
<tr>
<td>4-11</td>
<td>The media rule for the user menu.</td>
</tr>
<tr>
<td>4-12</td>
<td>The jQuery function for the user menu drop down list.</td>
</tr>
<tr>
<td>4-13</td>
<td>The user menus’ graphic interface. At the top is the default. Below is the dropped down list.</td>
</tr>
<tr>
<td>4-14</td>
<td>The name change in the lang directory.</td>
</tr>
<tr>
<td>4-15</td>
<td>The name option in the config.php file changed.</td>
</tr>
<tr>
<td>4-16</td>
<td>The name change in version.php.</td>
</tr>
<tr>
<td>4-17</td>
<td>The color rules for headings and links.</td>
</tr>
<tr>
<td>5-1</td>
<td>The created theme. Above: the theme on a default page. Below the menu tab opened.</td>
</tr>
<tr>
<td>A1</td>
<td>The source code for the layout.php file.</td>
</tr>
<tr>
<td>A2</td>
<td>The source code for the JavaScript functions.</td>
</tr>
<tr>
<td>A3</td>
<td>The source code for the general.php layout file.</td>
</tr>
</tbody>
</table>
Table of tables

Table 3-1. The attributes found in config.php and their description .................................................. 20
Table A-1. The different page types that exists within Totara LMS ...................................................... 65
Table A-2. The options available in the config.php file within the theme folder ................................. 66
1. Introduction

Learning management systems are used throughout the educational and personal training world. The graphical user interface, theme, is an important aspect of the learning management system and can heavily influence the learning efficiency and user experience. Different themes can be used to target different audiences, so it can sometimes be useful to create a theme for a specific audience. The work done in this paper focused on the function and creation of themes in a learning management system, called Totara.

The paper reports the results and experiences in creating a theme and especially addresses the questions of creating a new theme from scratch vs cloning an already existing theme. The results aims to provide information about what ways are most efficient and in what circumstances when developing themes in Totara, and provide a guideline for developers when choosing how to start developing their themes.

The results found, showed that for an inexperienced developer in this area, in most general cases, it is better to start with a clone from an existing theme and further develop this, whereas more experienced programmers might benefit from both, and could decide what approach taken, depending on what kind of theme they want to create.
1.1. Outline

In chapter 2 we start with a brief history of educational technologies and distance education, in a wide historical context, and how learning managements system came about and developed. Next a short overview of the use of learning management systems, followed by a discussion of the economic side of this technology and the future of this industry. The biggest open source learning management system Moodle is described, followed by a description of Totara. Chapter 2 ends with an analysis of what role a theme has in a learning management system.

Chapter 3 starts by showing examples of different kind of themes in different learning management systems followed by short descriptions of the coding languages and software used. After that, a comprehensive case study of the Standard Totara Responsive Theme is done, ending with an explanation of the role that the bootstrapbase theme has.

In chapter 4 we get a close look into the implementation and development of a theme from scratch followed by a short description of how cloning the Standard Totara Responsive theme is implemented.

Chapter 5 is then going to present the results of the created theme. The difference in implementation between creating a theme and cloning a theme is brought up, and a couple of comparisons are made between the two approaches. The chapter ends with a general criticism of the framework provided by Totara for creating themes in terms of code aesthetics and effectiveness.

Our conclusions and result discussions are then presented in chapter 6 followed with personal thoughts about this Dissertation and closing remarks.
2. Background

2.1. Introduction

In this chapter, a brief history of distributed learning and the arrival and evolution of Learning Management Systems (LMS) is described. The characteristics of LMS's and how they differ from one another is described and the use of LMS's today and in the future, is discussed.

The arrival and history of Moodle, one of the first open source and free to use LMS’s, is brought up. Then the company that this work was done with, Kineo, and their main product Totara LMS, which the work in this paper is based upon, is described and analysis of themes of different LMS’s are shown so that the reader gets familiarized with the notion of what a theme is and how they might affect the entire experience of an LMS.

Lastly the argument of why this work should be done and what a desired result should be is discussed.

2.2. E-learning history

Teaching and learning have always been dependent on the resources available to the teachers and the students. In the past, if you wanted to learn something, such as a profession or a trade, you had to learn it directly from someone, person to person. Knowledge was gained through a personal exchange of information between people, or by experience through trial and error.

Tools to make learning easier have always been common and new technologies are often integrated into the classroom environment. Writing slates used by students have been around for long time. The abacus which has been around for more than two thousand years, though primarily used by merchants, has been used for faster and easier learning [2]. In the 15th century with the introduction of print, books became widespread and somewhat easily available [3]. Since, they have been the base of education and learning. But books are limited in their ability to give the reader feedback on their progress. Thusly, the writer’s
intent might not be reflected clearly enough, so for the reader, who reads and interprets the text, there is no way of directly correcting misinterpretations that might occur.

In 1728 an attempt was made of giving lessons about writing techniques using postal services [4]. In the mid-19th century the first real distance course was given by Isaac Pitman [4]. This was made possible through the mail system, were Isaac and his students exchanged assignments and answers via letters [4]. In the 19th century a few universities offered distance courses via mail as well [4].

Both radio and television have been used as educational tools since the 1930s and they provided means of interaction that written text isn’t capable of [5]. Radio provided a more oral approach where students only needed to listen, which can be argued, is a more natural way of human communication than reading[6]. Television via broadcasting could include a more visual approach as moving images allowed for more illustrative explanations. Television and radio was a major milestone in the distribution of information and thus education, but both radio and television are limited in the same aspect a book is: The communication is one-way.

With the widespread use and availability of computers and mobile devices and the development of the world-wide web, the potential means of bringing educational content to students changed dramatically. Entire courses, with lectures, assignments, and other resources, can be put online, which made access as easy as having a computer with an internet connection. Students can turn in assignments and get feedback through the course site at the click of a button. The internet revolution has certainly, and continually will, change the way which teaching and learning is practiced.

2.3. Learning Management System

The internet opened up a world of new ways in which students could interact with teachers and educational material. However, the need to organize material, manage students, deliver feedback and other practicalities arose when education started moving to the digital world. This is where a LMS comes in.
As the name suggests a LMS is used to handle all aspects of the learning process [7]. It is capable of delivering course content, administer courses, track student progress and performance, register students to courses, and much more [7].

FirstClass which was used in the 90s is considered to be the first modern LMS [8]. At the turn of the century the market for electronic education grew and a number of different LMS’s were created to suit different needs. In 2002 the free and open source LMS, Moodle, was released and has continued to be developed and is still evolving to this day [9].

Today, most of the modern LMS’s are hosted on the cloud, meaning that there is no need for a server on the company property and maintaining the in-house systems [10]. A number of different LMS’s exist, catering to different types of education and businesses. An example of this is Kannu which is built for creative education in music, arts, and design [11]. Kannu was released in November of 2014 by a California based company Kadenze.

2.4. The use of LMS’s

Many businesses today rely on employees being competent in their profession and willing to expand their knowledge. New technology is introduced and applied every day, so it is vital that employees get familiar with it fast. An employee that does not have sufficient knowledge and skill will certainly be less productive than an employee that does, and is therefore an unnecessary cost for the company.

Keeping the employees level of competence to match new technologies is important in many different aspects [14]. The biggest benefit can be a more effective and happy workforce, which keeps costs down and production high [13]. Companies that do not promote development of their employees are at risk of falling behind to competitors that do [13].

In schools and educational institutions, the migration of material available as online resources, is ever increasing. Both schools and companies can benefit a lot by using an LMS
Themes in Totara for their students and employees [12]. An LMS could for example: make the learning more structured, aid in supervision of users, and increase accessibility of content [12].

For companies, it could be both a price effective and time saving investment [13]. Many LMS solutions today are offered as a Software as a service (SaaS), where the LMS is cloud based and managed by the LMS provider [10]. This does not require as much resources in the form of hardware and technicians from the buyer to deploy.

The future use of LMS’s seems promising as more and more companies and educational facilities choose to adopt them. According to a new market research report, published by MarketsandMarkets, the LMS market size is expected to grow from USD 5.22 billion in 2016 to USD 15.72 billion by 2021 [15]. A big contribution to that is the increase of digital learning among corporate organizations and academic institutes.

In higher education, it is reported that “Estimates of institutions running an LMS are almost always near 99%. According to the first ECAR survey of faculty and IT, 85% of faculty use an LMS (with 56% using it on a daily basis), and 74% say it is a useful tool to enhance teaching. Among students, 83% use an LMS, and 56% say they use it in most or all courses.” [16]. These reports show that the LMS market will continue to grow substantially in the foreseeable future.

2.5. Moodle

Moodle (Modular Object-Oriented Dynamic Learning Environment) is a free and open-source software LMS [17]. Moodle was developed in the early 2000s and was released in 2002 by Martin Dougiamas [18]. Moodle is used as a platform for distance education and other e-learning projects in schools, universities, workplaces and other sectors [17].

Moodle is built on PHP which runs well together with the most widely used web server Apache [19]. Since the software is built on a modular design, the base software is highly versatile, which can be customized for any purpose, both commercial and non-commercial endeavors [17]. Plugins for all sorts of specific purposes exists to increase the LMS
functionality. Blocks are another part of Moodle that are used to create visual interface elements to certain plugins. These blocks can be displayed and modified so as to alter the tools visible for the user. One such example is the calendar block which displays a calendar for the user.

Moodle has been in constant development since its release, the open source nature of Moodle has made it possible for anyone interested and capable to contribute [18]. Moodle has a very large active community of people who are using and developing the system [18]. There are a number of forums on the Moodle home page which deal with the development of Moodle. Today the development is being managed by the independent company Moodle Pty Ltd [18]. They are responsible for developing the Moodle platform and incorporating new ideas and solutions from the Moodle community into the Moodle core [18].

2.6. Kineo

Kineo was founded in 2005 in England and was from the start focused on how technologies for learning could be used [20].

Today their main business is providing companies with solution for education mostly through digital means [20]. They tailor custom LMS solutions based on Totara, see chapter 2.7, to suit the needs of their customers [20].

2.7. Totara

Totara LMS is an open source custom distribution of Moodle that extends upon the functionality that Moodle provides [21]. Totara brings with it a range of core functionalities that are typically required for corporate, healthcare and government learning environments [21]. Totara was released on the global market in 2011 and took a role that Moodle was unable to provide, which was the core requirements expected of a high-end corporate LMS [22].

2.7.1. Totara functionality

Some of the core functionalities that are typically required for a corporate learning environment included in Totara are:
Hierarchies

Hierarchies are one of Totara LMS biggest features [22]. Most companies have a worker hierarchy with supervisors and subordinates whom all need different competence and training. Hierarchies in Totara makes it possible to build a ranking system for employees, this makes it easier to manage them and make sure that different employees get the training that are suited for their position within the company [22].

Sophisticated Reporting

Totara LMS has a very rich and flexible reporting capability which allows the user to customize the reporting to their needs [22]. One such example is the 360 feedback which allows users to give, and receive feedback from subordinates, peers and supervisors [22].

Easy Course Authoring

Flexible course options allow you to set roles and permissions, define course progression, and form group-based activities [22]. In Totara, it is easy to implement instructor-led, self-paced, social and blended learning strategies. Totara covers both authoring and learning management [22].

Flexible Learning Plans

Learning plans makes it easy and keeps you up to date with all the learning, competencies and objectives that employees are working on [22]. Totara makes it easy to set up learning plans for the learners, with learning plan templates to structure the connected workflow between learners and their managers [22].

Badges

Badges are used as motivation among other things. They are very simple but a very effective way of accrediting and gamifying learning. A badge can be assigned if the criteria’s of said badge are completed, for example completing a course or passing an assessment. They can be a visual recognition of performance. Badges can also be used for a number of other reasons, like identifying polls of talent and specialists quickly and easily.
2.8. Themes

A theme is the foundation for the interface and user-experience of any website. How the LMS looks and feels is perhaps one of the most important part of the user experience. A theme might have all the features that the user requires, but if the interface makes using these features obscure and counterintuitive, the user might not be able to utilize them correctly.

The theme dictates the position of content, navigation, and several user-interface elements, basically the information architecture of the LMS. The information architecture affects the experience of users of the website and it is an important point to look for when creating a theme.

Today when many browse the web on their smartphone or tablet, it is also vital that the theme looks good on smaller devices. Websites that adapts the layout to fit the users screen are said to be responsive. A website with a responsive design will automatically change to fit a smaller or bigger screen size, without compromising the usability of the user interface.

Another aspect that might have a significant impact on user experience when interacting with the LMS or any website in general, is the load time of the website. Some research suggests that up to 40% of users abandoned a website that takes more than 3 seconds to load [24]. The theme structure and implementation can greatly influence the load time of a website. Therefore, it is desirable to make a website not only appealing in aesthetics and somewhat intuitive to the user, but still load fast enough not to make the user lose interest.

2.9. Analysis and Creation of a Totara theme

Totara is built upon Moodle which means that most of the principles in creating a theme for Moodle apply to Totara, but there are no guides or software to help in the process of creating a custom theme for Totara.
The purpose of this thesis is to make it easier for theme developers to know when it is most useful to create a custom theme from scratch or cloning an existing theme. Both these approaches will require an understanding of the components and workings of themes in Totara. The components of Totara themes and their structure are analyzed and documented and the different approaches of creating a theme, is evaluated.

The documentation should provide a reference manual for creating future Totara themes and should also be a guide to when it is appropriate to create a new theme and when cloning an existing theme might be a better choice.

In the comparison, a number of important points are explained for how the themes work in Totara. How the structure of the theme looks like is explained and what role each file has and what is possible to do with them. This is important both when it comes to creating a new theme, and cloning a theme since both methods requires an understanding of how themes work in Totara.

2.10. Chapter Summary

In this chapter, we started with a short description and history of e-learning and the educational distribution all the way from the abacus to the modern internet. This then led to the concept and development of LMS’s and their characteristics. We discussed how the first LMS came to be and how they evolved through the 20th century.

Then the use of LMS’s today and their importance for educational institutions and companies was mentioned, followed by an analysis of the future use of LMS’s.

The attributes of the one of the first and, as of now, the biggest open source LMS, Moodle, was shortly mentioned. The company which we are doing this work for, Kineo, and their main business was mentioned along with their use of the Totara LMS based on Moodle.
Totara LMS was brought up and its key attributes described. The concept of themes was explained and what effect they have on LMS’s as a whole. And lastly arguments for why this paper was done and what it aims to achieve was discussed.
3. Themes in Totara

3.1. Introduction

In this chapter, different themes from different LMS’s will be looked at and described to give the reader a visual reference of what different themes might look like and how they can impact the user experience.

Then a more detailed view on what programming languages that make a theme work in Totara is brought up. A short list of different editors that can be used to create a theme in Totara is described.

The Totara standard responsive theme is described in detail. The structure of the theme is analyzed and the different files explained.

3.2. Examples of LMS themes

There are many different LMS’s and many of them have many different themes they can use. Different themes are used for different application purposes and businesses. They have different aspects that target their specific audience and educational goals.

3.2.1. Samsung Backstage

![Figure 3-1. Samsung Backstage LMS. From the left: front page, user page, and course selection page.](image)

Samsung backstage is built on Totara and is a job Kineo did for Samsung. The LMS is designed to educate salesmen in retailers about certain products. Emphasis lies on sales pitches. Products are clearly visible on the front page, seen on Figure 3-1, and clicking on
them takes you right to a course about that certain product. This makes user interaction quite easy and intuitive. A retailer employee that wants to learn more about a certain product, simply clicks on the product window to go to a course page where there are information and quizzes about the product.

The front page has a big sliding image gallery which can introduce new items that are relevant to the user. Moving images often catches the user’s attention so products shown in the sliding image gallery is likely to be targeted by the user. Further is a user information block which displays a user profile picture. The badges block is also clearly visible on the front page. Badges are collected through completing courses and tasks and offers encouragement for further learning.

Samsung backstage is based on Totara and uses blocks, described in chapter 2.5, to display most of its content. Every block is an interface to the course page of that particular block. This makes it easy to add and remove content in a dynamic way where blocks are added or removed by supervisors to promote the current courses.

3.2.2. Fronter

![Fronter LMS used by Högskolan Dalarna. From the left: front page and course page.](image)

Fronter is an LMS that recently was acquired by the company It’s learning [25]. Fronter has been focused on making LMS’s for academic learning environments such as schools and universities [26]. Figure 3-2 shows the Fronter LMS environment used by Högskolan Dalarna, an institution for higher education in Sweden. Since the LMS should contain many different
courses and learning material, used by students of different backgrounds, the layout is quite general and discreet.

Navigation is done with clearly visible menus and navigation items at the top and at the left but leave most of the page to be used for course material. The color scheme of Fronter is neutral compared to Samsung which is vibrant and full off sharp pictures.

3.2.3. Standard Totara Responsive

![Figure 3-3. The Standard Totara Responsive Theme login menu](image)

![Figure 3-4. The Standard Totara Responsive Theme front page](image)
The Standard Totara Responsive theme (STRT), is a basic theme with a simple page layout. Figure 3-3 shows the login page, which is quite simple in its layout, without any sharp colors or advanced features. The main layout, used with most pages seen in Figure 3-4, incorporates a header with navigation, a middle section with columns for content and blocks, and footer with information and navigation.

The theme is designed not to be used as a finished product, but to be a base for extending new themes. Therefore, the theme is a bit barren and empty with not many whistles and bells, compared to the Samsung Backstage layout, both are Totara LMS.

3.3. Programming languages and libraries

Totara is a web based service and is, just as Moodle, written in PHP. Themes are responsible for the look and feel of the site, and are therefore mostly made up by HTML, CSS and JavaScript. Most themes have minimal PHP code since most changes that is made to change the appearance of a theme can be accomplished using mostly CSS, and if necessary some HTML. JavaScript provides the theme with functionality and is used to give the user a more dynamic experience.

3.3.1. PHP

PHP is script language that is mainly used by server side clients in web based systems but can also function as a general-purpose scripting language [28]. In web development, PHP creates the logic and function of the web site at the server side and might also communicate with a database for data retrieval and data insertion. The core of both Totara and Moodle is written in PHP.

3.3.2. HTML

Hyper Text Transfer Protocol (HTML) is the language of the web, and since Totara is a web based service, its contents are presented in the HTML format to a browser. HTML is a markup language which dictates how data should be formatted [28].

3.3.3. CSS

Cascading Style Sheets (CSS) is a way of styling the HTML elements in a wide variety of ways [29]. HTML is all about structure, while CSS is all about looks. Different CSS rules can make the same HTML look completely different.
3.3.3.1. Less

Less is a pre-processor for CSS and allows for writing CSS in a more advanced and object oriented way, with functions and variables [30]. The Bootstrap base theme which is a parent theme to the standard responsive theme uses less for its CSS. Less files are compiled into regular CSS files.

3.3.4. JavaScript

JavaScript is the most popular client side scripting language on the web [31], and is natively supported in the most popular web browsers of today such as Google Chrome, Safari, Firefox and Internet Explorer [42]. JavaScript can add functionality to a website such as moving menus and pop up messages.

3.3.4.1. YUI

Yahoo User Interface (YUI) is a JavaScript and CSS framework API which aims to make development easier [32]. YUI is used to make more interactive web pages and minimize the work effort. Many of the core functions within Totara use the YUI library to make dynamic user interface elements that eventually will impact the appearance and behavior of a theme.

YUI development has been suspended by Yahoo and the Moodle community has plans to move away from YUI and incorporate more of jQuery and Asynchronous Module Definition (AMD) modules instead [33]. Therefore, it is likely that Totara will use less of the YUI library in the future as well.

3.3.4.2. JQuery

JQuery is a JavaScript framework API which includes a large library of common functionalities to manipulate objects in the Document Object Model (DOM) model. The library includes an API to get easy access to any element in the DOM tree, react to events, change element style, attributes, and html, and create animations such as sliding and fading [34]. JQuery is free to use and is the most popular JavaScript library as of today [34].

3.4. Software

To create and modify themes in Totara all that is needed is a good text editor, preferably one which supports PHP, HTML, JavaScript and CSS. Moreover, if Totara is located on a remote server, a way of transferring files from the work computer to the server is required. For this a FTP client is a good choice.
3.4.1. FileZilla

FileZilla is a free cross-platform File Transfer Protocol (FTP) application, which is used to transfer files from computers to the server hosting the website [35].

3.4.2. Notepad ++

Notepad++ is a programming text editor for windows. Notepad++ has syntax highlighting and auto completion for many languages, including PHP [36].

3.4.3. Brackets

Brackets, is a free and open source text editor, designed specifically for web development [37]. It features a wide number of functions to aid web development and web design. Beyond regular IDE features such as syntax highlighting and auto completion, it allows for real time editing so that changes in the html and CSS code are immediately visual in the browser [37]. This allows for more dynamic development because of quick changes and instant feedback.

3.5. The STRT - a case study

Every Totara theme has a similar structure and all contain the same types of files and folders. There are some differences but all themes have at least a common subset of files and folders. The Totara version available for this project was version 2.7.17 which is built upon Moodle version 2.7.15. Newer versions of Totara might look a bit different, but the overall structure is likely to be the same.

The easiest way of getting an overview of how themes work and their structure is to look at an existing theme and analyze its components. The cloned theme used for this paper will be based upon The STRT, so it is a good idea to start looking at how this theme is structured.
Figure 3-5 is a screen caption of the STRT folder that gives an overview of the theme structure. Here the different folders and files that make up the theme are displayed. The STRT is a good example of what a Totara theme folder contains, as all files and folders are straightforward, there are no files for development, such as for example less files, or special JavaScript folder full of library functions.

3.5.1. Config.php

The config.php file has a number of settings and options that will configure how the theme will behave and work. Table 3-1, shows the settings and options defined in the STRT but there are a lot more of options that can be set in this file. Most options however, are not necessary for most theme creators. See Table A-2 for a full reference and examples.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Every theme needs a name to differentiate it in the Totara system.</td>
</tr>
<tr>
<td>parents</td>
<td>The parent themes. An array of names of the parent themes which is being used as a base for the theme.</td>
</tr>
<tr>
<td>sheets</td>
<td>The CSS files that this theme will use for its layout files.</td>
</tr>
<tr>
<td>layouts</td>
<td>A list which layouts will be used for the different pages.</td>
</tr>
<tr>
<td>enable_dock</td>
<td>Specifies if the theme will allow docking of elements.</td>
</tr>
<tr>
<td>rendererfactory</td>
<td>Specifies the names of custom renderers that will be found in the file renderers.php in the theme directory.</td>
</tr>
</tbody>
</table>
3.5.1. Name

The name of the theme, which is likely the name of the theme folder, is a reference to this particular theme.

3.5.1.2. Parents

The parent option specifies possible parents of the theme. Themes will inherit all the CSS, layouts, and JavaScript from their parent themes. If not overwritten, CSS, JavaScript, and layouts from the parent theme will be used. If the parent theme in turn extends another theme, that theme must also be specified in the grandchild theme.

3.5.1.3. Sheets

In the config.php is a list of CSS files that will be included in Totara page requests. This list contains only the names of the CSS sheets that are to be imported, not the file extension* and it is expected that these sheets can be found in the style folder in the theme directory. If a CSS sheet is in the style folder but not specified in the sheet list in the config.php file, it will not be imported.

3.5.1.4. Layouts

The layouts list in the config.php file is a list of all the layouts that the theme should use. The layout files are composed of fixed html and PHP to insert the interchangeable content, generated by a renderer. That means that a layout will have a fixed structure but with different content. Different layouts are specified so that they are used for different page types. There are a number of different page types defined in Totara. Examples are base, standard, login, and so on (see Table A-1 for all page types). If a particular layout is not specified in the theme, a layout from the parent theme will be used.

The STRT uses specific layouts for page types: base, noblocks, login, popup, and report. That means that if a user were to request a page that has a different page layout type, say for instance course, a layout specified in the parent theme must be used.
Further information about each layout is specified in the layout list such as: What theme a layout is from, which means that a layout that exists in a different theme could be used, and the name of the layout file must be specified, which is a reference to the actual layout file which should be in the layouts folder.

Here is also a specification of block regions which tells Totara where blocks are allowed to go in the layout file. Block regions are sections of the page where blocks reside and blocks can, if permitted, be moved from one region to another if there exists more than one region. One can also specify a default block region where all blocks are located per default.

3.5.1.5. enable_dock
The enable dock attribute is an option to allow the theme the ability to hide away blocks into a dock. The default dock is hidden when there are no blocks in it and is a thin vertical bar on the very left of the page when it is visible. The dock can however be styled just as any other HTML using CSS and thus can take different forms.

3.5.1.6. rendererfactory
Renderer factory is an option for specifying that a custom renderer should be used in this theme. A renderer object is an object that handles all the output of the html content which is sent to the user. Overriding a renderer gives the theme creator more control over the html output which greater increases customization abilities.

3.5.1.7. javascript_footer
The javascript_footer option is an array of all the JavaScript files, without file extension, that should be included from the script folder. JavaScript footer sets the layout up such that the JavaScript is added at the footer, or bottom of the page, which can greatly increase page load time compared to adding it to the head of the page. This is because the html and CSS is allowed to be rendered before the scripts are fetched and executed by the browser.
3.5.2. renderers.php

Renderers are PHP objects that produce HTML output for the page. They are used to produce HTML output for the pages in Totara. There are several different renderer classes that are available from the core system. These are the core renderer used to create the main HTML for course content, navigation and such.

There are two different renderers for different uses within the system. The core renderer is used to render general HTML, and plugin renderer which plugins such as blocks use to create HTML. As we see in the picture, both these renderers inherit from an abstract class.

In renderers.php you can extend the renderer class and so specify more precisely what HTML should be output to the page which increases the customization ability of the theme. Overwriting renderers gives the theme creator full control of the HTML output that is created for each page. In Totara, there also exist a renderer specific for Totara elements. This can also be used in the theme to create menus, search buttons and much more, that are more specific to the Totara LMS system.

3.5.3. version.php

This file is nothing but Meta data about the theme. The name of the theme and the version of the theme could be specified. Information such as which version of Totara that is required for the theme to work, or any dependencies that the theme might have, can also be specified.

3.5.4. Theme Directories

The subdirectories in the theme folder, contains resources that the theme uses either when the system is running or resources for development. The Standard Responsive Theme has only files used directly by the theme and no development folders or other resources.

3.5.4.1. JavaScript

The JavaScript directory is the default location of the JavaScript files that the theme uses. The standard responsive theme only has one line of JavaScript that affects the look of the datepicker object.
The datepicker object is a jQuery User Interface object that is used with html inputs to allow users to select a date from a visual calendar [38]. The changes to the datepicker object are small, only removing the title and alternative text.

Most of the JavaScript functionality that is used in the standard responsive theme is inherited from the parent theme bootstrap base. Other JavaScript functions may also be included in various plugins and some core elements in Totara are rendered with additional JavaScript.

3.5.4.2. Style
The styles of the STRT are divided in files that are responsible for styling different elements and sections of the page.

![CSS files in the style folder](image)

Figure 3-6. CSS files in the style folder

Seen in Figure 3-6 are the stylesheets that the STRT uses. They are divided into files which each handles a different aspect of the page style. The names of the files indicate their function, for example the navigation file, is responsible for styling navigation elements and the course file styles course material.

3.5.4.3. Lang
The lang directory contains a set of string that this theme can use and can specify these strings in different languages. These strings can be used throughout the theme and are designed to be used with country codes to change depending on what language setting the user has.
This means that different users that speak different languages all get language strings that match their language. For example, in the `lang` directory, there might be a directory called `ita` that contains the set of strings to be used if the user speaks Italian, as described in Figure 3-7.

![Figure 3-7. The queen is welcome, and Mario is benevenuto.](image)

The theme does not have a particular big need for large `lang` files as themes are not supposed to change what is being displayed, but rather how it is displayed, so the `lang` directory will be sparse. The most important string in this folder is the one that contains the name of the theme. This string is used by the system to represent the theme.

3.5.4.4. Layouts

This file contains the layouts files for the theme. The layouts consist of html and minimal PHP to create the page content that is delivered. The page content which is different for different pages and users is inserted into the layout file using the core renderer object.

Different layout files are often used with different pages within Totara. A login page might use one layout while the front/start page uses another. However, it is possible to have multiple pages use the same layout file. The difference between the pages then, will only be the main content of the page, the page layout will most likely stay the same.

![Figure 3-8. The layout files in the layout folder.](image)
In Figure 3-8 are all the layouts that come with the STRT. There are three different layouts that are used throughout the system. These are the columns layouts. The columns layouts all consists of a header a large area with one to three columns, and a footer. One column always contains the main content of the page, while the others are capable of displaying blocks. The other layouts: embedded maintenance popup and secure are only used with their respective page type.

Not all pages are being represented in the STRT. For example, should a user request a page with the layout type course, Totara will check the current theme for the layout that represents this page type. If none is found, then the parent theme will be checked for a layout that corresponds to that page type. If no page type if found again the grandparent themes will be checked and so on. There should exist one layout for each of the page types or Totara will create one standard when the page is requested. This has the risk of making styles not apply, which makes the page look, and function badly.
Figure 3-9. The columns layout file with the headers’, columns and footer sections marked.
In Figure 3-9, parts of the layout file columns3 is shown. The code responsible for rendering the headers, columns and footer are marked with colored boxes. The structure of the layout files is HTML for overall layout purpose, with bits of PHP to create the actual content of the page.

The $OUTPUT object as seen in Figure 3-9, is an instance of the core_renderer class. It produces the header menu, the page header which is usually breadcrumb trail of links. Under the headers, in the main section of the page, the renderer creates the main content of the page and the block sections. And finally the footer is created at the very end of the page.

Figure 3-10. The page layout of the STRT. The header, columns and footer are marked.
Figure 3-10 shows the result of the layout file columns three. The color boxes marks the result of each bit of code that was marked in Figure 3-9. The responsive nature of the theme is shown in Figure 3-11, as the device width goes below a certain point, the content adapts.

3.5.4.5. Images
Some themes use images in their layouts. The default pix directory is used exclusively by the theme to incorporate images. The pix_plugins and pix_core directories are different. When pages are created, images are looked for first in the theme, then in any used plugins, and lastly in the core system. Themes can also overwrite images used by plugins and even overwrite images used by the core system by specifying folders with correct names and images.

Pix
The pix directory is the default directory where images that the theme uses are located. Here for example is the logo that the standard responsive theme displays at the top of many pages.
Pix_plugins

Overwrites images used by plugins. The plugin which should be overwritten must be specified by a subfolder.

Pix_core

The images that reside in this directory are used to overwrite images that are part of the Totara core and are used throughout the system.

3.6. The Bootstrapbase Theme

The bootstrapbase theme is meant to be a base of which new themes can evolve. It features little styling and only basic functionalities. The theme has no parent theme, and must therefore handle all different page types that Totara has. Thus every page type has a layout file associated with it in the bootstrapbase theme so that the bootstrapbase theme can handle all types of pages that a user might request. For the standard Totara responsive theme, this means that page types that do not have a corresponding layout file within the theme, will use the layout file found in the bootstrapbase theme.

3.6.1. CSS

The bootstrapbase theme implements the less CSS preprocessor to compile its CSS files. The effective CSS for the bootstrapbase theme is only a minimum to create the layout and some basic colors for the theme. It is purposefully designed that way so that it can be used as starting point for new themes.

The bootstrapbase theme CSS is built with the twitter bootstrap CSS framework. The framework allows for a responsive design where the page adapts to the page size. However to use the bootstrap framework, it is required to add classes to the elements that should be affected. To do this the bootstrapbase theme overwrites the core renderer, so that it produces elements with the correct classes. However, only a small part of the actual framework is being used.

3.6.2. JavaScript

The JavaScript that the bootstrapbase theme comes with a file to allow for backwards compatibility with internet explorer version 8 and 7, changing some elements and CSS for
these browsers, since those web browsers might have problems displaying the site properly otherwise.

Then there are the JavaScript based on YUI that handles dynamic functionality such as showing and hiding elements, handling iframes, and configuring the dock.

3.6.3. Layouts
The layout files in the layout directory of bootstrapbase have the same names as those found in the standard Totara responsive theme. This indicates that the standard Totara theme is, from the start, a copy of the bootstrapbase theme. The files are not identical in content though, as the standard responsive theme has added some functionality to them but the overall structure is the same. The information architecture is all structured in the same way with a header, a footer and number of columns for main content and blocks.

3.7. Chapter Summary
In chapter 3.2, three different themes were shortly analyzed and compared. It is easy to see that a theme can make a huge difference in appearance, but also functionality of an LMS. What and how the user interacts with the LMS, is a big factor in how well the LMS core functionalities are utilized. For example, incorporating badges but not display them to the user in an obvious way may nullify their purpose. The theme should make the user experience as pleasant and easy as possible by showing the functionalities of the LMS in an easily accessible way.

In 3.3, the programming languages and libraries that are used in Totara themes were described, then, a short list of software that can be used to create a theme in Totara was reviewed.

In chapter 3.5 the standard responsive theme was analyzed and described. The files and folders in the theme and their purpose were described in detail. An example of how a layout file is constructed and the resulting page information architecture that the user experiences were described.
4. Project Implementation

4.1. Introduction
As mentioned in chapter 2.8, it is important to pander to one’s audience. This is of course important when it comes to themes as they are the first, and arguably the most important impression, that the user will have. The specification that was given regarding the work in this paper, did not suggest any particular outcome, that is, particular theme style or function. This led to a general approach, creating a theme that was different from the STR theme, but still was a general purpose theme, not focusing on any particular target group, or profession.

4.2. Creating a Totara theme
The starting procedure when creating a theme from scratch is creating a new folder that should be used as the theme folder. This folder will contain all the files and folders that the theme will utilize. There are a number of important files that needs to be created and added for the theme to work properly and to display in a reasonable manner. The most logical approach is to start with the config.php file, since this file will control the entire workings of the theme.

```php
<?php
    THEME->name = 'create';
?>
```

*Figure 4-1. The new config.php file.*

The content of the created config.php file is visible in Figure 4-1. For now, only the name of the theme is specified. As seen the newly created theme is called create.

4.2.1. Other folders
The next step should be to create all the folders that a theme should have. In chapter 3.5 the STRTs structure and folder was explained. This leads us to the folder that the newly created theme should contain.
The *lang* directory, that contains strings that the theme can use, is created. The structure and content is created with the STRT as a guide. For now the only language that is supported is English. For future pictures, a pix folder is created as well. No pictures are used by the theme at the moment so the pix folder can be left empty.

4.2.2. The first layout

Next, a directory for the layouts should be created. In the created layout directory, there should exist at least one actual layout file.

The layout file is responsible for creating the HTML of the site. This is done by having some static HTML and using the output renderer as discussed in chapter 3.5.2. The most important part that is needed in the layout file is the call to `main_content` method. Without this the layout will fail.

```php
<?php echo $OUTPUT->doctype(); ?>
<html <?php echo $OUTPUT->htmlattributes(); ?>>
<head>
    <title><?php echo $OUTPUT->page_title(); ?></title>
    <?php echo $OUTPUT->standard_head_html(); ?>
</head>
<body <?php echo $OUTPUT->body_attributes(); ?>
    <?php
        echo $OUTPUT->standard_top_of_body_html();
        echo $OUTPUT->main_content();
        echo $OUTPUT->standard_end_of_body_html();
    ?>
</body>
</html>
```

*Figure 4-2. The general layout file.*

The created layout file is seen in Figure 4-2. It contains basic HTML for a basic structure, surrounding PHP functions that created the actual content of the page. Most of the pages HTML are created with the `OUTPUT` object which is an instance of the `core_renderer` class.

Now there is a layout directory and a layout file in this directory. This will enable references in the *config.php* file of page types to actual layouts, thus making the theme operational, though not perhaps ready for actual use. Since this is a brand new theme and there are no parents to this theme, this theme should reference all the page types to a layout. Right now,
only one layout file exists, so there is only one reference that can be made. This could be changed later if other layout files are added.

```php
$THEME->layouts = array(
    'base' => array('file' => 'general.php'),
    'standard' => array('file' => 'general.php'),
    ...
```

*Figure 4-3. Part of the config file where the layout option is filled with every page type.*

Now every page request that the user makes will use the layout file that was created. Figure 4-3 shows the reference of layout files to the page types. This particular layout file might not suit the needs for every page type so the theme isn’t working in an optimal way at the moment, but it does work. It is a good start. Specific page layouts can be implemented later.

*Figure 4-4. The current login page of the theme.*

Figure 4-4 shows the current login page. For now, it is plain HTML without any styles or specific functionality. The thing that the theme in Figure 4-4 is missing is proper styling, so the next step is the creation of a style folder.

4.2.3. Start styling

The style folder alone is of course not enough, what is needed are some stylesheets. The stylesheets can be created using some CSS preprocessor. However the most direct way of starting to style the theme is to create the stylesheets as plain CSS files using some text
editor. How to structure the style sheets, that is how different rules should be divided up into different files are a matter of individual opinion. One solution is to have all rules into one single big file, but for reasons of management, it is desirable to divide the styling rules into files in a logical way.

The idea behind the first file created, was that one CSS file should dictate some general rules that would apply to general html elements. This would create a general file where other stylesheets could then build upon. The file was named generalhtml.css to indicate its purpose as an HTML element stylesheet, rather than one targeting special elements.

The first rule that is added is applied to all elements and is a sort of reset to all elements and also allows for easier layout styling later. The margin set to zero will ensure that each element is a clean slate for later styling. Without this some elements have, per default, some margin. This could interfere with later styling so it is a good to remove it. Elements that should have margin will be given a margin explicitly instead.

```css
* {
    margin:0;
    box-sizing: border-box;
}
```

*Figure 4-5. CSS rule affecting all elements.*

The box-sizing property is used so that the browser calculates each block element width, including border and padding. This will allow better control over elements when it comes to their actual size, and not complicate things. Having to calculate the width or height of an element and also take into account border or padding that might be present, is somewhat tedious. The simple rule is defined in Figure 4-5.

To make the stylesheet actually apply to the layout file, it must be defined in the `config.php` file. Adding the general file as a style sheet in the `config.php` file will make sure that the CSS file is imported in every page request that a user makes. The option in the `config.php` file is seen in Figure 4-6.
4.2.4. Renderer

Before continuing it is worth mentioning the renderers and the approach taken. The theme has an option of overwriting output renderers to customize the HTML for the page as discussed in chapter 3.5.2. This is however can quite tedious and the results are not always worth the hassle.
Considering the output method `user_menu` as seen partially in Figure 4-7. The method is quite big in size, and utilizes a number of functions that are part of the core system, that are not related to the HTML used by the theme. There are several different options and cases that are being accounted for, such as the system being installed, and handling the user not being logged in the user being a guest user, and so on. Most of the output renderers’
methods deal with the nitty gritty details of the system before any output is actually created and returned for the theme to use.

The only time it is really necessary to overwrite an output renderer method completely, is when the actual content, that is the content gathered from the system such as what a menu should contain, specific user information and such, should be changed. This paper focuses on the creation of the theme, not the LMS content, but rather the presentation of the content. The content used will be that of which the STRT uses.

For the theme creator, the output renderer methods are only specific HTML that can styled using CSS, and manipulated by JavaScript. Styling HTML using CSS is a very powerful way of altering how content is displayed, so the need to change the actual returned HTML of an output method is not really needed. All that is needed to actually change the appearance of HTML returned by an output method, is knowing what structure this HTML has. Knowing the structure, what HTML elements and what classes and ids' that the returned HTML will have, is enough to be able to apply almost any changes in appearance that CSS provides.

For example, the method user_menu will return the HTML seen in Figure 4-8, when a normal user is using the system. This HTML can be manipulated via CSS so that it gets the appearance that is desirable. Using the output methods to create HTML and then using CSS to style this HTML is the approach taken to all the output methods. So using a number of output methods, the standard layout gets some more content.
Figure 4-9 shows the layout file with the user_menu method among other methods added for navigation, headers and for linking in JavaScript at the end of the footer. It is also wrapped in some HTML for a basic layout structure. Also added is a block section. This is a section where blocks are placed within the layout. To allow blocks in the layout, options must be added in the config.php file.

If extra HTML should be wrapped around any elements produced by the output methods it could be put in the layout file. Another way however is to overwrite the renderer, only adding the wrapping HTML to the already existing function output. This could be structurally better if the output methods took care of as much HTML as possible, and also handle the logic that will be added later when different page types get different content. The renderer_factory option is added in the config.php file and a file called renderer.php is created in the theme folder. This file will contain an output renderer class, overwriting the core_renderer so it is possible for the methods to be overwritten.

So for example, the navbar method returns a breadcrumb trail of links. This could be restructured so that the purpose of the method makes more sense in the layout file.

```php
<?php echo $OUTPUT->doctype(); ?>
<html <?php echo $OUTPUT->htmlattributes(); ?> >
<head>
    <title><?php echo $OUTPUT->page_title(); ?></title>
    <?php echo $OUTPUT->standard_head_html(); ?></head>
<body <?php echo $OUTPUT->body_attributes(); ?>>
    echo $OUTPUT->standard_top_of_body_html();
    echo $OUTPUT->user_menu();
    echo $OUTPUT->page_heading_menu();
    echo $OUTPUT->navbar();
    echo $OUTPUT->main_content();
    echo $OUTPUT->blocks(block-menu', block-menu');
    echo $OUTPUT->page_doc_link();
    echo $OUTPUT->login_info();
    echo $OUTPUT->home_link();
    echo $OUTPUT->standard_footer_html();
    echo $OUTPUT->standard_end_of_body_html()
</body>
</html>

Figure 4-9. The general layout file extended with more output methods.
public method `breadcrumb` is added. This method implements the `navbar` method and wraps the returned HTML in a div element with the class `breadcrumb`.

This kind of abstraction is made with several other methods as well. This makes the layout file much cleaner and the specific purpose of the output methods for this theme can be made more obvious. The purpose and the returned result of `breadcrumb` are easier to understand than the method `navbar`, which is ambiguous at best.

Another thing that should be abstracted away in the new output renderer is the layout options. These can be used for controlling specific layouts and their contents depending on the page type. Checking for these options could be done in the layout file, or it could be made in the output renderer methods implicitly. This means that the layout file will not contain any logic. All the decisions are made in the output methods before any HTML is returned.

There are different page layout types which have specified that some elements should not exist on the served page. An example is the popup layout type. This page layout should preferably not have any navigation, no blocks and no footer, since these elements do not contribute to the function of this particular page layout.

Removing these elements can be accomplished using layout options. Adding options such as `no_navigation`, `no_blocks` and `no_footer` to the layout in the `config.php` file is the first step. Then in the output renderer methods that are responsible for the output of these elements, a condition is added to check if the option is set.

```
public function breadcrumb()
{
    if(empty($this->page->layout_options['no_navigation']))
    {
        return "<div class='breadcrumb'>" . $this->navbar() . "</div>";
    }
    else
    {
        return '';  
    }
}
```

*Figure 4-10. The `breadcrumb` method.*
Figure 4-10 shows the method breadcrumb. The added condition checks if the page that is being requested should contain navigation or not and returns correspondingly. Conditions like these are added to methods as needed, following the page layout types specifications. This approach enables one single layout file to be responsible for the entire range of page layouts.

4.2.5. Continue Styling

Now, most of the layout content is done. Essentially what is left is styling, and dynamic functionality in the form of JavaScript. Starting with the CSS, creating a number files to get a starting structure. At the moment there is only one CSS file which as mentioned should contain some general CSS rules. How the structure of the CSS files into logical parts are done, is a subjective choice as discussed previously in chapter 4.2.3.

As of now, the single existing layout file contains minimal HTML and almost only output methods for HTML output. Therefore a good choice of CSS structure is dividing the rules applying to each output method that returns visible HTML elements, into separate files. Thus a logical separation of style rules is achieved and it is easy to style the HTML for each output method at a time.

One important aspect discussed in chapter 2.8, is that themes should be responsive. The responsiveness is often implemented through the use of CSS. With the correct CSS the styles can change according to some inherent feature of the device using the system. The most common is using the screen width of a device, separating different rules that should apply depending on some boundary value(s).

The media query is used to differentiate style rules, so that small screen get one set of rules, and bigger screen gets another. The approach taken was to differentiate between two different screen sizes. The boundary value of 786 pixels was used as a threshold for rule sets, meaning that screen sizes with less width that 786 pixels got one set of rules and screens above, another. The value of 786 was used since it is used by the popular bootstrap CSS framework as a boundary value.
Figure 4-11 shows the media rule for the user menu. The style rules shown will only apply when the screen width is less than 786 pixels, which means that the devices affected by them are smaller devices such as smartphones.

4.2.6. Functionality

After some styling, the theme is starting to look like it can actually be used by a human being. There is however features that can make a huge difference in the user experience other than appearance. Functionalities that make the site appear dynamic and living, are important for making the user experience worthwhile.

A common thing seen on many websites is the use of drop down menus. These add functionality for menus allowing most information to be hidden, until a user either clicks or hovers the mouse pointer on the menu. Drop down lists can be done through the use of CSS and the hover selector. But if the user mouse click should be used as a triggering event for the menu to drop down, JavaScript must be used. First, adding a JavaScript directory in the theme folder and adding a JavaScript option in the config.php file, is necessary to enable the theme to use the scripts.
The script for creating a simple drop down list is quite straight forward. Using jQuery, which is loaded automatically in Totara for any plugin to use, any element in the finished page can be altered using the selectors that jQuery provides. Using the available id’s and classes that the user menu method returns, the drop down menu can be created in a single line of code.

```javascript
$("#action-menu-0-menubar").click(function(){
  $("#action-menu-0-menu").slideToggle("slow");
});
```

*Figure 4-12. The jQuery function for the user menu drop down list.*

Figure 4-12 shows the JavaScript code for the drop down menu. Figure 4-13 shows the result. The CSS for the user menu is also altered, making the dropped down menu hidden from the start.

4.3. Short on the method of cloning a theme

4.3.1. Cloning

The method of creating a theme in chapter 4.2 was to begin from scratch, and create every folder and file as they were needed. Another way of creating a theme is to start with the files and folder of another theme and then possibly extend the looks and functionality of that theme. This process is called cloning a theme.
4.3.2. Cloning the folder

Using the STRT as a base for the cloned theme there are a number of steps that should be taken. The first step is to make a copy of the theme, which in this case is the STRT. Now there should be a copy of the STRT directory. After that the new theme needs a new name and it is important to only use lowercase letters and underscores. In this example it is named `clone`.

In the new clone theme, the files and directories will of course be identical to the STRT. Before the theme can be recognized by the system and thus used, it is necessary rename all those instances where the theme name occurs.

4.3.3. Rename

The clone theme directory is as mentioned a copy of the STRT, which we explored in chapter 3.6 Design of the Standard Responsive Theme. There are a number of instances where the theme name must be changed.

Within the `lang` directory, as discussed in chapter 3.5.4.3, is the `en` (English) directory which have a file named after the STRT. This file is renamed accordingly and the language strings that the file contains also needs an update. The file contains strings that the theme and the system can use depending on the language of the user. One of the strings is a reference to the theme name. This reference is changed from STRT, to the new name clone. The change is seen in Figure 4-14 below.

```
$string['pluginname'] = 'clone';
```

*Figure 4-14. The name change in the lang directory.*

There are two more places that require changes and those are the `config.php` file, and the `version.php` file. In the `config.php` file, the name option should be set accordingly to the new theme name, and in version there is a reference to the name of the theme that is used by the system for general information about the theme, and this also needs to be changed. The two name changes are seen below in Figure 4-15 and Figure 4-16 respectively.
With the name changed, the new theme can be recognized by the system and is ready to be used as a standalone theme. Since the theme is an exact copy of the STRT except for the recently mentioned name changes, the theme will look and behave just like the STRT.

4.3.4. CSS
The CSS structure of the new clone theme has was briefly discussed in chapter 3.5.1.3, as it is the same as the STRT structure. Changing the style of the clone theme is done with these files and so it is a good idea to get acquainted with the structure of the files and what goes where. Since the STR theme has the bootstrapbase (chapter 3.6) as a parent, the CSS files that the parent theme have, also applies to the STR theme, and thus the clone theme. There is however options that will allow the theme designer to discard chosen style sheets from the parent theme, so if it should be an annoyance it could be dealt with.

The way to finding the bit of CSS code corresponding to the element in need of change can be a challenge if the entire system is new. The names of the style files and the class names and ids’ does not always give away their purpose.

Using a tool such as the developer tool in the Firefox browser or google chrome, which can be used to display the HTML code of a page, will make the process easier. Using the inspect element function on the theme and highlighting the element that should be styled, it is possible to see both where that particular CSS file for that element is currently at, what its name is, and the class names and ids’ of the element and other elements around it.

Most of the general CSS rules are in The Core style sheet. Most of the changes here are through the whole theme as for example. In Figure 4-17, the HTML elements headings and links and their color styles are shown.
Changing the color of the headings can be done by changing the color value to a new value. That will change the color of the HTML headings elements. These are example of the easiest changes that are possible, which only have a minimal effect on the theme as a whole.

However, CSS is a powerful tool and with enough changes in the style sheets, it is possible to change the theme into something that is quite far away from the STR theme that was the starting point. Though, if some element should be moved around on the resulting page, the layout files could be a good way to start.

Using the standard responsive theme to create a new theme is possible by just coding in CSS if there is no need to remove or add new elements, but just changing the CSS of the standard responsive theme would not make it look like the theme created from scratch, so changes in the layout is also necessary, making changes in the layout is more advanced than CSS.

4.3.5. The layout files
The layout files that come with the STRT were discussed in chapter 3.5.1.4. The main page layout is the one with the header, columns and footer. If there should be changes like changing where the headers should be located, the easiest way of accomplishing this is moving the corresponding HTML in the layout files and changing the CSS correspondingly. For example, changing the theme, so that it becomes more like the Samsung backstage theme, seen in chapter 3.2.1, small changes in the CSS files will not be enough. This would require extensive redoing of the structure of the layout files.

The page layouts used by the clone theme can be located in the layouts folder and seen referenced in the config.php file. The clone theme defines a number of pages which have references a theme specific layout. Most page layout files however are not referenced in the
clone theme config.php file. Those page layout types are defined in the parent theme, which of course is the same as the parent of the STR theme, the bootstrapbase theme (chapter 3.6).

4.3.5.1. Custom renderer

The renderer used by the STR theme, has some of its methods overwritten to produce specific HTML. The methods are basically copies of the standard methods, with some class names changed, no core content is changed. The CSS that needs to be written in the clone theme, to change the elements produced by the output renderer, will therefore differ from the created theme.
5. Results

5.1. Introduction

In chapter 4, the method of creating a theme from scratch was explained. The most important workings of Totara themes were explained step by step, as the theme creation was described. Shortly mentioned was also the method of cloning a theme, using the STRT. The two approaches to making a new theme in Totara can give the theme creator some different results, both in expected results and in how the work will be conducted. The amount of effort and time consumption is also greatly influenced by the method, and the desired result might be a determining factor when choosing which approach should be taken.

In this chapter the resulting Totara theme that was created from scratch will be looked at, as well as comparing this approach to cloning a theme. The two ways of going about to create a theme, what the advantages and disadvantages are, and how much time and effort is necessary to create the themes are going to be compared.

The chapter ends with a general criticism of theme framework code for themes in Totara.

5.2. The created theme

As mentioned in chapter 4.1, there was no concrete specification on what the results of the theme would be. The resulting theme is therefore a representation of the general approach that was taken from the start. The theme is not directed at any particular business or educational institution. The look and also the function of the theme are reflected by this as it has some general purpose functionality and looks. The main focus when it came to styling was the desktop environment. Even though a substantial effort was made to create a usable mobile interface, the main concern was with desktops, since it was thought of as the main device that access LMS types of pages [39].
The created theme can be seen in Figure 5-1. The top of the page is used for navigation through both user related menus and general Totara menus. The middle section below the navigation is where the main content, such as course material, goes. As seen in the picture, the block sections that contain all the block elements, is revealed when the menu button is pressed. This in turn darkens the rest of the page so that the user focus lands on the menu. There are functions for quickly getting up to the top of the page if one begins to scroll down, and also to focus on the main content.

5.2.1. Comparison to the STRT
One clear difference between the created theme and the STRT is that, the content that is directly visible has been reduced. The STRT includes up to three columns of different content displayed at the same time, while the created theme focuses on presenting the
actual main content of the site and hides away most of the information that does not require attention all the time. The upside of this is that the main content of the page is in focus. Thus it can be easier to focus on the material that should be focused on, and not get distracted by other content. Menus and navigation are easily accessible though not visible. The downside can be that the user does not find what it is they are looking for, if it is not presented right in front of theme. The STRT shows all content directly, which gives the user all information at once.

5.2.2. The resulting structure

The structure of the created theme aims to bind together the layout files and CSS files in a logical way, and thus make further development easier. Using an approach where each of the output methods gets their own CSS file, changing both the location and the looks of the output of a method is as simple as moving it in the layout file and change the style rules in the corresponding CSS file.

The JavaScript files were also divided in a similar way so that it would be easy to see what functions where used with what elements, only by looking at the file names. This also makes functionality easy to manage and change if needed. Compare this with the STRT, which uses built in JavaScript functions from the Totara system, so the dynamic functions are harder to manage. Functions for hiding and showing menus for example, are not located in the theme folder but in other locations.

5.2.3. Problems

Problems that were encountered when creating a theme from scratch were first of all, the difficulty of overwriting the renderer in an easy and aesthetic way. This led us to bypass the entire process of overwriting the renderer completely and using the existing results given by the old renderer for the theme.

The other big problem was that some elements in the Totara system did not function properly in the newly created theme. These were elements that used the YUI framework extensively, and there were no good solution found, but in the end, the newly created theme had to make the bootstrapbase theme its parent to overcome this problem. The goal
from the beginning was to make the newly created theme a standalone theme, without parents, but this did not work out.

5.3. Difference in approaches

By creating a theme from scratch, there will be no resources in terms of code to start from, which could lead to a shaky beginning until a structure starts to form out. This is obviously not a problem when creating a theme by cloning as there already exist a structure to build upon. This is both an advantage and a disadvantage for cloning.

Learning to develop a new theme for Totara can be hard if a person never had any kind of experience with themes in Totara. The first thing that should be done is to study the STRT which gives a good overview of how themes function. This is the best way of learning how Totara themes function and should be the starting point for any new theme developer.

The conclusion of this is that the best way of learning how to write Totara themes is by analyzing existing themes extensively, before any development starts.

5.3.1. Product

If the end product is not going to stray far off from the standard responsive theme, meaning it is going to use the same basic page layout as an already existing theme, but want to personalize it, then cloning is by far the best option. Because all that might be needed is some CSS, to change some colors and minor details to reach the desired result.

If however the result should be something completely different from any themes that can be cloned, depending on previous experience, starting from scratch might be a just as good or even better approach.

Starting with nothing, means that the structure of the theme, in terms of files and folders can be planned and laid out from the beginning. A cloned theme already has a structure that might not coincide with how the theme should be, and thus be more of an obstacle than a help. Also if the theme created should be part of a system which uses a completely different set of contents than any existing theme, the only way of getting a good design could be do start from scratch.
5.3.2. Time

Time is another important aspect to consider, when choosing the appropriate approach when making a new theme. Cloning is generally much more time efficient, since all the ground structure has been laid out for the developer. Every page layout type has a fitting layout file, and most rudimentary HTML elements that are part of the system, are being accounted for in terms of styling.

Even though cloning is generally faster a lot of time could be spent just navigating the different files to find the piece of code which needed change, and some of the themes could do without a lot of the code which comes from cloning the standard responsive theme.

5.4. The Totara framework

Having used the Totara framework to build a theme from scratch, one major conclusion is drawn: It could be a lot simpler.

First is the YUI library. The YUI library is used throughout the system and handles a lot of functionality. It also handles some styling of elements and adds unique ids’ to some HTML elements throughout the resulting page. This ends up being confusing, when the theme creator does not want to use the YUI library. Further no way was found, that would remove the unnecessary YUI mess, without making some part of the main content unusable. Good thing is that the Moodle community has decided to move away from using the YUI framework and Totara will most likely do the same.

Totara is based heavily on Moodle and both are open source projects. This means that anyone can contribute to the code and add their particular coding style in the mix. Moodle has been in development for a long time, and under that time the main programming language of Moodle, PHP, have also undergone changes. One in particular is the use of classes and objects. There are a number of global functions that are used in Moodle and there are also a number of global objects. It can be argued that choosing one programming paradigm and sticking to it, is better than mixing things, to a degree where there seems to be no actual coding convention.
In terms of themes, the part that should be refactored is the output renderer. On the Moodle website where development is discussed, there is an explanation of what an output renderer does [40]. There, it is stated that the output renderer should not contain any logic other than what is required to generate the displayed HTML. It is arguable, that this is not the case. The output renderer does output HTML but many methods are long, full of conditions that use functions and object methods in an unholy mix that could be better placed outside of the output method. The logic behind what content that should be created should be dealt with somewhere else, not in the output renderer.

In the book clean code by Robert C. Martin[43] for example, the principle behind classes is that they should only have one responsibility, the so called single responsibility principle. That is, they should only care about one thing. The output renderer should only care how things are being returned in terms of HTML, not what is being returned. All conditions and functions used to get the content should be stored away in some other class, which then can be used by the output renderer.

This could greatly increase the flexibility offered by a theme developer as they should not be really be concerned with anything else, but the resulting HTML, CSS and JavaScript. If a theme developer would want to overwrite an output renderer, it should be easy. As it is right now, one cannot purely change the HTML, without considering lower level system functions that are unrelated to the theme creation, since these functions are part of the output renderer.
6. Conclusion

6.1. Introduction

Based on the results of the work conducted in this project paper and looking towards the future in regards to the development of Moodle and Totara, the only time that creating a theme from scratch is applicable, is when there are significant insight into the LMS system and sufficient experience of creating Totara themes. The general outlook is that the best way of creating a new theme is by cloning an existing theme. This could perhaps be generalized to most software development, in that a programmer inexperienced with a system can benefit greatly by starting out with a set blueprint, whilst an experienced one might find more freedom in a fresh start.

6.2. Future development

Since the development of Moodle 3.2 released in December last year, a new theme called clean with the sole purpose of being used as a starting point to create new themes [41]. This has been created in a way to have as little code as possible and only the essentials needed to create a new theme, which gives the developer a solid base, but not too much other than the essentials.

It is very likely that the future version of Totara incorporates a theme with similar purpose as the clean theme in Moodle or multiple themes, so that the developer will have multiple starting points to choose from. This will make creating a new theme from scratch a bad choice as it will only take more time to create a new theme, and most companies which does some kind of work with coding, probably wants the quickest solution.

Big companies might have resources to create their own version of a clean theme that they can use a starting point for their theme projects. Most small companies however, will not have the time or resources to create something similar and will use the STRT, or create one from scratch.

This means that the results of this papers’ main point will not be that useful to any future independent theme developer, but there is still some parts which can still be used. The case
study, regarding the STRT in chapter 3.5, for example which can be a good starting point for new Totara theme developers for understanding how themes works in Totara.

6.3. Project Evaluation

6.3.1. General problems

The biggest problem we had with the project was that there was not a distinguishable problem to solve in the project specification. The assignment given was in essence, that we should study the STRT and later create a new theme from what we have learned, but there were no end result stated clearly, as to what the theme should be like and what specifications that it should fulfill.

So there were no questions we could answer for the purpose of the paper. This created a situation where a problem to solve had to be made clear, based on a small amount of knowledge about the system at hand, which changed the direction of the paper. The importance of clearer specifications with clearly stated end results cannot be understated.

Also making sure that, resources that were necessary for easier development should be one of the first priorities. Now the development of the theme was painstakingly slow, due to the fact that all the new changes had to be uploaded to a server before they could be seen. This meant that each incremental change that was made had an upload time associated with it.

The starting competence when it came to design and web development were somewhat limited, but the main issues revolved around understanding the Totara framework. Help with knowledge about the inner workings of the system, and the resources that were given for development could have made a big difference in startup and development time. Most of the things learned came from trial and error and studying the STRT.

6.3.2. Time

The biggest time thief in this project has been designing. Writing good CSS code to get a desired result is not always easy, further, the desired result might turn out to be a bad design choice, due to artistic shortcomings. This led to many hours tweaking the rules, trying
to get the design to look reasonable. The time that was estimated, in terms of designing and styling the theme was no way near the actual result.

Going from CSS to the layout and renderer showed to be a considerable challenge, as coding for those parts were much harder than expected and a lot of time was wasted just trying to figure out how some parts worked, and we often got stuck. With no expertise at hand, we had to solve the problem in the simplest and most time efficient way to stay on course with the time schedule. Some of these problems took a considerable amount of time, which slowed down the writing of this paper to a crawl sometimes and made the entire project be behind schedule most of the time.

6.4. Closing words

The project has been a learning experience. It was an insight into the business of LMS which clearly have a bright future. The experience with web design and the problems that arise with coming up with designs and finding their solutions in code, was both fun and painful. Even though there was a theme created for this project, it can be argued that web design should be done by people that have an artistic side. The author of this paper feel like this is missing in their arsenal of competences, considering the resulting look of the theme.
References

[15] “Learning Management System Market by Application, Delivery Mode (Distance Learning and Instructor-Led Training), Deployment (On-Premises and Cloud), User Type (Academic and Corporate), Vertical, and Region - Global Forecast to 2021”
[31] Developer Survey Results – 2016. Most Popular Technologies per Dev Type, Full-stack
[33] YUI. https://docs.moodle.org/dev/YUI, January 2017
[38] JQuery datepicker. https://jqueryui.com/datepicker/, January 2017
[40] https://docs.moodle.org/dev/Overriding_a_renderer, January 2017
[41] https://docs.moodle.org/dev/Moodle_3.2_release_notes, January 2017
A. Appendices

Abbreviation list

LMS - Learning Management System
CSS - Cascading Style Sheets
HTML - Hypertext Markup Language
RTL – Right To Left
STRT – Standard Totara Responsive Theme
Moodle – Modular Object-Oriented Dynamic Learning Environment
SaaS – Software as a service

Page Types

In Table A-1, is the complete number of page types and their description which can specify different layouts for a theme.

<table>
<thead>
<tr>
<th>Page type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>The layout used per default.</td>
</tr>
<tr>
<td>standard</td>
<td>Standard layout which should be used with most pages with general information.</td>
</tr>
<tr>
<td>course</td>
<td>Course main page layout.</td>
</tr>
<tr>
<td>coursecategory</td>
<td>Layout for course listings.</td>
</tr>
<tr>
<td>incourse</td>
<td>Layouts used for areas within a course.</td>
</tr>
<tr>
<td>frontpage</td>
<td>The home page layout.</td>
</tr>
<tr>
<td>admin</td>
<td>Administration page layout.</td>
</tr>
<tr>
<td>mydashboard</td>
<td>The users dashboard layout.</td>
</tr>
<tr>
<td>mypublic</td>
<td>The users public profile layout.</td>
</tr>
<tr>
<td>login</td>
<td>The layout for the login page.</td>
</tr>
<tr>
<td>popup</td>
<td>Used for popup windows.</td>
</tr>
</tbody>
</table>
Themes in Totara

frametop | Layout for outermost content of a page constructed with frames.
--- | ---
Embedded | Page layout that are emebedded, like an iframe.
maintanence | Used during upgrade or installation.
Print | Layout that should be used when printing a page.
redirect | Used for redirect layout.
report | Used for report layouts.

### Theme options

There are a number of different options available for a theme. These are specified in the `config.php` file within the theme. Table A-2 lists all the possible options that can be specified along with a description of their function.

**Table A-2. The options available in the config.php file within the theme folder**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parents</td>
<td>A list of parent themes</td>
</tr>
<tr>
<td>sheets</td>
<td>The names of all style sheets, given without file extension, that should be included. The order given is the order that they will be included.</td>
</tr>
<tr>
<td>parents_exclude_sheets</td>
<td>A list of all style sheets that should not be included from parent theme.</td>
</tr>
<tr>
<td>plugins_exclude_sheets</td>
<td>A list of style sheets from plugins to exclude.</td>
</tr>
<tr>
<td>editor_sheets</td>
<td>A list of style sheets that are used by the text editor body.</td>
</tr>
<tr>
<td>javascripts</td>
<td>A list of JavaScript files, without file extension, to be included. Files should be located in the themes' JavaScript folder by default.</td>
</tr>
<tr>
<td>javascripts_footer</td>
<td>The same as the javascripts option, but files are referenced in the footer of the page, thus reducing the page load time.</td>
</tr>
<tr>
<td>parents_exclude_javascripts</td>
<td>JavaScript files that should be excluded from the parents.</td>
</tr>
<tr>
<td>layouts</td>
<td>A list of the layout files that correspond to their page type. The layout file is located in the layout</td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rendererfactory</td>
<td>Name of the renderer factory class that should be used to overwrite a output renderer.</td>
</tr>
<tr>
<td>csspostprocess</td>
<td>Specifies a string function for custom CSS post-processing, if this should be used.</td>
</tr>
<tr>
<td>rarrow</td>
<td>String that is used as arrow-character in menus and navigation. Default character is used if none is specified.</td>
</tr>
<tr>
<td>larrow</td>
<td>Same as rarrow but different direction arrow-character.</td>
</tr>
<tr>
<td>enablecourseajax</td>
<td>Boolean value for enabling ajax course editing. Default is true.</td>
</tr>
<tr>
<td>doctype</td>
<td>Determines the doctype of the pages served to the user. The default doctype is html5.</td>
</tr>
<tr>
<td>name</td>
<td>Simply the name of the theme.</td>
</tr>
<tr>
<td>dir</td>
<td>The folder where the theme files are stored. Is set automatically</td>
</tr>
<tr>
<td>enable_dock</td>
<td>Boolean option whether or not to allow docking of block elements.</td>
</tr>
<tr>
<td>hidefromselector</td>
<td>If true, the theme will not be shown in the theme selector unless theme designer mode is off. Default if false.</td>
</tr>
<tr>
<td>yuicssmodules</td>
<td>An list of YUI CSS modules to be included in each page.</td>
</tr>
<tr>
<td>blockrtlmanipulations</td>
<td>An associative array of block region manipulations that should take place if the user is using an RTL language.</td>
</tr>
<tr>
<td>supportscssoptimisation</td>
<td>Boolean value that indicates wether or not the themes' CSS can be run through an optimiser. Default is true.</td>
</tr>
<tr>
<td>lessfile</td>
<td>Boolean value indicating if there is a less file that should be compiled before serving the page. This less file should be in a folder named less, in the theme directory. The default is false.</td>
</tr>
<tr>
<td>extralesscallback</td>
<td>Name of a function which gets the less code to inject. Default is null.</td>
</tr>
<tr>
<td>lessvariablescallback</td>
<td>Name of function which gets extra less variables. Default is null.</td>
</tr>
</tbody>
</table>
blockrendermethod

| blockrendermethod | Sets a render method that should be used for rendering custom block regions. The default is blocks_for_region. |

Source code for created theme

Below is the source code for the config.php file, the JavaScript functions and layout file.

```php
<?php

$THEME->name = 'create';
$THEME->yui_css_modules = array('combo');
$THEME->parents = array('bootstrapbase');
$THEME->parents_exclude_sheets('moodle', 'editor');
$THEME->sheets = array('standard', 'login', 'general', 'blocks', 'colors', 'menu', 'user_menu', 'modules');

$THEME->layouts = array(
    'base' => array(
        'file' => 'standard.php',
        'regions' => array(),
    ),
    'standard' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
        'defaultregion' => 'block-menu',
    ),
    'course' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
        'defaultregion' => 'block-menu',
    ),
    'coursecategory' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
        'defaultregion' => 'block-menu',
    ),
    'incourse' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
        'defaultregion' => 'block-menu',
    ),
    'frontpage' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
        'defaultregion' => 'block-menu',
    ),
    // Server administration scripts.
    'admin' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
        'defaultregion' => 'block-menu',
    ),
    // My dashboard page
    'mydashboard' => array(
        'file' => 'standard.php',
        'regions' => array('block-menu'),
    ),
);`
Appendices

Figure A1. The config.php file.

```php
'defaultregion' => 'block-menu',
'options' => array('langmenu'=>true),
),
// My public page
'mypublic' => array(
  'file' => 'standard.php',
  'regions' => array('block-menu'),
  'defaultregion' => 'block-menu',
),
'login' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'popup' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'frametop' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'embedded' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'maintenance' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'print' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'redirect' => array(
  'file' => 'general.php',
  'regions' => array(),
  'options' => array('langmenu'=>true),
),
'report' => array(
  'file' => 'general.php',
  'regions' => array('block-menu'),
  'defaultregion' => 'block-menu',
),
'secure' => array(
  'file' => 'general.php',
  'regions' => array('block-menu'),
  'defaultregion' => 'block-menu',
),
'noblocks' => array(
  'file' => 'general.php',
  'regions' => array('block-menu'),
  'defaultregion' => 'block-menu',
  'options' => array('langmenu'=>true),
),
);

$THEME->rendererfactory = 'theme_overridden_renderer_factory';
$THEME->javascripts_footer = array('menu');
$THEME->javascripts = array();

?>


$("#action-menu-0-menubar").click(function(){
  $("#action-menu-0-menu").slideUp("slow");
});

$(document).scroll(function(){
  var scroll = $(document).scrollTop();
  var back2top = $("#back2top");
  if(scroll != 0)
  {
    back2top.css("display", "block");
  }
  else
  {
    back2top.css("display", "none");
  }
});

$("#back2top").click(function(){
  $("html, body").animate({scrollTop: 0 }, 100);
});

$(".totara-menu > ul > li").hover(function(){
  $(this).children("ul").slideDown(100);
});

$(".totara-menu > ul > li").mouseleave(function(){
  $(this).children("ul").slideUp(100);
});

var enlarged = false;

$("#enlarge-button").click(function(){
  if(!enlarged)
  {
    $("#focus-wraper").css("display", "block");
    $("#focus-wraper").css("z-index", "3");
    $("#region-main").css("z-index", "4");
    $("#region-main").css("position", "relative");
    enlarged = true;
  }
  else
  {
    $("#focus-wraper").css("display","none");
    enlarged = false;
  }
});

$("#focus-wraper").click(function(){
  $(this).css("display", "none");
  enlarged = false;
  if(blockMenuVisible)
  {
    $("#block-menu").animate({width:'toggle'},350);
    blockMenuVisible = false;
  }
});

var blockMenuVisible = false;

$("#showblock").click(function(){
  $("#block-menu").animate({width:'toggle'},350);
  if(blockMenuVisible)
  {
    blockMenuVisible = false;
  }
});
Figure A2. The JavaScript functions responsible for dynamic functionality
Figure A3. The general layoutfile file.