Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Architecture and Design
Department of Design

Lars Erik Lunde
Marcus Leikfoss Swensen

Gurisentret 3D Print

Bachelor's project in Bachelor in Web Development
Supervisor: Ivanna Baturynska
Co-supervisor: Oleksandr Semeniuta

May 2021

NTNU
Kunnskap for en bedre verden

Lars Erik Lunde
Marcus Leikfoss Swensen

Gurisentret 3D Print

Bachelor's project in Bachelor in Web Development
Supervisor: Ivanna Baturynska
Co-supervisor: Oleksandr Semeniuta
May 2021

Norwegian University of Science and Technology
Faculty of Architecture and Design
Department of Design

NTNU
Kunnskap for en bedre verden

ABSTRACT

Title: Gurisentret 3D Print
Date: 14.05.21
Participants: Lars Erik Lunde & Marcus Leikfoss Swensen
Supervisor: Ivanna Baturynska & Oleksandr Semeniuta
Employer: Smøla Municipality & NTNU AddLab
Contact Person: Heidi Rognskog Mella
Keywords: 3D printing, learning platform, web development, website
Number of pages: 51
Number of attachments: 7

This project is the bachelor thesis in Bachelor in Web Development at Norwegian University of Science and Technology, provided by Smøla Municipality Nærings- og Kultursenter KF in collaboration with NTNU AddLab. Smøla municipality is currently remodeling their museum, Gurisentret, and in this context they want a digital solution where their visitors, primarily school children, can engrave a text of their choice in a 3D printed model of a monumental stone. This report will describe in detail how the solution “Gurisentret 3D Print” was created through research and development.

SAMMENDRAG

Tittel: Gurisentret 3D Print
Dato: 14.05.21
Deltakere: Lars Erik Lunde & Marcus Leikfoss Swensen
Veileder: Ivanna Baturynska & Oleksandr Semeniuta
Oppdragsgiver: Smøla Kommune & NTNU AddLab
Kontaktperson: Heidi Rognskog Mella
Stikkord: 3D printing, lærings platform, web utvikling, nettside
Antall sider: 51
Antall vedlegg: 7

Dette prosjektet er gitt av Smøla Kommune Nærings- og Kultursenter KF i samarbeid med NTNU AddLab og er det avsluttende prosjektet i Bachelor i Webutvikling (BWU) hos Norges Teknisk-Naturvitenskapelige Universitet. Smøla kommune bygger om Gurisentret og ønsker i den sammenhengen å få en digital løsning der deres besøkende, i hovedsak skolebarn, skal kunne inngravere en egenvalgt tekst på en 3D printet modell av en bautastein. Denne rapporten vil beskrive i detalj hvordan løsningen “Gurisentret 3D Print” kom til live gjennom innsikt og utvikling.

FOREWORD

First of all, we would like to thank Smøla Municipality for the amazing engagement and interest in our project. It has been an exciting and challenging project. Thank you to Heidi Rognskog Mella from Smøla Municipality. She has always been positive and encouraging throughout the project and always curious about our progress and showed interest for our field and technologies, something we have greatly appreciated.

We would also like to give a special thanks to Ivanna Baturynska & Oleksandr Semeniuta from NTNU AddLab for their great cooperation and guidance during the project. You have given us valuable insight into the world of 3D printing and 3D in general.

Lastly, we would like to thank all the amazing people that wanted to contribute through the user tests for their valuable thoughts and insights that helped improve our solution.

Gjøvik, 14.05.21

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Background of the Project

This bachelor thesis is the final assignment in Bachelor in Web Development (BWU), which is to be delivered by the end of spring 2021. The project is provided by Smøla municipality Nærings- og Kultursenter KF in collaboration with NTNU AddLab. Smøla municipality is currently remodeling their museum/cultural center, "Gurisentret", which is set to reopen spring/summer 2021. When finished, it will present history and cultural legacy in a new and contemporary way using modern technology. Here cultural monuments will be displayed and made available for locals and tourists that are interested in exploring, learning and sharing knowledge about our cultural legacy. The goal for Gurisentret is to educate future generations on their identity and cultural heritage as well as create a new arena for preserving and conveying history (Gurisentret, 2019).

One of the new presentations is a project for 3D printing a scaled version of the monumental stone "Kuli stone" (NTNU, n.d.). This is a historic monumental stone, with the oldest finding of writing mentioning the word "Norge" and the christening of Norway from the early 11th century. This stone was modeled in 3D in 2019 and it's the model that will be printed at Gurisentret, as well as one other monumental stone. Gurisentret would like its visitors, primarily school children, to write their own inscriptions in runes on a 3D model, then have it 3D printed to bring back home as a souvenir. By letting school children and other visitors make 3D printed models of monumental stones with runes inscriptions, they hope to achieve a fun and engaging way to learn about history, culture, runes, monumental stones and, in particular, the Kuli stone. They will also be able to paint and write on the monumental stones after they are printed to give it their own personal touch. Smøla municipality was inspired by Teknisk Museum in Oslo where they have a similar solution for children to 3D print different objects.

Smøla municipality has also become a part of "Skaperverkstedet", an organization that wants to convey how technology and programming can be used in a creative way across multiple subjects in school (Voll, n.d).

1.2 Description of Goals

The goal of this project is to share Norwegian history and culture in an educational and inspiring way through a user-friendly website primarily for school children.

The scope of this project is to create a user-friendly solution where primarily children can choose a 3D model where they can write their name or a sentence in rune text, choose where the text will be positioned and then print it using a 3D printer.

1.2.1 Result Goals

The result goal of the project is to design and create an inspiring and educational stand/presentation for Gurisentret in Smøla Municipality. Here visitors, primarily school children, can write their own inscription in rune text on a scaled version of the historic monumental stone “Kulisteinen”. The inscription will be engraved in the 3D model and printed through a 3D printer as a souvenir from the visit.

1.2.2 Effect Goals

Smøla municipality wants to attract more visitors by using modern technology to present history and culture. Aimed mostly at school children the goal is to inspire and engage children and other visitors to learn more about local history and culture.

The group’s effect goal is to learn new technologies, as well as gathering useful experience on how to work as developers in a real-life work situation.

1.2.3 Research Question

How can we make the most user-friendly interaction for children to be inspired and engaged in Norwegian and local history and culture, by using modern technology through our solution?

1.3 Structure of the Report

This report will present the solution developed for this bachelor thesis. It will give an insight into what has been done in order to create a solution that will engage younger audiences in history and culture, why things have been done, and how they have been done. The report will first go through the methods, research and tools that have been used to investigate, design, and code. From there the final solution will be presented and then discussions around how and why the solution met the goals set for the project, as well as choices made during the process. Finally, there will be a conclusion to the project and further development possibilities.

1.4 Work environment

The group consists of two members: Marcus Leikfoss Swensen and Lars Erik Lunde. We both had a high ambition level to complete this course in the best way possible. Our goal as specified above is to create the best possible solution for the product owners. As the group only has two members, we decided that we did not need to set a fixed work schedule. However, we worked together as needed throughout the project, with a goal of meeting at least 2-3 times a week in addition to working individually at home. Towards the end of the project daily meetings were scheduled.

2. METHODS

2.1 Qualitative Interviews

Qualitative interviews' purpose is to receive in-depth insights from a user, rather than measuring different variables. There are three different types of qualitative interviews, unstructured, semi structured and structured interviews.

Unstructured interviews are interviews where the theme and questions are not defined completely beforehand. This type of qualitative interview is intended to give the interviewer a deeper understanding of the problem. These interviews are often more informal and casual and will often require a large amount of analysis afterwards (Østby, H, *et al*, 2017 p. 104-105).

Semi structured interviews are similar to unstructured interviews where they give the interviewer a large amount of flexibility when it comes to the type of questions asked. What separates the two is that semi structured interviews have a predefined theme. However, it is more open to questions asked based on the respondent's answers (Østby, H, *et al*, 2017 p. 105).

Structured interviews are different from both unstructured and semi structured interviews. Here there are predefined and often open questions. The questions often have a structure of what questions will be asked first and when they are asked. This is usually arranged in a form. This type of interview will often be easier to analyze than unstructured or semi structured interviews (Østby, H, *et al*, 2017 p. 105).

During any type of qualitative interview, it is important to take notes and listen to what the respondent has to say. It is also important not to interrupt the respondent and let them answer freely and not guide them towards a specific answer that the interviewer wants.

2.2 Agile Software Development Methodology

Agile software development methodologies refer to the group of different methodologies based on an iterative development structure. Here the solution iterates over time and encourage adaption based on user feedback. These methodologies are used to create user friendly software

with quick delivery. This project adapted agile development since it requires user input and changing the solution based on feedback. This ensures the best possible user experience and provides simplicity for the end user (Cprime, n.d).

2.3 Iterative Design

Iterative design is a design methodology where you gather information through user research and generate ideas to create and test a prototype. From there you take the information gathered from the user testing, change the design and test again. The purpose of iterative design is to create a more economical design and development process. It is often a lot easier and less expensive to change the design often, than to develop a product and realize later on changes are required. Conducting user research before development often only provides an indication on what the user wants. However, what they want might not end up corresponding with what is developed. Therefore, it is important to include the user in the design and development process over several iterations to make sure the finished product is what the user actually needs (Interaction Design Foundation, 2021) (Nielsen, 1993).

2.4 Brainstorming

Brainstorming is a design method where the goal is to come up with a multitude of new ideas. This method contains multiple rounds of iterations where you move closer and closer to a final solution. It contains two phases, in the first phase you come up with as many ideas as possible. In the second phase you make choices of which ideas to use and may combine several ideas if possible. This method is an efficient way to solve problems you may have, increase the number of ideas, and produce ideas quicker (IDEO, n.d *A*).

There are several rules for brainstorming. These are to defer judgement, encourage wild ideas, build on the ideas of others, stay on topic, stick to one conversation at a time, be visual and go for quantity. These rules are in place to make sure you get the best possible outcome of the brainstorming session (IDEO, n.d *B*).

3. RESEARCH AND TOOLS

3.1 Research

The research on the project was mostly carried out through interviews and obtaining information from the product owners. Over several meetings, information about the background of the project and several requirements to the solution were obtained using unstructured and semi structured interviews in meetings with the product owner. These interviews and meetings were conducted over zoom because of the coronavirus and due to the product owner being located in a different city. The information obtained in these meetings laid the groundwork for the initial design and further development of the solution. After the information was received through the interviews and meetings, there was a brainstorming session to generate different ideas to solve the problem and create an optimal final solution.

3.2 User testing

User testing is a useful tool to retrieve feedback from users about the product. In Jacob Nielsen's article “Why You Only Need to Test with 5 Users” he states that testing with only 5 users will reveal as much as 85% of the problems in a product. Therefore, it is better to only test with five people and then reiterate before conducting more user tests (Nielsen, 2000). Thus, we conducted in total twelve user tests due to covid restrictions. Three user tests were conducted on the initial design, four on the development build and five on the final build.

User testing was done in three phases, initial design, development build and final build. Since the solution was supposed to be used by everyone, every phase was tested on other students. The final build was additionally tested on some users in the main user group. Because of the Covid-19 virus and a lockdown we were unable to test the initial design and the first build on the main user group. All test subjects were given a task to choose a specific model, write a text of their choice and send the order to print. After each user test the test subjects were asked a couple of questions about the design and user experience. All of the iterations were modified based on the feedback from the users. Furthermore, the solution and design was also tested on two interaction designers. This was conducted to ensure the best possible user experience. The product owners also had access to the development build and based on some of their feedback, changes were made for the final build, especially color choices and parts of the layout. Some

user tests were conducted on the admin site which provided useful feedback that led to significant changes. No personal information that can be linked back to a specific person, was gathered during the user testing. Therefore, no approval from NSD[1] was needed.

3.3 Tools

In order for us to work efficiently as a group, a variety of different tools were necessary. This section will give insight into the tools that were used, how they were used and why they were used.

3.3.1 Collaborative tools

We decided on using different collaborative tools for working together as an effective team. These tools allowed us to work together both physically and digitally, as Covid-19 made meeting in person unpredictable in terms of social restrictions. Therefore, these tools allow us to switch from physical to digital meetings with ease and collaborate regardless of where we are in the process.

Trello

Trello is a web-based, list-making collaboration tool that helps organize the workload of the project. Trello was divided into 5 main sections, TODO - Important, TODO - Less important, Currently working on, Bugs and Completed. This made it easy to keep track of what has to be done and who is working on what.

Figma

Figma is an online, co-operative tool used for creating high quality, digital and interactive prototypes (Figma, n.d). Since the group had previous experience with Figma, this became the tool selected for prototyping. Here both AdobeXD and Sketch were considered. However, Figma is the only tool out of the three that has the capability of real time collaboration. It provides all the necessary functions needed for the prototype to be developed in an easy way. Figma also has an easy-to-use prototype testing function. This made it possible to send the prototype to potential real-world users for user testing, in addition to the product owners for approval of the design.

[1] Norsk senter for forskningsdata

Google Drive

Google Drive is used to store different documents, notes and files relating to the project. Given that Google Drive is free, easy to use and very familiar, this was a simple choice for this project. Google Drive is a cloud-based storage service and therefore it was possible to access all the documents from different locations at any time. This also removed the risk of one computer crashing and therefore losing the whole document.

Google Docs

Google Docs is an online word processor used first and foremost as a collaboration tool to write the bachelor thesis. In addition to this, it is used to collaborate on notes, timelines and more. Google Docs was chosen over word mostly because of its real time collaborative writing. This allows the group to work simultaneously in the same document and see changes in real time.

Microsoft Teams and Zoom

Microsoft Teams and Zoom were mostly used as a platform for digital meetings. When the group needed to work from home, meetings were held over teams to ensure maximum capacity on collaborating and solving issues or problems. In addition to this Zoom was used for meetings with the bachelor supervisor and the product owners as well as during user testing if they had to be conducted online.

GitHub

GitHub is a code hosting platform for version control and collaboration for the coding aspect of the project. The group worked in branches for different features and merged them into the master branch as the project progressed. Branches were used in order to prevent merging conflicts, as it makes it easier to experiment with new features at the same time, without the concern of breaking existing functionality.

3.3.2 Software and coding language

3.3.2.1 Frontend

Vanilla JS/HTML/CSS

The front-end is built using mostly vanilla JavaScript[2] with some jQuery and HTML[3]. Bootstrap is used for most of the layout and complemented by local CSS[4] when needed. Front-end frameworks were considered, but were not selected for this project, see 5.1.

Bootstrap

The CSS framework Bootstrap was used to build the design of the user interface. Bootstrap is one of the most popular CSS frameworks and provides easy to use components for developing responsive websites. In its core it is a huge collection of small bits of reusable code that is written in HTML, CSS and JavaScript and has become an essential part of modern front-end development. Instead of writing large CSS files, Bootstrap allows a developer to add classes to different tags in HTML which applies styling. Bootstrap also comes with hundreds of pre-made components such as navigation bars, dropdowns, progress bars, thumbnails etc. to easily be used and added to the project. These components and classes can also be modified to fit the theme of the project in an easy way. Because of Bootstrap's enormous popularity and open-source code, it has a large community of designers and developers behind it. Because of this, finding solutions to problems that may occur is uncomplicated (Ouelette, 2021).

jQuery

jQuery is a JavaScript library designed to simplify HTML DOM[5] manipulation and event handling. jQuery is a "write less, do more" type of library and provides AJAX[6] which enables web pages to update asynchronously without reloading the whole page.

[2] Using plain JavaScript without any libraries
[3] Hyper-Text Markup Language
[4] Cascading Style Sheets
[5] Document Object Model
[6] Asynchronous JavaScript and XML

Libraries/dependencies

What is node package manager (npm)?

Using Node.js gives us access to npm[7] which is a package manager or build-tool for the front-end. The npm registry hosts over 1,000,000 open-source packages free of use. In this project we are using 15 dependencies or modules. Open source means that the source code is publicly available for others to view, copy, learn and share it (opensource.com, n.d).

Three.js

Three.js is a library for displaying 3D content on a webpage. Three.js uses WebGL to draw 3D. WebGL is a low-level system that draws points, lines and triangles (webglfundamentals.org, n.d). WebGL generally requires a lot of code, but three.js simplifies this by handling scenes, light, textures etc., which WebGL would require you to write yourself.

Orbit control

Orbit control is an instance of three.js and returns an OrbitControls class. Orbit control takes in an object, the camera, and the HTML DOM element which allows users to rotate the camera around a central point, while keeping the Y-axis locked. This prevents the scene from getting "tilted" off-axis. With Orbit controls a user can orbit, zoom and pan using their mouse og touch controls (Three.js, n.d, *A*) (RIP Tutorial, n.d). Orbit control is set to a minimum (150) and a maxim (250) zoom distance from the 3D object and pan is disabled.

STL Loader & Exporter

The STLloader dependency is an instance of three.js that allows us to display 3D models that are STL ASCII files with the .stl[8] file extension. The loader returns a non-indexed buffer geometry (Mrdoob, n.d**).** The STL Exporter exports a three.js scene mesh to STL, making it suitable for 3D printing.

Bcrypt

Bcrypt is a library that provides a method for hashing[9] passwords. Taking a plain-text string as input and outputting a fixed length hashed string to store in the database. Bcrypt also comes

[7] Node Package Manager
[8] Standard Tessellation Language, most commonly used format for 3D printing
[9] Turning a string into a scrambled representation of itself

with salt, which is adding a random string to the hash, which makes the hash algorithms output no longer predictable (Selzer, 2020).

Cookie-parser

Cookie-parser is a dependency that allows us to use cookies with Node.js. It allows us to store data and populate req.cookies() with an object keyed by the name of the cookie. Cookies are a piece of data that the computer receives and sends back to a server without changing its content. A cookie often stores data such as activity, visits, login information etc. (Norton, 2019).

EJS

EJS is a template system and a tool for generating HTML web pages that can include dynamic data and display HTML markup using JavaScript. EJS is used with Express as it hooks into Express's view-engine architecture and can render web pages to the client with res.render() (EJS, n.d).

Path

The path module of Node.js is a npm dependency that provides a way of working with directories and file paths (nodejs, n.d).

Dotenv

The dotenv dependency is a module that loads environment variables from a .env file into process.env to keep secrets from the source code (npm, 2021b). To host our solution on Heroku, the app's web server must listen to a specific port (Heroku, 2021). This port is indicated by the PORT environment variable.

Nodemon

Nodemon is a dependency that is used only during development. Nodemon automatically restarts the node application/server when file changes in the directory are detected. Nodemon is a replacement wrapper for node (remy, n.d).

3.3.2.2 Backend

For the back-end we are using a Node.js web server with express and MongoDB which is used as our database for storing data.

Node.js

Node.js is a JavaScript runtime environment that is built on Chrome's V8 JavaScript engine. Many widely used frameworks employ a Node.js runtime in order to run JavaScript outside of the browser. Node is popular for building backends because it utilizes JavaScript as the base language. This allows developers to develop both the front-end and back-end of web apps using JavaScript, sharing and reusing code across the stack with a single package manager (npm).

Express

Express is one of the oldest and widely used back-end web application frameworks for Node.js. It's a framework to help organize a web application into an MVC[10] architecture on the server side. Express helps manage everything from routes to requests and views/templating.

MongoDB

MongoDB is an open-source, document-oriented database service. Instead of storing data in tables of rows or columns like in a SQL database, each row in MongoDB is a document described in JSON. The documents are in turn organized into collections (Guru99, n.d) (MongoDB, n.d). MongoDB is designed for building web-application in a fast and iterative way and using a NoSQL online database reduces the complexity of deployments. MongoDB is easy to integrate in Node.js via npm and is fast to set up (MongoDB, n.d).

Mongoose

Mongoose is a JavaScript library that lets us define schemas. A Mongoose schema defines the structure of the document, default values and validators. With a schema we can create a model based on the specific schema. A Mongoose model is then mapped to a MongoDB document via the model's schema definition and provides an interface to the database for creating, querying, updating and deleting records. Mongoose also provides schema validation, relationship between data and is used for representing MongoDB objects in code (Karnik, n.d).

[10] Model-view-controller

3.3.3 Prototyping

Figma was used to create the hi-fi prototype, based on conversations and meetings with the representative from the product owner. The first draft created on paper, see attachment 6, was mostly based on the initial information from the project description. However, after talking to the product owners, it became clear that a different design was necessary. Since the product was mostly going to be used by school children, simplicity became a heavy focus of the prototype developed in Figma. The design below is based on the initial design after feedback from the first user testing.

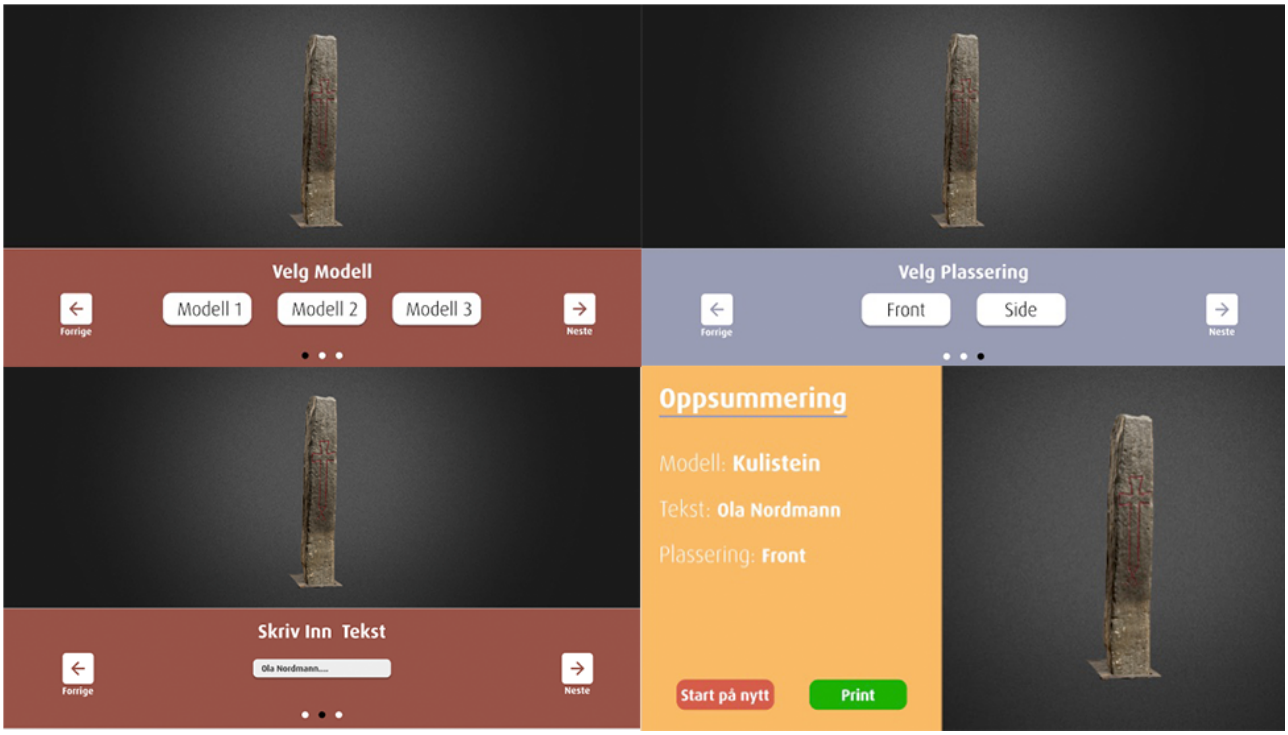


Figure 1, Image of prototype from Figma

4. IMPLEMENTATION/SOLUTION

4.1 Frontend

The product is supposed to be a simple solution for users to choose a 3D model of a monumental stone, write text and choose the position of the text. After this, they receive the 3D printed model with the text engraved.

Figure 2, Final Solution Section 1

4.1.1 Main page

It was very important for the product owners that the user interface was simple for everyone to use, therefore this was the intention behind the design. When you enter the site, you get to choose between two different models to print, as well as type in the text you want engraved in the model. From there the user chooses the position of the text. Since the printing process will take approximately 1 hour per model, the user is able to fill in their contact information on the last page for the museum to contact them when the print is ready. In addition to their contact information the user can choose whether they will pick it up or have it sent by mail. If the user does not complete the process a screensaver will be activated after three minutes. When the solution detects input, the page will reload and take the user to the beginning.

Figure 4, Final Solution Section 2

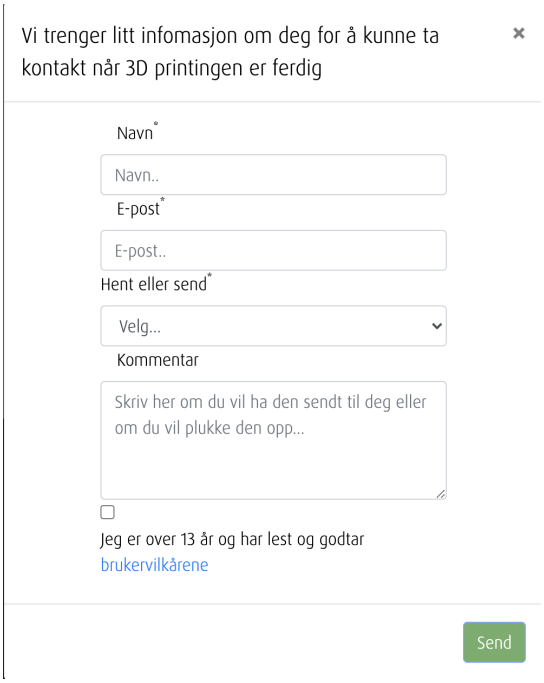


Figure 3, Final Solution Contact Form

4.1.2 Admin page

The 3D model visible to the user is an .stl file which means it has to go through a slicer to generate the .gcode[11] file needed for the 3D printer. Every print is submitted to the admin page, where the admin will have control over every print available. The admin page is divided into three sections. The first section contains the prints that have been submitted by visitors. These are sorted by when they were completed on the user side of the page, to ensure that the first to send in their print will be the first to receive their print. From here the admin user is able to download the model that the user created, see section 4.1.4 Print Page, run it through an external slicer software and print it. To make sure the admin has full control over what model is under printing and who it belongs to they will also have access to a receipt, see section 4.1.3 Receipt Page.

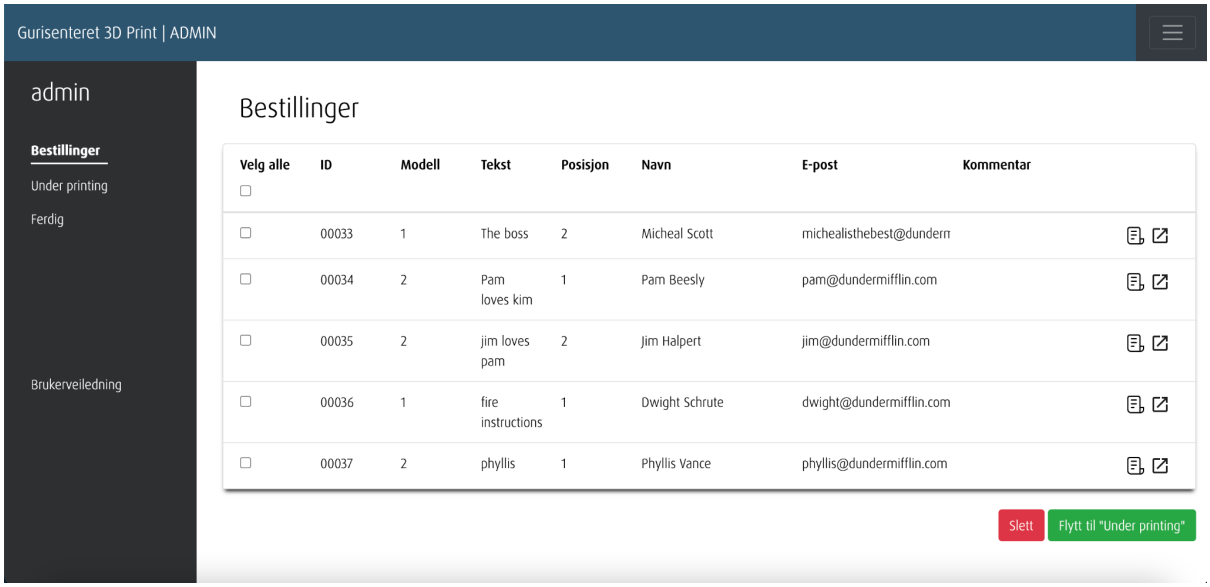


Figure 5, Final Solution Admin Page

After the administrator has downloaded the model and started printing it, they will be able to send that specific object or multiple objects to the next section. This section contains all the models that are currently printing. Here they are still able to print a receipt if that has not been completed yet. When they have successfully completed the print, they can send the prints to the last section.

[11] File type that contains instructions for the 3D printer

The last section of the admin page contains all of the completed prints. When a user picks up their item or the item is shipped to the user, they can delete the print from the database and the process is complete.

On the left side menu, the admin also has access to the user guide which will open as a pdf. This will display a step-by-step guide on how to use the admin page, as well as the correct settings to use on the 3D printer. By the request of product owners, a button to download the .gcode file containing the entire alphabet in runes was added to this menu.

The admin page also has a dropdown menu on the right where the admin can create a user or delete a user. This was created to make it possible for Smøla municipality to be able to create different users for the different people working there. There are a couple of restraints with this, since you currently have to know the password of the user you want to delete, and there is no way of changing password once you have created a user. The admin login is only designed to make sure people using the front-end will not have access to the admin site. More on this in section 4.5 Security.

4.1.3 Receipt page

The receipt page is an overview of all the information the admin requires to identify whom the print belongs to. The page retrieves the information from the database based on what print is clicked on and displays it in a simple form with a printId, name of the model, the text that is written on the model, what the text looks like in runes, where the text is located, any comment the user had, email and name of the user the print belongs to, as well as the information required to ship the print. On the bottom of the page there is a simple print button which allows the administrator to print out the complete page.

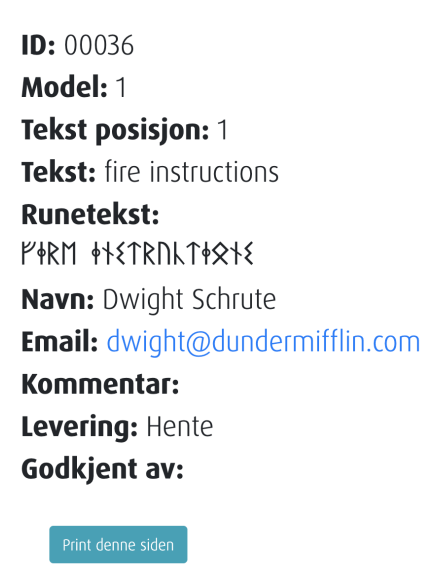


Figure 6, Final Solution Receipt Page

4.1.4 Print Page

The print page is the 3D model's final stop in this solution. Here the admin user can inspect the 3D model with engraved user input text, before downloading it and exporting it to a third-party slicer. The page also provides some information about the print and the visitor whose model it belongs to.

Since the models are based on historic monumental stones with runic inscription, the 3D printed result will have the text engraved in the printed model. To achieve this there had to be performed a CSG operation on our model with the text object. CSG[12] is a technique allowing us to create new surfaces and objects by using Boolean[13] operators to combine objects (Foley, 1996).

Figure 7, Final Solution Print Page

The model and user text are two separate objects. The model is loaded with an STLLoader, see 3.3.2.1, and the text geometry is created with three.js. The two objects are combined using CSG, the text object is then subtracted from the monumental stone model, which gives us the desired result of engraved runic text on the model ready for print.

To perform the CSG engraving operation we are using two open source util files. These files are a CSG library for three.js licensed under the MIT license (manthrax, n.d) (Open Source

[12] Constructive Solid Geometry
[13] A logic that creates true/false statements

Initiative, n.d). The CSG operation is quite processing intensive and led to some problems, see 5.1.5. The models are scaled down and a base is added with three.js. The base is engraved at the bottom with its printId using the same CSG technique as described above.

To download models for print, the print page uses a library called STLExporter, see 3.3.2.1. This takes the scene from three.js and exports a .stl file, which is why the admin user has to open the print page and load in the model before downloading. The download-file is named the same as it's printId.

4.1.5 Printed runic alphabet and numbers

By the request of the product owner, characters in rune text and numbers were created. These standalone characters and numbers will be used by visitors to create words and sentences before using our solution to place it on a monumental stone. This file is available to download through the admin page. This will further promote a fun and engaging way to learn about culture, history and runes.

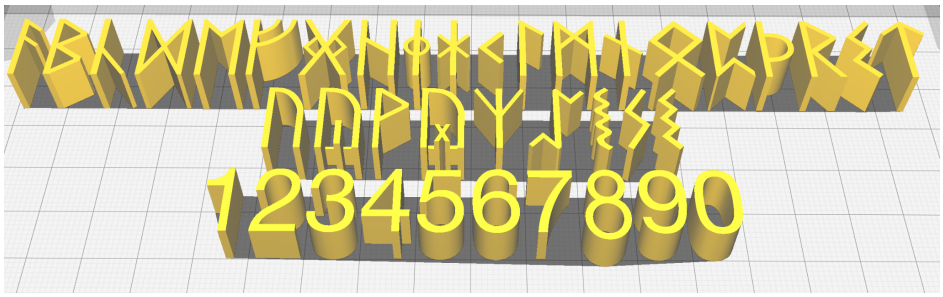


Figure 8, Runic Alphabet

4.1.6 Sitemap

Below is a sitemap describing the layout of the solution, what sections lead where and where they are located on the site. The red areas are publicly available and the green areas you have to be logged in to access. The orange describes login/signup functions, and the blue opens up in a new tab when clicked.

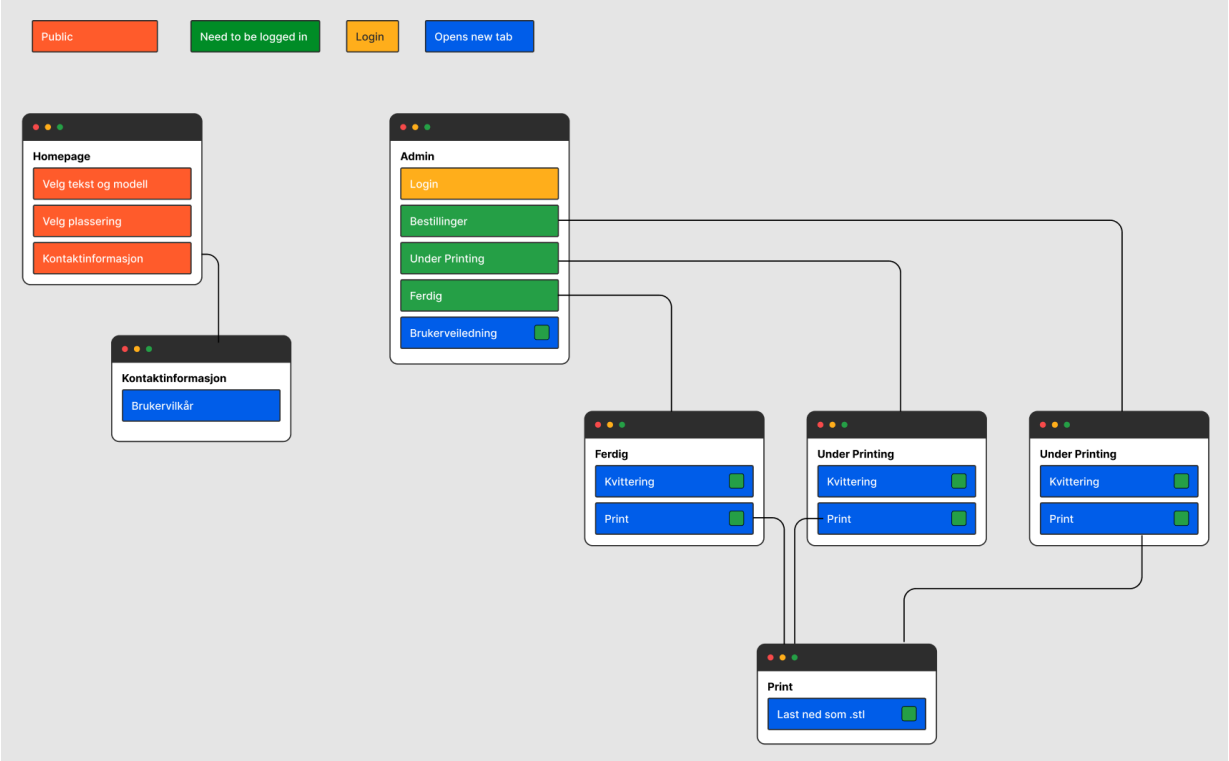


Figure 9, Sitemap, Created in Figma

4.1.7 Three.js

Renderer

The renderer is the main object of three.js. It takes in a scene and draws the 3D scene that is inside of the frustum[14] of the camera. The scene takes in a group used to accumulate all meshes under one directory.

[14] Frustrum, in 3D computer graphics is the region of space in the scene that may appear on screen, see figure 11

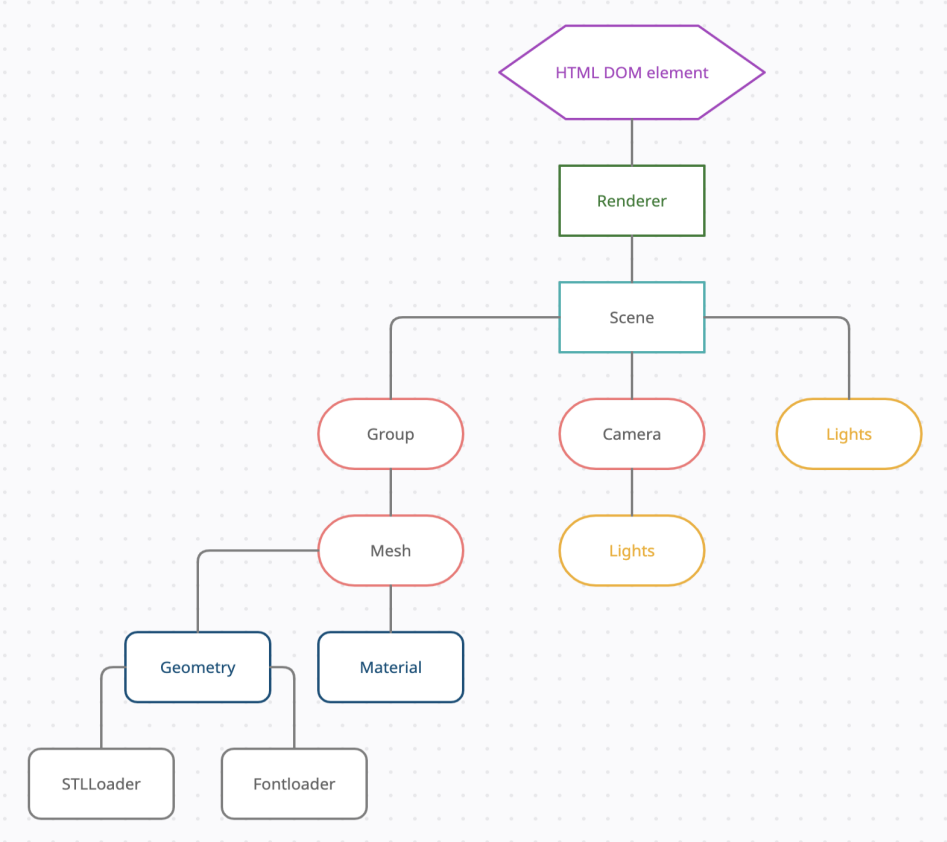


Figure 10, Three.js Structure, Created in Creatly

Scene

A scene enables us to set up what and where things are rendered by three.js. The scene is a container sitting at (0, 0, 0)[15] and is where the object, lights and camera is placed. The constructor for creating a three.js scene is Scene() and takes no parameters. When scene.add() is called, what is added to the scene graph will be added to coordinates (0, 0, 0) unless specified otherwise.

Camera

The camera used, is a perspective camera built into three.js. The camera projection is designed to mimic how a human eye sees when rendering the 3D scene. When constructing the viewing frustum, the camera takes the FOV[16], aspect, near and far as parameters. What worked best for

[15] X, Y and Z coordinates
[16] Field of View

this solution was a FOV of 45, an aspect ratio of window width and 60% of widow height. Near and far camera frustum plane is set to 1 and 1000.

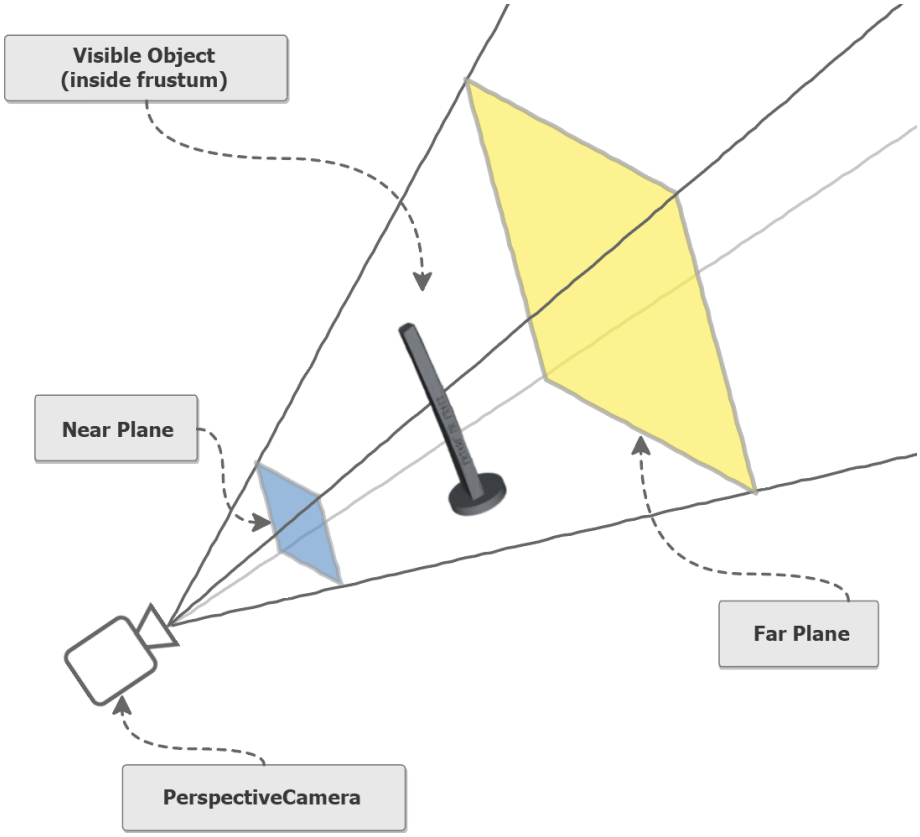


Figure 11, Visualization of Three.js Perspective Camera and Frustum, Created in Illustrator

Lights

Light is added to both the scene and the camera. The light appended to the camera shines a light on the model from the perspective of the camera, meaning from the user's point of view. This means that the models will have a light on them at all times. The light added directly to the scene is to light up the scene itself with an ambient, stationary light.

Group

Three.js Group is designed for containing groups of meshes, so it becomes clearer when objects are grouped together. This solution only implements groups on the main page, where the text is added to the model and not engraved in the model. This lets us update just the text in the scene with each keypress from the user, without modifying the model itself.

Geometry

A geometry is a representation of a mesh, line, or point geometry and defines the shape of a mesh. This includes vertices, faces, normals, colors, UV's, and can take custom attributes, reducing the cost of passing the data to the GPU[17]. In our final solution we have two types of geometries. The models which are loaded using a STLLoader, and text geometry created with three.js (Three.js, n.d, *D*).

Mesh

Three.js mesh class represents triangular polygon mesh-based objects. A mesh describes the shape of a 3D object using a collection of vertices, edges and faces. The constructor for a three.js mesh takes a geometry and material as parameters.

Material

Materials define the appearance of objects. This solution incorporates two types for three.js materials: MeshBasicMaterial and MeshLambertMaterial.

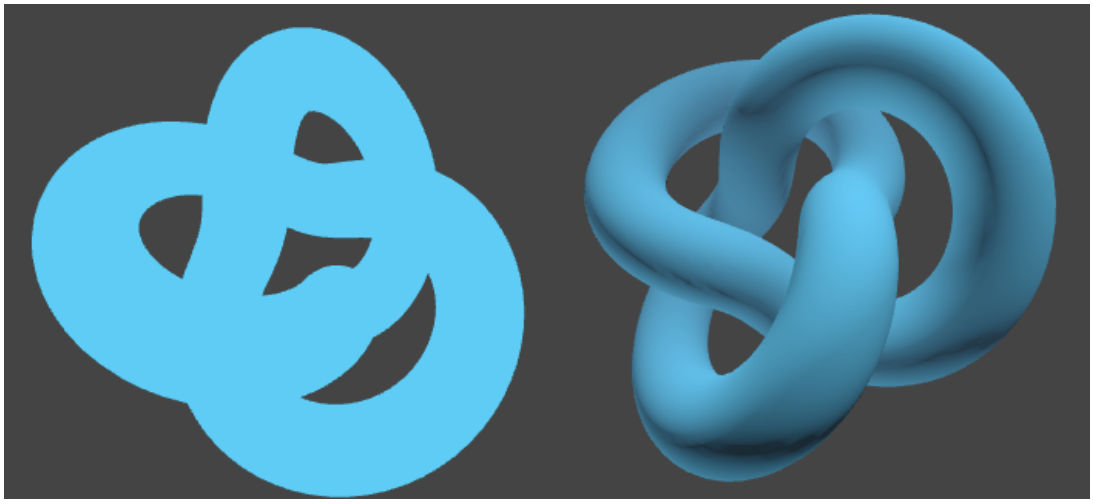


Figure 12, Visualization of MeshBasic and MeshLamberMaterial, Edited in Photoshop (Three.js, n.d, B, C)

MeshBasicMaterial is used for drawing geometries in a simple, shaded, flat way and is not affected by lights. MeshBasicMaterial is used for the appearance of the text on the main page

[17] Graphic Processing Unit

and has a bright white color for good visibility against the grey color of the model (Three.js, n.d, *B*).

MeshLambertMaterial is a material for non-shiny surfaces. It uses a Lambertian[18] model for calculating reflectance and is a good material for simulating surfaces such as wood and stone. The MeshLambertMaterial is used on our monumental stone models, along with a grey color for simulating an actual stone. Due to the simplicity of the illumination and reflectance of the material the performance will be better than other three.js material at the cost of some graphical accuracy (Three.js, n.d, *C*).

Positioning

The runic text on the models is created using a class of three.js for creating text geometries. It is constructed by providing a text string, parameters for text size and a font loaded with another three.js class. Both models have a x, y and z coordinates of (0, 0, 0) and the text geometries are positioned according to the model and if the text is going to be on the front or side.

```
if(model === 1 && pos === 1) {
    console.log('mod1pos1');
    textMesh.position.set(4, -25, 2.5);
    textMesh.rotation.z = Math.PI / 2;
    const resMesh = subtract(mesh, textMesh, material);
    scene.add(resMesh);
}
```

Figure 13, Codesnippet from Source Code

The code snippet above, figure 13, depicts the process of positioning the text to the front (position 1) of model 1 (Kuli-stone). The text is positioned and rotated 90˚. Figure 14 illustrates how the text geometry is positioned in the model geometry before the CSG subtraction operation for engraving is executed.

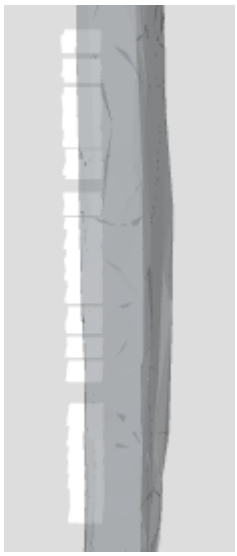


Figure 14, Model with 50% Opacity from Final Solution

[18] Lambertian reflectance is and matte or diffusely reflecting surface

4.1.8 Design

4.1.8.1 Colors

The colors for the solution were provided by the product owners at Smøla municipality. After being presented with the colors we chose a color scheme based on some of the suggestions. The canvas for displaying the model has a light grey background to display the model easily and ensure that the user understands that the model is possible to move around. The background of the second section is the primary color. This is #2D5670 which is a dark blue color. #EB9E43 is the contrast color used on buttons that are selected. This is a bright matt orange that compliments the blue nicely. In addition to this the buttons that are not selected have a very light-yellow color #FAF2D3 to create a smoother contrast than a normal white. This is the secondary color.

Figure 15, Color Palette

The color scheme on the admin page was created using the same colors provided by Smøla Municipality. Since the admin page is functionality based, the colors are simple. The blue color, #2D5670, is used on the header, while white and black is used on the rest of the page. The buttons used to change the location of the orders, or delete them have colors corresponding with their function, red for delete and green to send it to the next section.

4.1.8.2 Fonts

The font used on the page was provided by Smøla municipality. The font used throughout the page is an Adobe font called FF Dax Pro. Since this font is extremely thin, bold is used on most of the important locations on the client side to comply with WCAG 2.0 standards on color contrast. On the admin page the bold version of the font is used where appropriate.

The font used on the models as well as the preview section is a font from UiB called Gullhornet (Haugen, 2016) and was provided by NTNU AddLab and Smøla municipality. Some of the characters such as capital “M” and some numbers were not present in this font. Therefore, we had to manually adjust and create the characters in a font editor. More on this in section 5.1. The font used on the printId is Helvetica regular to be easily readable.

4.2 Backend

This solution uses a backend built with Node.js, Express and MongoDB with Mongoose. This solution has a single database with two collections, one for storing prints and one for storing admin users as documents in their respective collections.

From the start a database was needed to store text data for the solution. This database had to contain information about the print and visitor, as well as necessary data for admin users. The choice fell on MongoDB as it's an free, open-source, NoSQL database that stores data as JSON in documents, see section 4.2.1. Storing our data in JSON lets us have flexible and dynamic schemas and it’s human readable, making it easier to work with.

4.2.1 Table structure

MongoDB stores data in collections. Two collections are used in this solution. One for storing information about prints and user/visitor information. See figure 16. “_id” is an autogenerated id from MongoDB, while the “printId” is the id of the print itself. “Status” 1 through 3 is how far in the process a print is. It correlates to the different sections on the admin page. The “ok” stores a name, which is the name of the admin user that last updated the status of the print. All of the data stored in the print collection is available under the receipt page. “Tc” is a Boolean value which is true if the user/visitor has accepted the terms and conditions.


```
[
    {
        "_id": "607f03886a6961b241475d58",
        "createdAt": "2021-04-07T14:48:32.026+00:00",
        "printId": 123,
        "model": 1,
        "text": "Examle text",
        "pos": 1,
        "name": "Ola Normann",
        "email": "ola@normann.no",
        "Message": "Example Message",
        "tc": true,
        "status": 2,
        "ok": "admin"
    }
]
```


Figure 16, Print Collection Database Structure

The second collection used is the user collection. This collection only holds an auto-generated MongoDB id, a username and a hashed password.


```
[
    {
        "_id": "605b9aa4f05bb749a4884380",
        "username": "admin",
        "password": "$2b$10$rtyDZ/WKe7wxlLthqd0MxuFed2.J343lWqwwBT2L.gyNBoPk9bED6"
    }
]
```


Figure 17, User Collection Database Structure

Figure 18 is a visualization of access to the database and who can do what. A user/visitor can only submit prints to the print collection. The admin has the ability to update the status and delete prints in the print collection. The user collection is only available to the admin user. Here the admin can create and delete users.

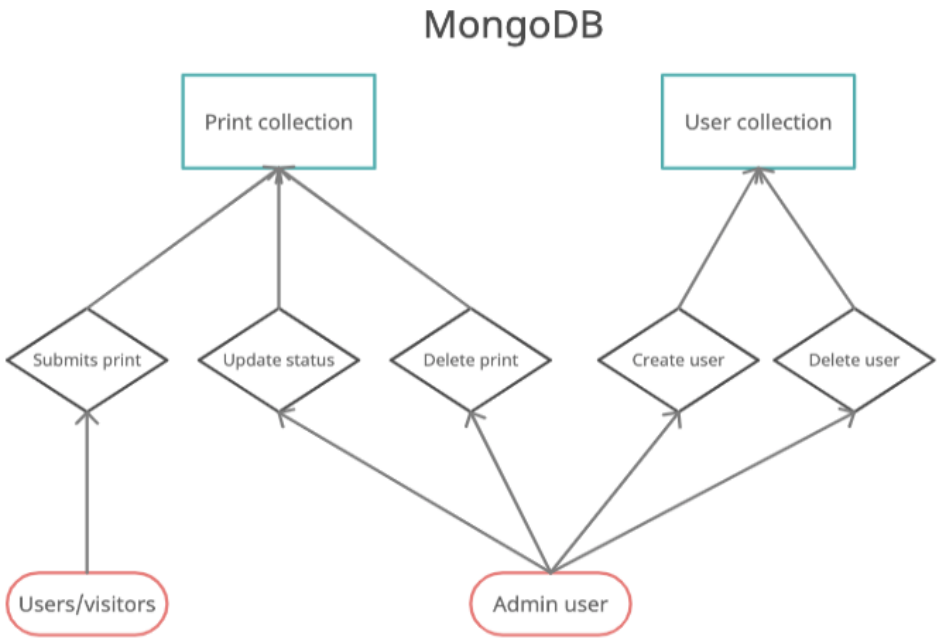


Figure 18, Database Access Visualization

4.3 WCAG 2.0

Since the web solution will be used by a large diversity of people, it is important that the solution satisfies the web content accessibility guidelines (WCAG). Universal design is about designing for all people no matter their abilities or disabilities. The main goal of universal design is to include everyone, and this also applies to digital products and platforms. Therefore,

the WCAG 2.0 standard was a heavy focus from the start when designing the user interface. According to Norwegian Law following the WCAG 2.0 standards are required and therefore important to comply with when designing and building a website. This is to ensure that they are within the standards of Norwegian law as well as being operable no matter the users' disabilities.

WCAG 2.0 consists primarily of four main principles. The perceivable, operable, understandable and robustness. For the two first principles there are in total 8 guidelines. The third principle has three guidelines, and the last principle has one guideline. In total there are 12 guidelines, and they all contain testable success criteria (uutilsynet, n.d *A*). To pass the WCAG 2.0 standard by Norwegian law the website has to comply with at least 35 of the 61 success criteria. In this report, the most critical ones related to the website will be discussed.

Perceivable

The perceivable criteria include color, contrast and alternatives for text. To ensure that these criteria were met, there was a heavy focus on colors during the design phase. A color palette was delivered by Smøla municipality for our use, and these colors were combined in the final design to accommodate for the correct contrast ratios. During development it was important to make sure images were perceivable for users with reduced vision and alt-tags were included for text-to-speech engines. Normal text in paragraphs was made sure to always have a contrast ratio of at least 4.5:1 (uutilsynet, n.d *B*).

Operable

The operable criteria include keyboard accessible, enough time, seizures and navigability. During development there was a heavy focus on ensuring these criteria were met. Keyboard navigation was created using a natural hierarchy in the HTML, since the front page is developed by using "fake" pages, there was added an <a> tag to the next and previous buttons to make sure that these were accessible no matter what step you are on in the process. There is a timeout function on our page that starts a screensaver, however this is set to three minutes without any mouse movement or input to make sure the user has enough time to complete the steps (uutilsynet, n.d *C*).

Understandable

The understandable criteria include that the solution is readable, predictable, and helps users avoid error. To make sure the solution complies with these criteria the site language is corresponding to the language code in the html so that the text is being interpreted correctly by a screen reader. The navigation is consistent throughout the entire solution to make sure the user has a predictable layout. In addition to this, all forms have correct and understandable labels as well as descriptive placeholder text (uutilsynet, n.d *D*).

Robust

The robust criteria are mostly about coding and that the code is validated and correct. It was made sure that the criteria were met by using lighthouse testing in the chrome browser, see section 4.6, and to make sure standard HTML elements were used where it was appropriate and available.

4.5 Security

4.5.1 Authentication, authorization and data handling

A user's role dictates their ability to access and write particular instances of data. This solution only implements one set of user roles, which is an admin role. Security was never the main focus of the solution but was implemented as a restriction to keep visitors/users from accessing the admin page with or without intention. There is no signup page, existing admin users can create and delete other admin users. If there are no existing admin users, an admin user will be created and inserted into the database automatically. An admin user consists of only username and password. The password is hashed using bcrypt, see section 3.3.2.1, algorithm before it is stored in the database.

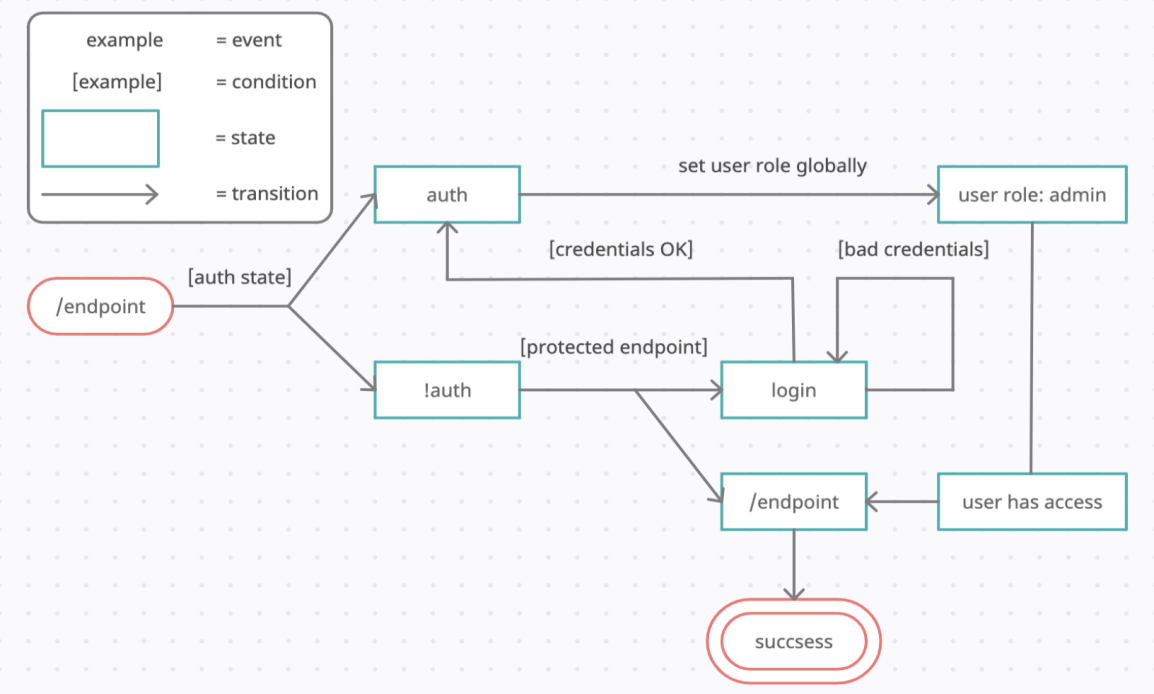


Figure 19, Sate Chart Explaining the Authentication and Authorization Process

MongoDB has network encryption. TLS and SSL are both standard technologies used by MongoDB for encrypting network traffic.

When an admin user signs in, an isLoggedIn cookie is created. When an admin user signs out, or after 30 minutes the cookie will expire and restrict access to the admin pages and functionalities until an admin sign in again.

4.5.2 STLExporter

There are two high severity vulnerabilities in this solution. One of the vulnerabilities is connected to the STLExporter package. This is dependent on depreciated code from three.js 0.125.0 or earlier. This code is vulnerable to ReDoS[19], when handling

```
three  <0.125.0
Severity: high
Regular Expression Denial of Service - https://npmjs.com/advisories/1639
No fix available
node_modules/three-stlexporter/node_modules/three
  three-stlexporter  *
  Depends on vulnerable versions of three
  node_modules/three-stlexporter

2 high severity vulnerabilities

Some issues need review, and may require choosing
a different dependency.
```

Figure 20, Npm Security Vulnerabilities

[19] Regular expression denial of service

RGB[20] or HSL[21] colors. However, the STLExporter is a crucial part of this solution. Since there are currently no updates to the STLExporter it has to be included as it is to keep important functionality of the website (npm, 2021a).

4.6 Performance

The figure 21 is a screenshot of the lighthouse results in the main page. The performance score indicates the site's overall performance when loading, see below. Accessibility indicates the site's overall compliance with accessibility guidelines. Best practices indicate the coding practices and checks the overall code health. SEO[22] indicates how well search engine optimization has been incorporated into the webpage.

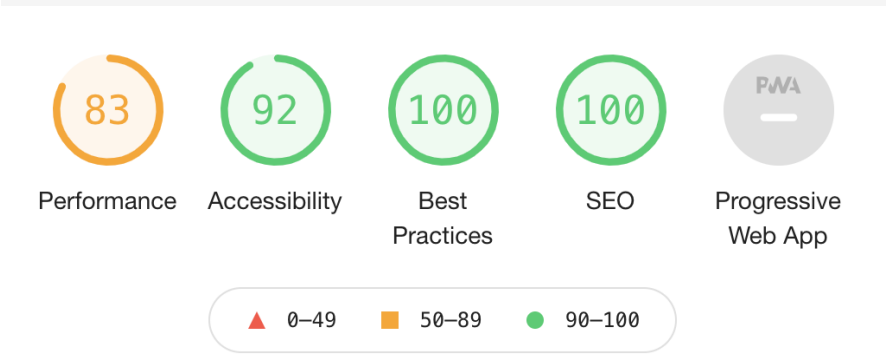


Figure 21, Screenshot of Lighthouse Results

This solution spends a large amount of time on executing code, therefore it is likely that the solution is CPU-bound, opposed to I/O-bound. This script executed is mostly done by the CPU[23]. These processes tend to run until they are preempted by the process scheduler[24] because they do not block I/O processes. An end-to-end test has been conducted in an attempt to back this up.

> An I/O bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations (Silberschatz, et al (p. 113, 2013).

[20] Red Green Blue color model
[21] Hue Saturation Lightness color model
[22] Search Engine Optimization
[23] Central Processing Unit
[24] Method to handle the removal of a running process from the CPU and select the next task

An I/O bound process spends most of the time submitting I/O requests and waiting for the response. Most GUI[25] applications are I/O bound, as they spend most of their time waiting for user input.

It's hard to measure the exact CPU usage of any given web application due to the several layers of abstraction involved in modern browsers. Additionally, the development team has little power over various end-user factors, such as the number of tabs open or plugins a user may have running.

The end-to-end test was run on a MacBook pro 2020 with an Intel Core i5 CPU with a clock speed of 2.0 GHz. Figure 22 one simulates a user browsing the main page and figure 23 simulates an admin user loading the print page. Both tests are an indication of runtime performance on slower devices, tested by throttling the CPU by a factor of 4x and the network speed is throttled to "fast 3G".

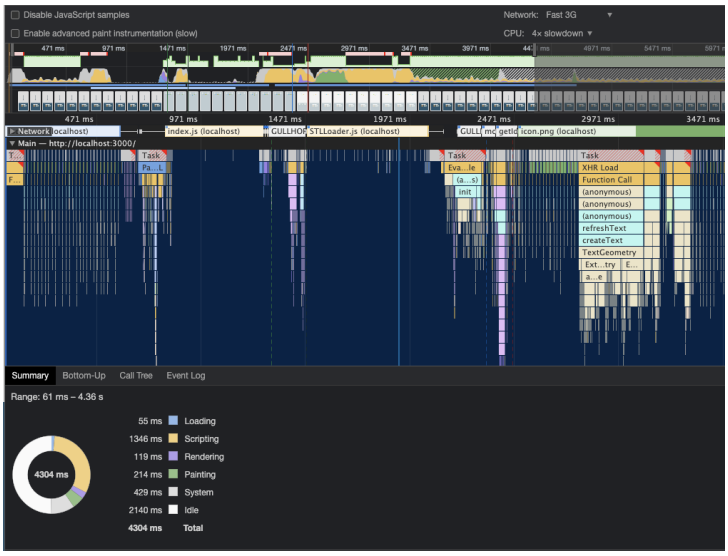


Figure 22, Screenshot of Performance Test, Main Page

As the solution is using most of its time on scripting, it's fair to say that the solution is CPU-bound. The admin/print page spent a lot of time executing JavaScript. This was expected due to the CSG process which is quite CPU intensive. In future development this could be improved by minifying JavaScript.

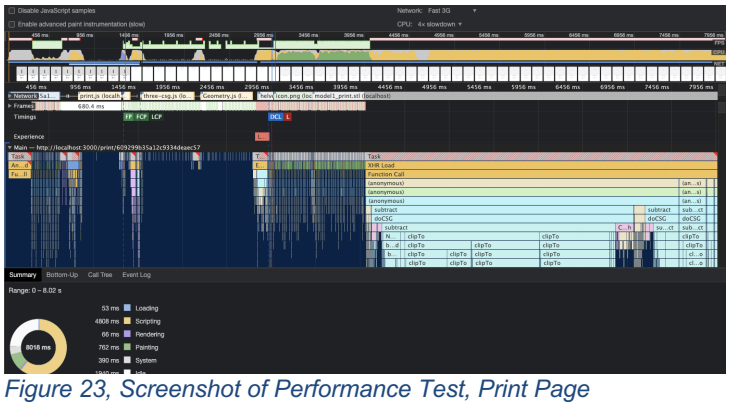


Figure 23, Screenshot of Performance Test, Print Page

[25] Graphical User Interface

Minifying the JavaScript refers to removing unused code, shorter names for variables and functions and removing redundant data.

5. DISCUSSION

This chapter will discuss the implementation and solution, what problems occurred during the process that had consequences for the solution and discuss the implications coronavirus had for this project.

5.1 Front-end framework

Early in the process we contemplated using a front-end framework. Vue.js was considered as the group had previous experience using this framework. A decision was made not to implement a front-end framework as we could not find a well working solution for implementing three.js and Vue.js in one solution. As three.js was essential for us in order to achieve our goal for the final solution, the group decided not to implement a framework and instead use JS/HTML/CSS to build out our front-end.

5.2 Issues along the way

During the development of this project, we encountered several issues, some more significant than others, that in turn had consequences for how the final solution turned out. This section will address the problems and challenges faced, how they were solved and how they affected the final solution.

5.2.1 Rune Font

The font for the rune text was provided to us by the product owners. The issue with the font is that three.js was unable to create geometry from a .ttf file and display it on the model. By using an online converter (Gero3, n.d), the text was converted from a .ttf file to a json object which three.js was able to load and display on the website. Some of the characters did not exist in this font and were changed by manipulating the json file. This file was manipulated by adding the properties of for example lowercase “m” to the uppercase “M”. In addition to this some characters created issues when they were sent to the database. For example, node.js interpreted “<” as a piece of code and removed the whole string when it was sent to the database. Therefore, this was removed and not allowed to be typed into the input field, to ensure that this

error did not occur. When the json file that displayed the letters on the monumental stone were changed, it no longer matched the preview of the text. Therefore, we had to manually edit the .ttf file in FontForge (FontForge, n.d) to make sure the text in the preview section matched the text that was displayed on the monumental stone model.

5.2.2 Stl downloader

The .stl downloader which is used to download the models on the print page of the admin site had restrictions to how large the downloaded file size could be. Since the models received from the product owners were extremely detailed, this made the file size very large. Therefore, the size of the models had to be reduced which will be discussed more further down under CSG Operations, see section 5.1.5. When the models were reduced it allowed for the stl downloader to work. The consequence of this however was that the models were a lot less detailed, however after printing them this difference was barely noticeable, see figure 26.

5.2.3 Matching print to customer

After user tests with interaction designers, it became clear that the way for admins to match the print to the customer was difficult. The first solution was the receipt page where the admin could compare the rune text on the model to the rune text on the receipt page. This was however not the optimal solution as people in general are not familiar with rune text. In addition to this, the possibility of someone writing the same text on two different prints was present. Therefore, a base was added at the bottom of the model that has two functionalities. Firstly, it made it possible for the model to stand upright and secondly a printId was added at the bottom of the circle. This id is also on the receipt so the admin can easily compare the id’s and give the right model to the right visitor. The consequence of this is the added circle on the bottom that is not true to the real-world monumental stones. However, the stone was unable to stand on its own before the base was added and therefore this was a better solution for the customers and the product owners.

5.2.4 PDF Generator

The receipt page was originally intended to be a site where a PDF document containing the order information was generated. This was going to be executed using a library called jsPDF. However, this caused some issues with Node.js since it would not import the library to the project with the way the current solution was built. Therefore, the receipt page was created as a normal web page containing the order information, which could be printed out if deemed necessary.

5.2.5 Engraving / CSG operation

The CSG operation of making the text engraved in the monumental stone model caused us some problems along the development process. As mentioned in 4.1.4 we are using a CSG library for three.js to achieve the engraving.

In the first version of engraving, the CSG library used was several years old and had not been updated for a long time. Because of this we had to downgrade the version of three.js used from v.0.126.1 to v.0.115.0, since three.js had a breaking update, v. 0.125.0, that depreciated some of three.js' older functionality. In March 2021 the git repository for the CSG library was updated to work with three.js > v.0.125.0 and we were able to use the most recent version of three.js, v.0.126.1, as of writing this.

The way the CSG operation works is by going through all of the model's vertices and faces and combining the text geometry and model geometry into one. The fever vertices and faces a model has, the faster the CSG operation can be completed. The models we received from the product owner were large, both in file size and details, especially model 1 "Kuli-stone".

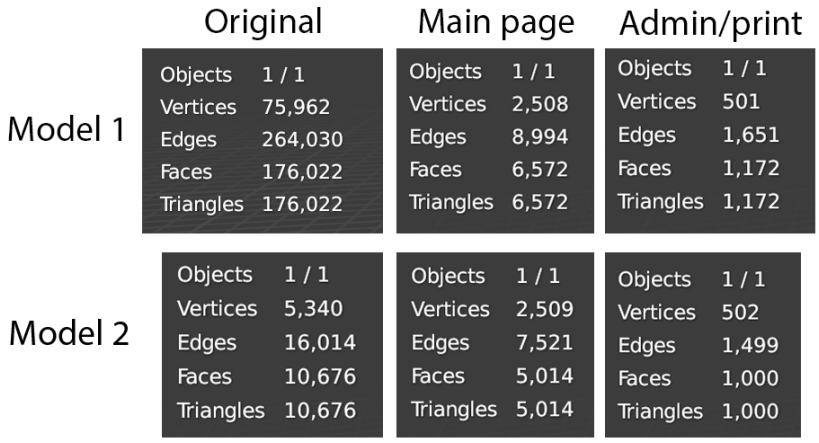


Figure 24, Model Size and Details, Screenshot from Blender

The models we received were too large to be usable in our intended solution. Vertices in this context is a point where two or more curves, lines, or edges meet, a face is a flat or curved surface on a 3D shape. A cube, as an example, has 8 vertices and 6 faces. A model with more than 10.000 vertices will be slow and a model with over 100.000 would make our solution close to unusable (3dless, n.d).

In our first version with functioning engraving with the original model 1, page loading time was ~8 minutes, which was unacceptable for what we wanted the solution to be. Using an online tool for simplifying STL models called 3DLess (3dless, n.d), we were able to reduce the number of vertices. After testing different amounts of vertices, 2500 vertices for the main page and 500 vertices for the admin/print page were optimal.

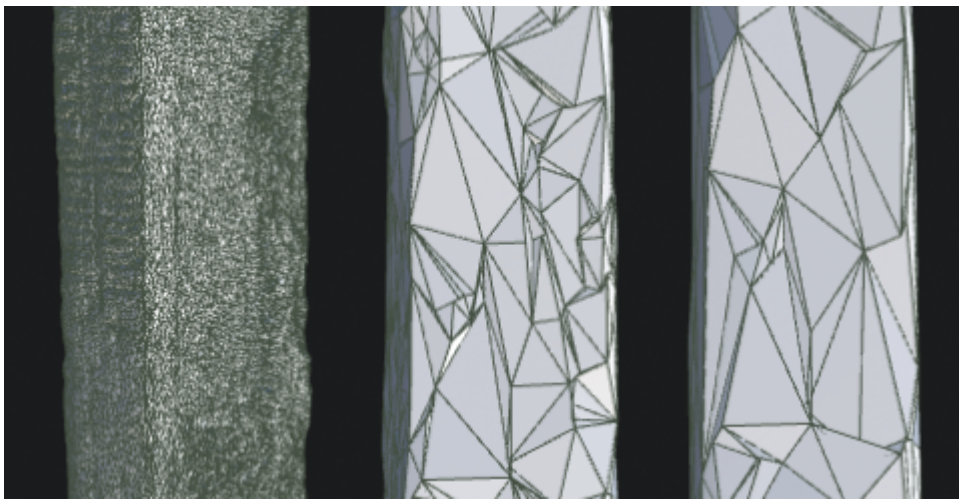


Figure 25, Visualization of Figure 24 (3dless, n.d)

With models reduced to 500 vertices, the page loading time for the admin/print page was reduced to just a couple of seconds depending on the length and number of characters in the user's text. A loading animation was added as feedback to users that the page is working and loading.

In the main page we wanted the text on the model to be updated as the user types in the input field. When the engraving operation runs as mentioned above, it creates a new geometry of the model and text geometry and renders it to the three.js scene. With engraving on the main page, the CSG operation would have to run at each keystroke, which would be very process intensive

and slow. We opted for keeping model and text geometries as two separate geometries, which means we don't have to do anything with the model geometry and only update the text geometry as the user types. This makes the change in input on the model instant, and the text is more visible for the user as the text is extruding from the model with a white color contrasting the grey color of the model.

Because of the reasons above for not applying engraving on the main page, we could have a bigger file-size model with more vertices and faces and more details without affecting performance. Keeping the models more detailed at 2500 vertices also makes it look better for the users while creating their print. On the admin/print page reducing the model to 500 vertices, the finished 3D printed model still had enough details to look good and keep its recognizable features.

Figure 26, Example of Two Prints During Development

5.2.6 Placing, rotating and tilting objects and text in 3D space

Positioning the text geometry to the model geometry led to some issues during development. Model 1 caused no issues as its surfaces; both the front and side are relatively straight and flat. Model 2 however has rougher, more uneven surfaces and starts thicker at the bottom and gets increasingly narrower to the top. On the front the text geometry is tilted slightly backwards so the text geometry is evenly engraved in the model. Position 2 on the side of model 2 however

had some issues as the rotations of the object were increasingly complex, after already rotating the text on the X and Z axis to move it to the side and in an upright position. Adding more rotation on the Y axis does not tilt the text towards the model, but it is rotated around itself, as we add rotation to the already existing rotation on the Y axis. Trying to solve this turned out to be too difficult to solve in the time we had left, with the complexity of positioning and rotations in 3D space.

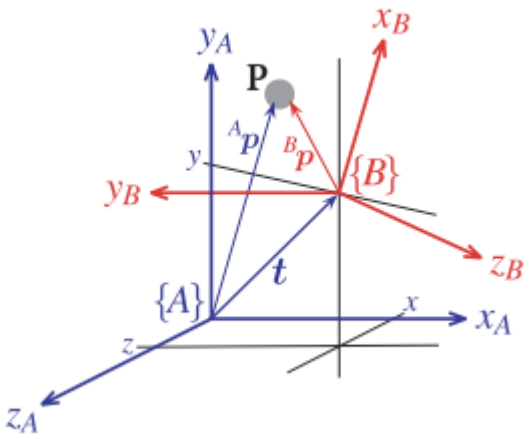


Figure 27, Image from (Corke, 2017, p. 32)

Figure 27 is an example of the different axis and rotations involved when trying to position and rotate an object {B} around another object {A}.

This is beyond the scope of our project and would be too time consuming to learn in time for the deadline. Therefore, the solution became to tilt the model itself into the text instead of tilting the text into the model.

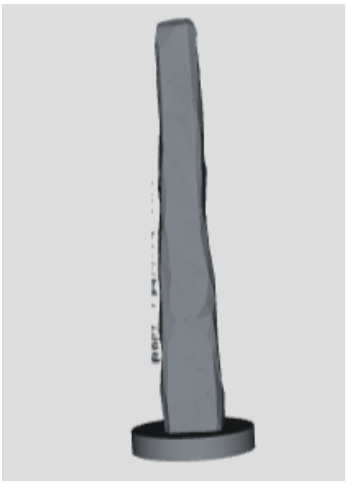


Figure 28, Screenshot of Tilted Model 2

Figure 28 shows model 2 with text position 2 as it is displayed for an admin on the admin/print page. The model being tilted has no effect on the final 3D printed model as it is being placed upright in the slicer before printing. Another side effect of model 2 not having as smooth and flat surfaces as model 1 is that we can see some material fragments extruding from the model in figure 28. This, like the tilt, does not affect the finished product when 3D printed.

Another problem we had with model 2 was that sometimes depending on the text length and position, the CSG operation would create empty layers through the model. This did not affect printing in other ways than it was not pretty to look at. See figure 26. If you look at model 2 on the left, you can see what looks like a horizontal line though the model ca. ⅓ from the top. This is the result of printing a model with an empty layer. As a solution to this problem, we switched position 2 from one side of the model to the other, as the model's surface was smoother and flatter on this side. This mostly fixed the problem.

5.3 Mobile Version

A mobile version was considered for this solution. However, this was deemed unnecessary since this solution is only going to be used on one computer and not on any mobile devices. However, there was created a prototype in Figma for a potential mobile version that could be implemented in the future, see attachment 6.

5.4 Deployment

The deployment of this solution will happen at the end of May. Here the group will travel to Smøla and Gurisentret to deploy the solution. The plan is to deploy the solution through Heroku. Heroku is a cloud platform as a service, which offers services such as hosting, monitoring, building and scaling applications for businesses. Here we will only use the hosting service to have the website up and running live for the product owner. The site could also run locally, since it is intended to be used on a single computer in the beginning. Unless the product owner does not want it deployed elsewhere, this is the plan for the deployment at the end of May.

In the beginning the product owners wanted the solution to be located at Squarespace since the rest of their portfolio is there. However, after some research this was deemed difficult as three.js was not supported on Squarespace and Node.js was difficult to implement as the backend of the solution.

5.5 Coronavirus

One of the major restrictions to this bachelor project was the ongoing pandemic and the covid-19 virus. This coronavirus made restrictions to user testing and the ability to meet in person. Fortunately, this project was simpler to adjust to these conditions. Most of the meetings were conducted digitally both with the product owners, and group meetings. The project was also in need of a 3D printer for prototyping prints to ensure the quality and the visibility of the engraved letters. Since the 3D lab was shut down for a couple of weeks, we used that time to finetune the solution, as well as make several different tweaks to the settings on the engraving to make sure we could get the best text possible on the 3D printed monumental stone.

6. CONCLUSION

The goal of this project was to create an inspiring and creative solution for Gurisentret's visitors where they could write their own inscription on a 3D model of the monumental Kuli-stone. This was achieved through thorough gathering of information through qualitative interviews described in section 3.1 Research. From there the information gathered was used to create the solution described in section 4. Implementation/Solution. The solution is created to be as user friendly as possible and have the lowest threshold for usability. This was ensured through multiple user tests of several age groups. The goal for Smøla municipality and Gurisentret was to engage and inspire children to learn about history, culture and runes through modern technology. Since this solution is being deployed at the end of May this goal cannot be answered at this point. To give a definitive answer to this goal, it is necessary to receive feedback from end users over time. However, feedback from the user test suggests that we have been able to convey culture and history in an engaging way through modern technology. In addition to this the user tests also indicate that the solution is very user friendly, however this remains to be seen.

6.2 Further development

There are some functionalities that could be implemented in further development. First of all, the admin page could have fewer clicks when downloading the final model. Since the printers have the possibility to be connected to the internet, this could also be implemented in the future. This will make sure the user of the admin page does not have to export the .gcode from the slicer and then manually insert it into the printer to start the print.

Since this solution is supposed to be used by a variety of different people, a translation of the entire solution to English could be developed further on. This would ensure a broader audience for the solution as well as better compliance to all users that might not understand Norwegian.

The possibility to change password and have a forgot password button on the login page is also something that could be developed in the future. This is not implemented in this version since the login restriction on the admin page is created with the sole purpose of preventing users of the client side to access the admin section. In addition to this the whole security of the website could be improved to prevent malicious attacks and unwanted access.

The page containing the terms of service are not complete yet since Smøla municipality has not yet created this. Therefore, there is still Lorem Ipsum[26] text in the terms and conditions page. A complete log of all prints that have run through the system could also be implemented in the future if needed. This could be used for statistics, keeping track of what prints are the most popular and what phrases are used the most etc.

As mentioned in 5.2.6 the model geometry of model 2 with position 2 is titled to make the text engrave properly into the model. In further development this should be fixed, so that the text is tilted instead of the model. The receipt page when printed should also be formatted to better fit an A4 page, when printed.

[26] Placeholder text

7. REFERENCES

Corke, Peter (2017), *Robotics, Vision and Control*. 2nd edn. Springer International Publishing AG.

Cprime (n.d), *What is AGILE?*. Available at: https://www.cprime.com/resources/what-is-agile-what-is-scrum/#:~:text=Agile%20software%20development%20refers%20to%20a%20group%20of%20software%20development,%2Dorganizing%20cross%2Dfunctional%20teams (Accessed: 22.03.21)

EJS (n.d), *EJS -- Embedded JavaScript templating*. Available at: https://ejs.co/ (Accessed: 02.05.21)

Figma (n.d) *About Figma*. Available at: https://www.figma.com/about/ (Accessed: 10.05.21)

Foley, James D. (1996) *Computer Graphics: Principles and Graphics*. 2nd edn. Addison-Wesley Publishing Company, Inc.

FontForge (n.d) *FontForge Open Source Font Editor*. Available at: https://fontforge.org/en-US/ (Accessed 27.04.21)

Gero3 (n.d) *Facetype.js*. Available at: https://gero3.github.io/facetype.js/ (Accessed: 10.03.21)

Gurisentret (2019) *Se Kulisteinen og Edøy gamle kirke i 3D!*. Available at: https://www.gurisentret.no/se-kulisteinen-og-edoy-gamle-kirke-i-3d/ (Accessed: 07.04.21)

Guru99 (n.d), *What is MongoDB? Introduction, Architecture, Features & Example*. Available at: https://www.guru99.com/what-is-mongodb.html (Accessed: 19.04.21)

Haugen, Odd (2016) *Gullhornet rune font*. Available at: https://folk.uib.no/hnooh/runefont/Gullhornet-e.html (Accessed: 02.03.21)

Heroku (2021) *Deploying with Git*. Available at: https://devcenter.heroku.com/articles/git#ssh-git-transport (Accessed: 10.05.21)

IDEO (n.d, *A*), *Brainstorming*. Available at: https://www.ideou.com/pages/brainstorming#why-brainstorm (Accessed: 15.01.21)

IDEO (n.d, *B*), *Rules of Brainstorming*. Available at: https://cdn.shopify.com/s/files/1/0259/7876/5396/files/IDEO_Rules_Of_Brainstorming.pdf?v=1596746304 (Accessed: 15.01.21)

Interaction Design Foundation (2021), *Design iteration brings powerful results. So, do it again designer!*. Available at: https://www.interaction-design.org/literature/article/design-iteration-brings-powerful-results-so-do-it-again-designer (Accessed: 22.03.21)

Karnik, Nick (n.d) *Introduction to Mongoose for MongoDB*. Available at: https://www.mongodb.com/why-use-mongodb (Accessed: 19.04.21)

Liv Oddrun Voll (n.d), *Om skaperskolen*. Available at: https://skaperskolen.no/om-skaperskolen/ (Accessed: 03.02.21)

Manthrax (n.d) *THREE-CSGMesh*. Available at: https://github.com/manthrax/THREE-CSGMesh (Accessed: 02.02.21)

MongoDB (n.d), *Why Use MongoDB & When to Use MongoDB*. Available at: https://www.mongodb.com/why-use-mongodb (Accessed: 19.04.21)

Mrdoob (n.d), *STLLoader.js*. Available at: https://github.com/mrdoob/three.js/blob/master/examples/js/loaders/STLLoader.js (Accessed: 16.04.21)

Nielsen, Jakob (1993), *Iterative Design of User Interface*. Available at: https://www.nngroup.com/articles/iterative-design/ (Accessed: 22.03.21)

Nielsen, Jakob (2000), *Why You Only Need to Test with 5 Users*. Available at: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (Accessed: 23.04.21)

Nodejs (n.d), *The Node.js path module*. Available at: https://nodejs.dev/learn/the-nodejs-path-module (Accessed: 16.04.21)

Norton (2019), *What are computer cookies?*. Available at: https://us.norton.com/internetsecurity-privacy-what-are-cookies.html (Accessed: 11.05.21)

npm (2021a), *Regular Expression Denial of Service*. Available at: https://www.npmjs.com/advisories/1639 (Accessed: 08.05.21)

npm (2021b), *dotenv*. Available at: https://www.npmjs.com/package/dotenv (Accessed: 10.05)

NTNU (n.d), *Kulisteinen - Norges dåpsattest.* Available at: https://www.ntnu.no/museum/kulisteinen (Accessed: 03.02.21)

Opensource.com (n.d) *What is open source?*. Available at: https://opensource.com/resources/what-open-source (Accessed: 09.05.21)

Open Source Initiative (n.d) *The MIT License*. Available at: https://opensource.org/licenses/MIT (Accessed: 02.02.21)

Ouelette, Aleksander (2021), *What is Bootstrap? A Beginner's Guide*. Available at: https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/ (Accessed: 15.04.21)

Remy (n.d), *nodemon*. Available at: https://www.npmjs.com/package/nodemon (Accessed: 16.04.21)

RIP Tutorial (n.d), *three.js Orbit Controls*. Available at: https://riptutorial.com/three-js/example/26556/orbit-controls (Accessed: 15.04.21)

Selzer, Michelle (2020), *Salt and Hash Passwords with bcrypt*. Available at: https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt (Accessed: 16.04.21)

Three.js (n.d, *A*), *OrbitControls*. Available at: https://threejs.org/docs/#examples/en/controls/OrbitControls (Accessed: 15.04.21)

Three.js (n.d*, B*), *MeshBasicMaterials*. Available at: https://threejs.org/docs/#api/en/materials/MeshBasicMaterial (Accessed: 09.05.21)

Three.js (n.d, *C*), *MeshLambertMaterial*. Available at: https://threejs.org/docs/#api/en/materials/MeshLambertMaterial (Accessed: 09.05.21)

Three.js (n.d, *D*), *BufferGeometry*. Available at: https://threejs.org/docs/?q=geometry#api/en/core/BufferGeometry (Accessed: 09.05.21)

Uutilsynet (n.d *A*) *WCAG 2.0 standarden*. Available at: https://www.uutilsynet.no/wcag-standarden/wcag-20-standarden/86 (Accessed: 15.04.21)

Uutilsynet (n.d *B*) *1. Mulig å oppfatte*. Available at: https://www.uutilsynet.no/wcag-standarden/1-mulig-oppfatte/714 (Accessed: 15.04.21)

Uutilsynet (n.d *C*) *2. Mulig å betjene*. Available at: https://www.uutilsynet.no/wcag-standarden/2-mulig-betjene/716 (Accessed: 16.04.21)

Uutilsynet (n.d *D*) *3.2 Forutsigbar*. Available at: https://www.uutilsynet.no/wcag-standarden/32-forutsigbar/726 (Accessed: 16.04.21)

webglfundamentals.org (n.d), *WebGL Fundamentals*. Available at: https://webglfundamentals.org/ (Accessed: 03.05.21)

Wikipedia (n.d) *Constructive solid geometry*. Available at: https://en.wikipedia.org/wiki/Constructive_solid_geometry (Accessed: 30.04.21)

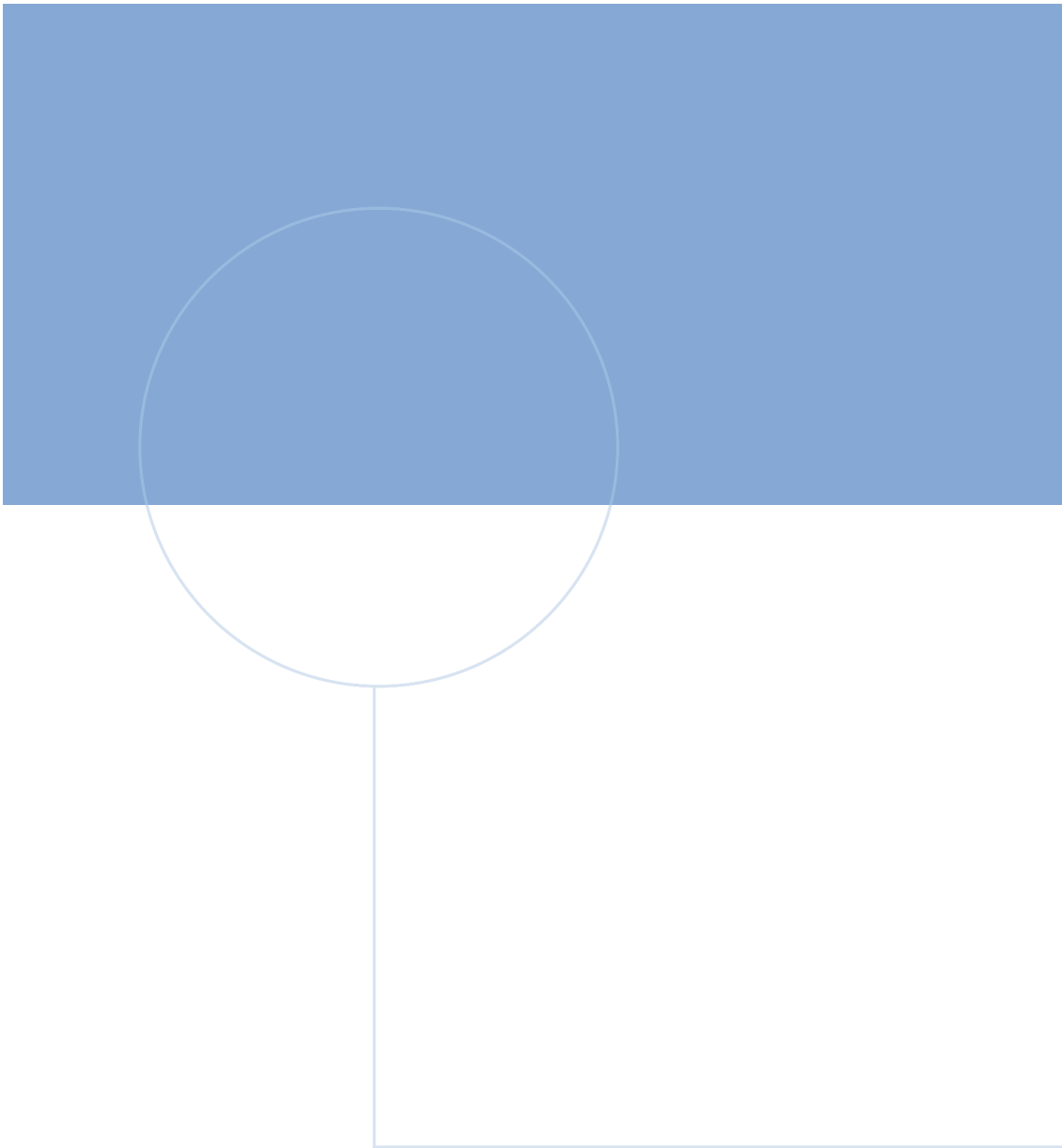
Østbye, H, *et al* (2017) *Metodebok for mediefag*. 4th edn. Bergen: Vigmostad & Bjørke AS.

3dless (n.d) *3DLess.com | Reduce STL File vertices*. Available at: https://3dless.com/ (Accessed: 11.03.21)

Silberschatz, Abraham., Galvin, Peter., Gagne, Greg (2013) *Operating System Concepts* 9th edn. John Wiley & Sons,Inc. Available at: http://www.cs.nthu.edu.tw/~ychung/slides/CSC3150/Abraham-Silberschatz-Operating-System-Concepts---9th2012.12.pdf (Accessed: 11.05.21)

8. List of Attachments

Attachment 1 – Source Code
Attachment 2 – NTNU Project Agreement
Attachment 3 – Figures
Attachment 4 – Links
Attachment 5 – Runic Alphabet .gcode
Attachment 6 – Prototypes and Sketches
Attachment 7 – Brukerveiledning Gurisentret 3D Print



NTNU

Kunnskap for en bedre verden