Ira Julia Rainto

Media file compression application

CASE: Semio

Thesis
Autumn 2018
SeAMK School of Technology
Information Technology

SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree Programme: Information Technology

Specialisation: Programming

Author: Ira Julia Rainto

Title of thesis: Media file compression application: CASE: Semio

Supervisor: Markku Lahti

Year: 2018 Number of pages: 51

The subject of the Bachelor’s thesis was to create a simple and easy-to-use media compression program for an advertising agency. The application was developed from the point of view of speeding up the content creation of websites. The idea is that an employee can change the file conversion that he most often needs with the developed application. The purpose of file transformations is to speed up the downloading of websites.

The developed application was aimed to allow to compress the size of image and video files. The application can use two different file compression methods in image files; lossless and lossy. In video files, it is possible to change the resolution and file format when compressing a file.

The image file formats available in this application are: JPEG, PNG, SVG, GIF and the available video file formats are: AVI, MPEG-4, Flash Video, Ogg, WebM, MOV, Windows Media Audio.

The program has been developed and tested on a separate server. Application is used in production and it is installed on the company Semio’s own server.

Keywords: ECMAScript, HTML5, CSS, Gulp, FFMPEG

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikka

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Ira Julia Rainto

Työn nimi: Medianpakkausohjelma: CASE: Semio

Ohjaaja: Markku Lahti

Vuosi: 2018 Sivumäärä: 51

Opinnäytetyön aiheena oli luoda yksinkertainen ja helppokäyttöinen median pakkausohjelma mainostoimiston käyttöön. Sovellustyökalu kehitettiin verkkosivustojen sisällöntuotannon nopeuttamisen näkökulmasta. Tavoitteena oli, että käyttäjä pystyy muuntamaan kehitetyllä sovelluksella kaikki yleisimmin tarvitsemansa tiedostomuodot ja nopeuttaa verkkosivustojen latautumista.

Opinnäytetyön tuloksena kehitetiinn sovellus, jonka avulla voidaan tiivistää sekä kuva- että videotiedostojen kokoa. Sovellus voi käyttää kuvatiedostoissa kahta erilaista tiedoston pakkaustapaa: häviötön ja häviöllinen. Videotiedostoissa voidaan tiedoston pakkaamisen yhteydessä muuttaa resoluutiota ja tiedostomuotoa.

Sovelluksessa käytettävissä olevat kuvatiedostomuodot ovat: JPEG, PNG, SVG ja GIF. Videotiedostomuodoista käytettävissä ovat: AVI, MPEG-4, Flash Video, Ogg , WebM, MOV ja Windows Media Audio.

Ohjelma kehitettiin ja testattiin erillisellä palvelimella. Tällä hetkellä sovellus on tuotantokäytössä, asennettuna yrityksen omalle palvelimelle.

Asiasanat: ECMAScript, HTML5, CSS, Gulp, FFMPEG

TABLE OF CONTENTS

Terms and Abbreviations

API	Application programming interface, an interface for different applications to exchange data with each other.
Cache	Stores recently used information so that it can be quickly accessed at a later time.
CLI	Command Line Interface is a tool that is used for interacting with software and operating systems by using commands.
Cross Browser	Application that works in different browsers, mobile and desktop. This means that sites show's up correctly in different browsers.
CSS	Cascading Style Sheet defines how the HTML structure is presented visually.
ECMAScript	Forms the basis of the JavaScript as a scripting language.
FFMPEG	FFMPEG is a cross-platform tool for converting, compressing and streaming video, audio and some image formats.
Front-End Framework	Pre-written CSS stylesheet and/or JavaScript library meant for saving time and for adding content to your web application.
Gulp	Gulp is an open source toolkit for JavaScript that makes certain kinds of tasks automatic.
HTML5	Latest version of HTML.
HTTPS	Hyper Text Transfer Protocol Secure is a combination of HTTP- and TLS/SSL-protocols.
I/O	Input/Output, is used to explain data transfer between an operation or device.
jQuery	Library that simplifies JavaScript programming.
Material Design	Is a guideline of visual language, developed by Google, which synthesizes good design and innovation.

Material Icons	Icons that are created by using material design guidelines.
Middleware	Software that works as a bridge between an operating system and an application.
Native App	Application that is built for a particular device and an operating system.
npm	Node package manager is a way to share or borrow packages.
PHP	Hypertext Preprocessor, general-purpose scripting language for web-development that can be implemented into HTML.
PPM	Context coding: PPM is considered to get the best compression ratios although it is slow.
Python	Object-oriented programming language similiar to Perl, Ruby, Schema, or Java.
Responsive	A concept of making same application to scale well visually. Independent from the device screen resolution or orientation.
REST	Representational State Transfer (REST) is an architecture model that is based on the HTTP protocol.
Progressive Web App	(PWAs) makes your application more native application like.
SSL	Secure Sockets Layer, is an encrypted link between a server and an interface.
Transfer-Encoding	Is a hop-by-hop header that works as a method to communicate between two nodes.

Figures

1 INTRODUCTION

Compression is one way to reduce the uploading time of websites. Image-, audio- and video -files have large file sizes and, therefore, longer uploading times. There are many ways to deploy compression on a website. File format compression is used for reducing wasted space in files. Lossless- and lossy compression are two compression algorithms used to achieve this.

The aim of this study is to create a media compression application for Semio Oy. The application is supposed to implement two different compression methods, lossless, lossy, and tools for media editing such as resolution change and changing format to another. The purpose of this application is to support the most used media formats and work with all modern browsers. The application is intended to facilitate the everyday life of employees and customers, through an effective media compression application.

The first chapter introduces background information and the aim of the study. The second chapter will go through the techniques used for making the application. The third chapter will explain how to achieve media compression through a web application. Then the rest of the thesis provides information about the application itself and its working methods and functionality.

2 COMPRESSION IN THE WEB

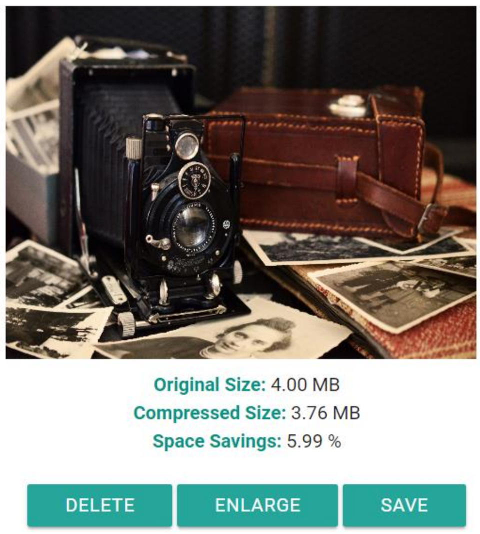
Compression has an important part in the web. It can increase the performance of the website and result to faster uploading times. In some case's file size reduction can have lower bandwidth capacity needs. Algorithms have improved over the years and they continue to improve still. These algorithms are supported in the clients and the servers and when implemented correctly the compression can work on its own. (MDN web docs 2017b.)

2.1 File format compression

Every data type has wasted space in its binary form. Media files are big and, therefore, it is important to optimize files before deploying them on the Web. There are two kinds of optimized compression algorithms for file formats. Those are lossless and lossy compression. (MDN web docs 2017b.)

Lossy compression is more efficient than lossless compression. Lossy compression may provide up to 90% decrease in size compared to the original file (MDN web docs 2017b). With the right compression ratio, there may be no quality loss detectable to a human eye and, therefore, it is advisable to use lossless compression to achieve the highest quality possible. If the web page is needed to be downloaded fast, the best method to use would be lossy compression. (Jessier 2017.)

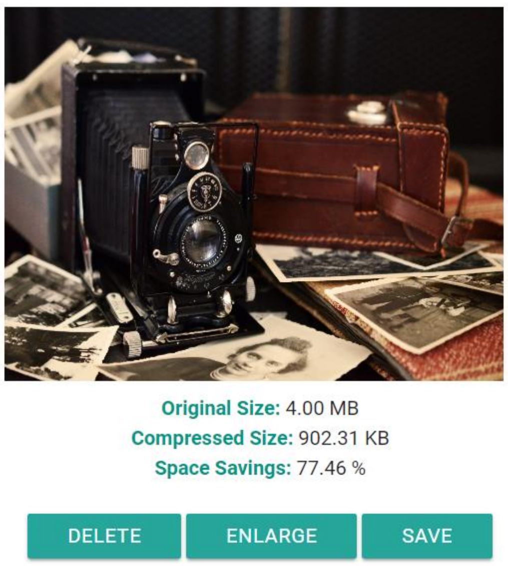
Lossless compression. This method is used in text compression and lossless format media files. Lossless compression provides ratios that are within 10% of the best algorithm PPM (Prediction by partial matching) but runs faster than PPM does. This type of compression reduces the size of the file but doesn't cause any quality loss. It rewrites the data in a more efficient way. In the figure1 it can be seen how small space savings are in lossless compression. In some cases JPEG and PNG remove unnecessary metadata and can thus cause slight quality loss. (Jessier 2017.)



Original Size: 4.00 MB
Compressed Size: 3.76 MB
Space Savings: 5.99 %

Figure 1. Lossless compression

Lossy compression. When using lossy compression method original data does get altered, meaning that it is not possible to reverse the file back to the original form. It still doesn't mean that the quality is lost in the process. In most cases human eye cannot detect the difference between the original and the output file. Video formats in the web and image formats such as JPEGs, are considered to be lossy formats. (Jessier 2017.)



Original Size: 4.00 MB
Compressed Size: 902.31 KB
Space Savings: 77.46 %

Figure 2. Lossy compression

JPEG algorithm may also have as high compression ratios as 90-95%, with little degradation. (Nelson & Gailly 1995.) Figure 2 shows how the space savings of a JPEG image are 77.46% in lossy compression compared to the original one.

2.2 End-to-end compression

Data savings in website speeds are achieved by using end-to-end compression, as shown in the figure 3. Server compresses the body of a message and it will stay untouched until it reaches the client. (MDN web docs 2017b.)

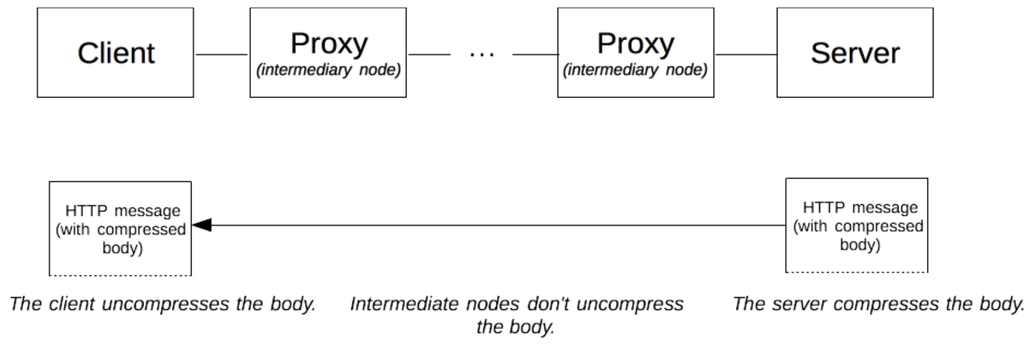

Client
Proxy
(intermediary node)
...
Proxy
(intermediary node)
Server
HTTP message (with compressed body)
HTTP message (with compressed body)
The client uncompresses the body.
Intermediate nodes don't uncompress the body.
The server compresses the body.

Figure 3. End-to-end compression (MDN web docs [2018b].)

The most common way to use this compressing method is gzip, since it can achieve compression ratios of 70-90% in text-based content but it is not efficient to use it with media files (Grigorik 2018). All modern browsers support gzip and suggest using it for all HTTP requests. Using gzip can reduce the size of the transferred response by 90%. This will lead to a reduced download time and data usage for the client. (Google Developers 2018.)

2.3 Hop-by-hop compression

Figure 4 shows how hop-by-hop compression doesn’t perform the compression in the server but in between any two nodes in the client and the server. Because this is possible between any two nodes in the body, it can create different compressions rates. (MDN web docs 2017b.)

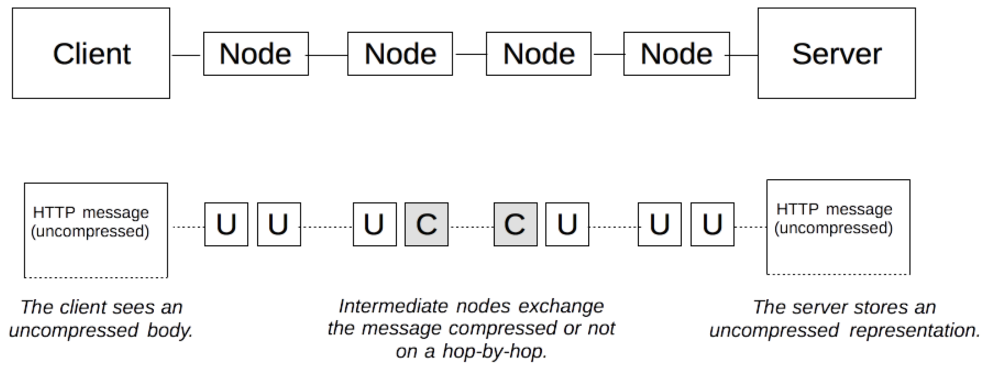

Client
Node
Node
Node
Node
Server
HTTP message (uncompressed)
U U
U C
C U
U U
HTTP message (uncompressed)
The client sees an uncompressed body.
Intermediate nodes exchange the message compressed or not on a hop-by-hop.
The server stores an uncompressed representation.

Figure 4. Hop-by-hop compression (MDN web docs [2018b].)

To use this compression between the two nodes, the node transmitting the request communicates to the other nodes using the Transfer-Encoding header. Other nodes will then react to the sending node with adequate method and apply it. The hop-by-hop compression is transparent for the client and the server and this is why it is rarely used. (MDN web docs 2017b.)

2.4 Images and transcoders

Image file formats. There are different types of digital image formats. Most familiar ones are JPEG (Joint Photographic Experts Group), PNG (Portable Network Graphics), TIFF (Tagged Image File Format), GIF (Graphics Interchange Format), SVG (Scalable Vector Graphics), RAW, BMP (Bitmap), DNG (Digital Negative Format) and PSD (Photoshop Document). SVG is included in this category but it is different from the others. It is based on XML markup. It means that SVG is a language that can exist in two dimensional vectors. (Dadfar 2018.)

JPEG. JPEG is used in many digital cameras which support this format as their output file. JPEG is a lossy quality format. This means that when taking pictures using this format there will be some quality loss already without using any compression software. This is useful when it is necessary to store as many images on the memory card as possible. (Dadfar 2018.)

PNG. PNG is a lossless format. This still doesn’t mean that file sizes will be big even though the compression rate is smaller. PNG results in small file sizes and good quality pictures. That is why PNG is ideal for internet use. It also allows partial or total transparency. This is ideal for creating logos and such effects as shadows. (Dadfar 2018.)

GIF. GIF files are lossless compression images that can be transparent. They can be used in animations but are limited to only 256 colours chosen from the 24-bit RGB color space. This is not ideal when wanting to create vibrant photos. (Dadfar 2018.)

SVG. SVG is a format that has more similarities with web pages than image formats. It can bend more than other formats and may be modified with a code, CSS or JavaScript. Vectors can be created in graphic editors, modified and saved. (Giltsoff, [Ref: 21.9.2018].)

Gulp (gulp.js). Gulp is a toolkit which automates functions. It helps the workflow by automating tasks. Gulp-imagemin is a tool designed to automate especially image compression tasks and it has a variety of different types of compression plugins. (gulpjs, [Ref: 24.9.2018].)

JPEG transcoders. Lossy and lossless compression use their own plugins. For lossless compression jpegtran plugin is used. It comes with the gulp-imagemin plugin and works by rearranging the compressed data. This means that there is no quality loss in the process. (GSP, 2014). Lossy compression plugin uses mozJPEG plugin. It is an encoder developed by Mozilla and it reduces file sizes of JPEG images. This means that there is quality loss in the process. (Mozilla, Github, [Ref: 24.9.2018].)

PNG transcoders. OptiPNG is a lossless compression transcoder. It comes with the gulp-imagemin plugin. It is a PNG optimizer and its purpose is to recompress image files without losing any information. (Truţa, C, 2008). For lossy compression pngquant is used. It can achieve 70% reduction in file sizes and it has high-quality RGBA conversions. (pngquant, [Ref: 24.9.2018].)

Gifsicle and Giflossy. Gifsicle is a multipurpose command-line tool for GIF images and animations. Giflossy is based on Gifsicle and it implements lossy compression. (kornelski/giflossy, [Ref: 24.9.2018].)

SVGO optimizer. SVG optimizer is a tool for optimizing SVG files. This plugin allows the SVG’s to be cleaned from useless comments, tags, and attributes. It basically removes unnecessary space to compress the image. (Guillaume 2015.)

2.5 Video codecs

A codec can compress and decompress digital media files that contain video or audio. Encoders perform compression and decoders will do the opposite. There are also containers that usually contain video- and audio codecs. Containers are video file formats. Different containers need specific kind of codecs. Common containers are AVI (.avi), MPEG-4 (.mp4), Flash Video (.flv), Ogg (.ogv), WebM (.webm), MOV (.mov), Windows Media Audio. (.wmv). (Ansoft Inc. 2018.)

Media formats in the web. Media formats in the web are provided in the <audio> and <video> element tags. HTML5 doesn’t support all media formats, like MOV files. WebM, Ogg, and MP4 are supported and can be displayed in the internet media players. (MDN web docs 2017f.)

FFmpeg as a multimedia tool. FFmpeg is meant for manipulating media files to get the wanted results. It includes codecs which are different kinds of compression algorithms or tasks that can be performed. (FFmpeg, [Ref: 5.7.2018].)

3 CLIENT- AND SERVER-SIDE TECHNIQUES

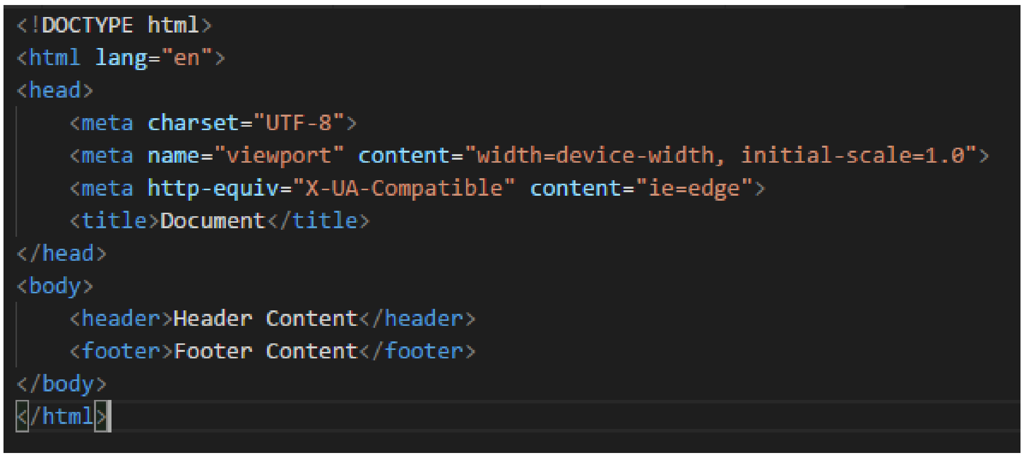
In this chapter I'll introduce the techniques used for building the application from the view of both the client and the server. Techniques used in the client-side focuses more on the user experience and user interface. The server-side consists of functionalities that were used to create the applications server.

3.1 CLIENT-SIDE

Client-side consists of different techniques. This chapter introduces techniques that were used in the visualisation of the website and what technique was used to interact with the server-side. This chapter’s figures shown are examples from the used techniques.

3.1.1 HTML

HyperText Markup Language, known as HTML, is the basic structure of the Web. It creates the content on the site and is often used with CSS and JavaScript to sculpture the appearance and functions of the website. HTML links sites to one another and creates so called “HyperText” using hyperlinks, and creates the World Wide Web linking everything together. Markup makes remarks to the content that is displayed. It uses elements that form the basic structure. (MDN web docs 2018a.) Example figure 5 shows an HTML template example with element tags.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <header>Header Content</header>
    <footer>Footer Content</footer>
</body>
</html>
```

Figure 5. HTML template

Basic HTML structure consists of element tags:

- “<!DOCTYPE html>” which tells this document to be HTML5.
- “<html> is the root of the file.
- “<head>” includes metadata.
- “<meta”> provides information about the document. It is not visible for the user and it contains such information as author web page description,. (w3schools.com 2018b.)
- “<title>” is the title of the file.
- “<body>” contains the visible content of the web page which the user can see in the browser.
- “<header>” and “<footer>” are tags that define visible content in the “<body>”. (w3schools.com 2018a.)

3.1.2 JavaScript and ECMAScript

JavaScript (JS) programming language is very lightweight. It is mostly used on the client-side that is seen on web pages. On the server-side it is used by languages such as node.js and Apache CouchDB. ECMAScript is a standard for JavaScript. All browsers support at least ECMAScript 3 and since 2012 modern browsers have been supported by ECMAScript 5.1. (MDN web docs 2018c.)

ECMAScript is a programming language that follows the object-oriented mode of operation. The purpose of this is to execute computations and to manipulate computational objects within a host environment. It was first developed as a scripting language. The purpose of this was to manipulate, customize and automate the functionality of an existing system. Since the using of the language has grown in many aspects it is not called a scripting language anymore but is now featured as a general-purpose programming language. (Ecma International 2018.)

In the client-side ECMAScript is used to include scripting code in the HTML web page and to add functionality to the windows, pop-ups, buttons, sliders, checkboxes and other objects. A host environment can use it to add events such as page and image loading, form submission and mouse scrolling, which are then displayed on the web page. (Ecma International 2018.)

The server-side can compute objects representing requests, clients and files or add permissions to download or upload content. Together the client- and server-side can communicate with each other and create a multipurpose Web application. (Ecma International 2018.)

3.1.3 MaterializeCSS

MaterializeCSS is a front-end framework, containing CSS and JavaScript files. MaterializeCSS comes with a minified and an unminified CSS. The unminified CSS is meant for development and to be minified after the wanted changes have been done to it. (Materialize 2017.)

Materialize library's CSS and JavaScript files include many different functions:

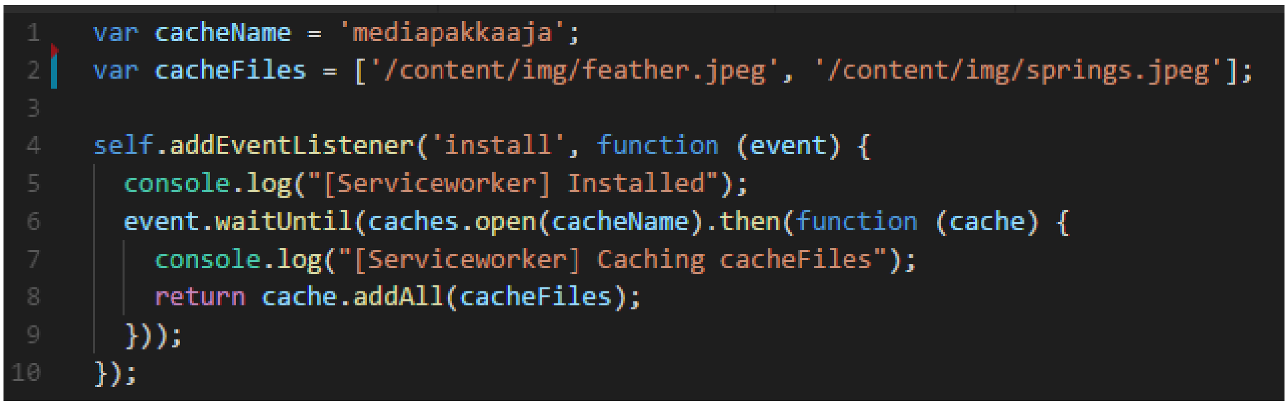
- Grid which aligns the content.
- Defined fonts, including font sizes.
- Components like buttons, nav-bars, side-nav etc.
- Support for Google material-icons.
- JavaScript functions to the components. (Materialize 2017.)

MaterializeCSS is using Material Design. That is Google's guideline for visual languages which synthesizes good design and innovation. The principles behind the design are:

- Material is the metaphor, which means that the design takes innovation from real world textures and applies them to its design.
- Bold, graphic, intentional, guides typography like icons, font-type and sizes. And uses Grid and scale for responsiveness.
- Motion provides meaning, interaction with the buttons, checkboxes and loading-bars, makes the user aware of what is happening.
- Flexible foundation, implementation of the components plug-ins and design elements that are made easy to mesh together with the code base.
- Cross-platform, upholds the same UI through, such platforms as Android, iOS, flutter, and the web. (Material Design 2018.)

3.1.4 Progressive Web App

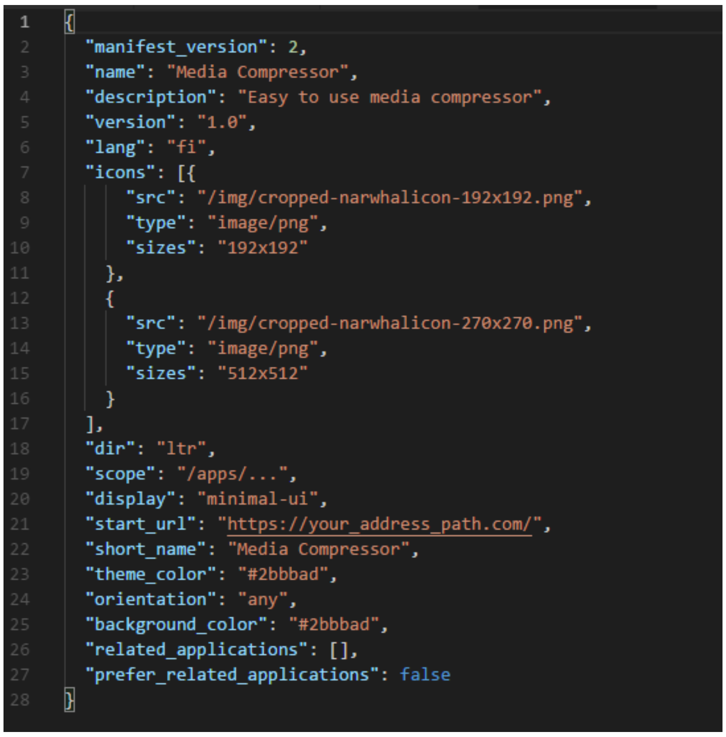
Progressive web app (PWA) is a way to make the application behave more like a native app, which loads content instantly and never shows user the delay. To achieve this kind of more native like app, a service worker is needed to perform and execute the wanted functions. The service worker can make the images on the web page to load faster when it gets a hold of the cache and then loads the images to the user's screen. Pre-caching of the images allows the content to be downloaded even when the Internet service is slow. It allows a smoother stage change when moving from one web page to another, because it can fetch the images from the cache and doesn't need to load them again. (Google Developers 2018b.) An example of this can be seen in the figure 6. It shows how the service worker is caching two images.



```
var cacheName = 'mediapakkaaja';
var cacheFiles = ['/content/img/feather.jpeg', '/content/img/springs.jpeg'];

self.addEventListener('install', function (event) {
  console.log("[Serviceworker] Installed");
  event.waitUntil(caches.open(cacheName).then(function (cache) {
    console.log("[Serviceworker] Caching cacheFiles");
    return cache.addAll(cacheFiles);
  }));
});
```

Figure 6. Service worker

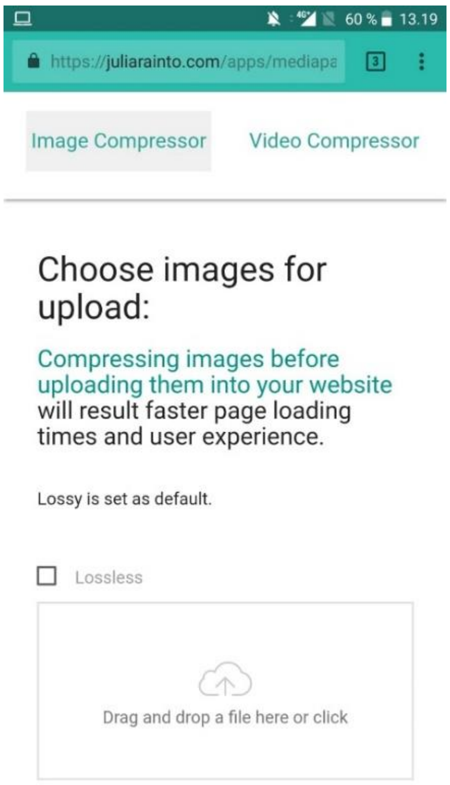
Manifest is one way to make the application more progressive. Manifest stores information about the application and allows the app to be installed to the user's home screen without using the app store. It also allows applcation to work offline and to receive push notifications if needed. (MDN web docs 2018g.) Example figure 7 shows what manifest can look like.



```json
{
  "manifest_version": 2,
  "name": "Media Compressor",
  "description": "Easy to use media compressor",
  "version": "1.0",
  "lang": "fi",
  "icons": [{
      "src": "/img/cropped-narwhalicon-192x192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/img/cropped-narwhalicon-270x270.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "dir": "ltr",
  "scope": "/apps/...",
  "display": "minimal-ui",
  "start_url": "https://your_address_path.com/",
  "short_name": "Media Compressor",
  "theme_color": "#2bbbad",
  "orientation": "any",
  "background_color": "#2bbbad",
  "related_applications": [],
  "prefer_related_applications": false
}
```

Figure 7. Applications manifest.json

Manifest is also a part of the progressive web app. The user will see the loading screen while page content is rendering in the background. This is done in the manifest.json file. (MDN web docs 2018g.)


60 % 13.19
https://juliarainto.com/apps/mediapa
Image Compressor
Video Compressor
Lossless
Drag and drop a file here or click

Choose images for upload:

Compressing images before uploading them into your website will result faster page loading times and user experience.

Lossy is set as default.

Figure 8. Determined theme color

An example of this can be seen in figure 8 which shows what the address bar looks like when the background color is determined. It will display the theme color in the address bar and give the application a more app like feeling. This is because the showing of the web application looks more of a native application.

3.2 SERVER-SIDE

Server side consists of techniques that creates the applications functionalities. This chapter introduces techniques that were used in the creation of the application server and what technique were used to interact with the client-side.

3.2.1 NodeJS

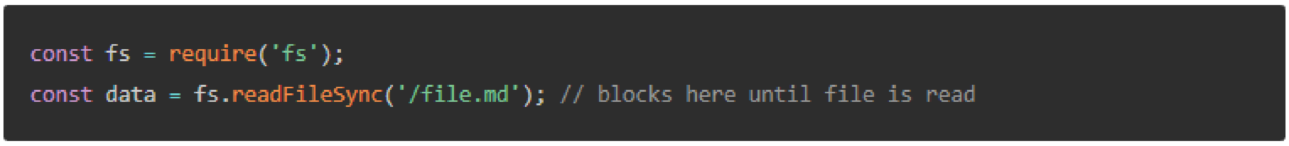
Node.js is an asynchronous event driven JavaScript runtime environment. NodeJS is built on Chrome's V8 JavaScript engine. It converts your code to a machine code that is easier and faster for the computer to read and analyse. As a Node run-time environment, it includes V8 engine, node API or node modules to execute written JavaScript. It was created to build network applications that are scalable not only in the browser, like JavaScript does, but also in the machine as an application separate from the browser. (Patel 2018.)

NodeJS extends JavaScript used for making website functions, to work like scripting languages does. Scripting languages are designed to communicate with other programming languages. Usually they work with languages like HTML, Java or C and they do not need to be compiled before deployment as they run from the bytecode or source code. (Techopedia Inc 2018.)

NodeJS is lightweight because it uses the non-blocking I/O model. The non-blocking I/O model has many advantages because it can handle multiple operations at a same time. Node.js also uses the open source library ecosystem known as npm. (Patel 2018.)

I/O calls in Node.js. A synchronous method is called blocking and an asynchronous method is called non-blocking. The synchronous method is called when Node.js process executes JavaScript and it waits until the non-JavaScript process is done. This occurs when the event loop is not able to continue because of an ongoing blocking operation. Because Node.js runs poorly in non-JavaScript operations, the Node.js standard library provides an asynchronous method for running operations in non-blocking way. (npmjs.com 2018.)

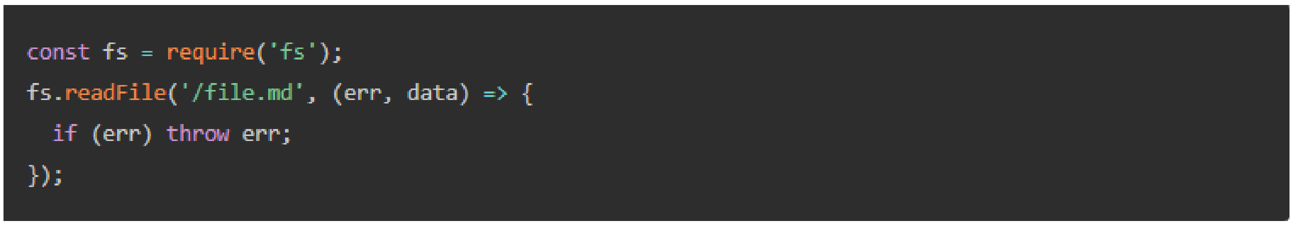
The difference between these two methods is error handling. In the synchronous example an additional JavaScript is blocking the execution until the process in done. Figure 9 shows synchronous file read in NodeJS. (npmjs.com 2018.)



```
const fs = require('fs');
const data = fs.readFileSync('/file.md'); // blocks here until file is read
```

Figure 9. Synchronous file read (Node.js Foundation, [Ref: 18.6.2018].)

In figure 10, callbacks in asynchronous version will perform an error handling and the developer will decide if it is shown or thrown. (npmjs.com 2018.)

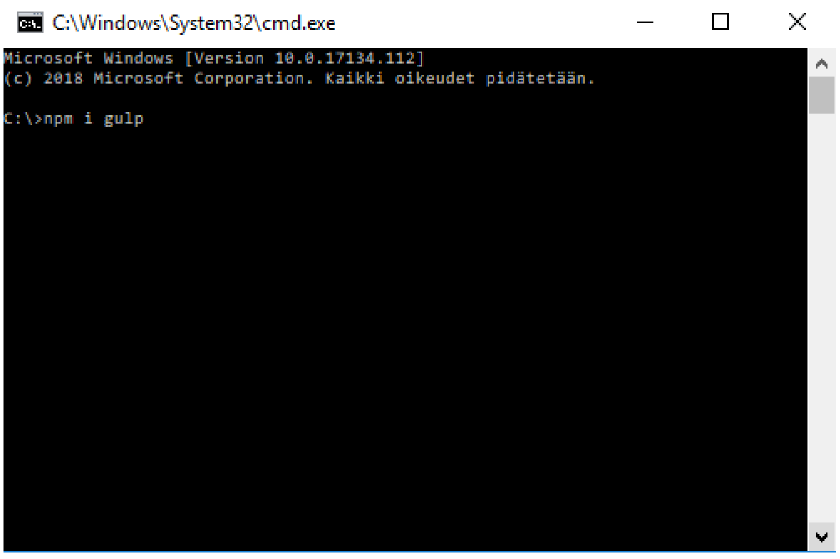


```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
});
```

Figure 10. Asynchronous file read (Node.js Foundation, [Ref: 18.6.2018].)

Node packaging manager (npm). Npm has about 3 billion downloads per week and 600 000 packages in its registry. Packages are building blocks of code that are contributed by developers and different organisations. Way to share or borrow packages is called npm. It is used in open-source and in private development. (npmjs.com 2018.)

Npm has three distinguishable components: the website, CLI and registry. The website is meant to be used to get handle of packages, profiles and managing them. CLI is a way to access and use npm from the terminal. Figure 11 shows an example how CLI can install gulp package using npm. Lastly the registry is a large public database that contains metadata and JavaScript software. (npmjs.com 2018.)



```
Microsoft Windows [Version 10.0.17134.112]
(c) 2018 Microsoft Corporation. Kaikki oikeudet pidätetään.

C:\>npm i gulp
```

Figure 11. Terminal view

Child process. Child process is NodeJS function. Child process provides the ability to execute `.bat` and `.cmd` files on Windows, Unix, Linux or macOS. There are functions where terminal is needed to execute commands. Child process allows execution of these functions within the JS file. (Node.js v10.11.0 Documentation, [Ref: 25.9.2018].)

3.2.2 HTTP Request

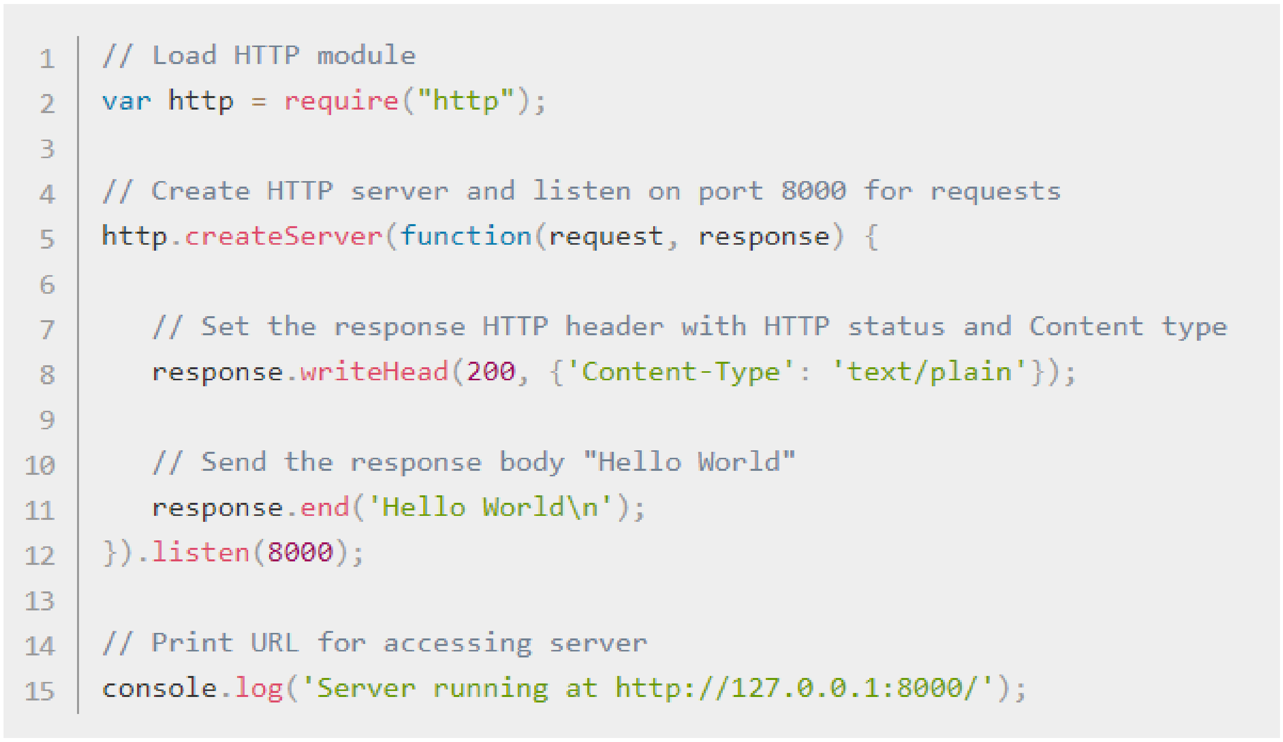
Hypertext Transfer Protocol (HTTP) was created for information to pass between web browser and web server. HTTP works as classical client-server. It is a protocol to transferring data between the browser and the server. HTTP opens a connection in the client and keeps it open until the process of making the request has been processed and then waits for a response from the server before closing the connection. (MDN web docs 2018d.)

HTTP -verbs, are a set of request methods, that indicates what kind of actions are to be called.

- “GET” request is used to retrieve data.
- “HEAD” does the same thing as GET without response body.
- “POST” request is used to submit data.
- “PUT” changes the current target with the requested one.
- “DELETE” deletes the current target.
- “CONNECT” forms a tunnel to the server that is recognized by the target.

- “OPTIONS” represents a request for information about the communication options available of the target.
- “TRACE” executes a message loop-back test to the target.
- “PATCH” creates modifications to the resource. (MDN web docs. 2018i.)

For the client to be able to communicate, a web server needs to be established. One way to do this is to use a Node HTTP package like shown in the figure 12. Once established, it can listen to the HTTP request on a specified URL. (MDN web docs 2018e.)



```
// Load HTTP module
var http = require("http");

// Create HTTP server and listen on port 8000 for requests
http.createServer(function(request, response) {

   // Set the response HTTP header with HTTP status and Content type
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body "Hello World"
   response.end('Hello World\n');
}).listen(8000);

// Print URL for accessing server
console.log('Server running at http://127.0.0.1:8000/');
```

Figure 12. Node HTTP package. (MDN web docs, [Ref: 18.6.2018].)

HTTPS. HTTPS is secured version of basic HTTP. Figure 13 shows difference between HTTP and HTTPS connection. Secure connection needs either SSL (Secure Sockets Layer) certificates or TLS (Transport Layer Security). Both the SSL and TLS protocols uses Public Key Infrastructure (PKI) system. They both use public- and private key to encrypt the connection between the user and the server. (Comodo CA Limited 2018.)

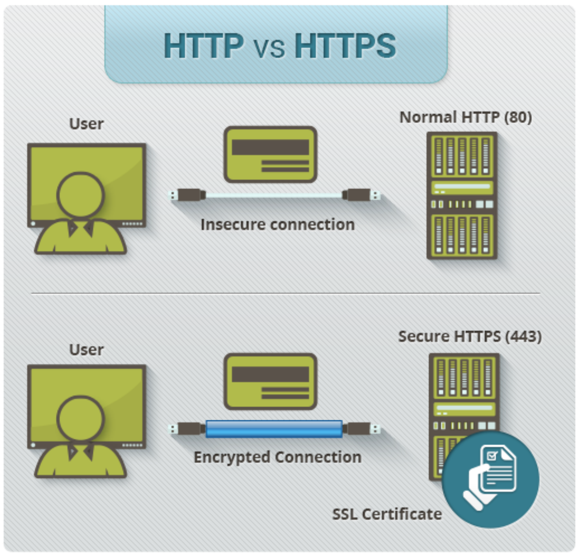
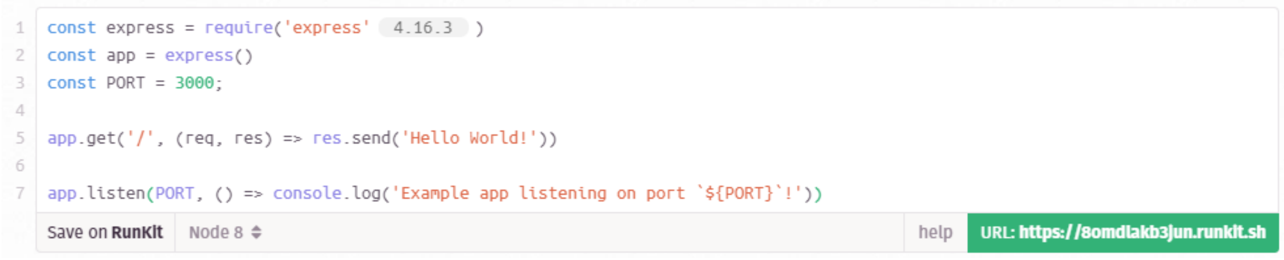


Figure 13. HTTP vs HTTPS (Comodo CA Limited 2018.)

Express. Express is a Node web framework. General purpose of express is to provide tools to different tasks. Rather than using only Node HTTP package, using a framework can simplify many tasks for you. Developers have made middleware packages to express to solve different kinds of web-development problems. (MDN web docs 2018e.)

In a normal website, a web application receives HTTP requests from the client. Based on the request method, received information is analysed and executed by the application. Figure 14 shows how to Express is initialized. (MDN web docs 2018e.)



```
const express = require('express'  4.16.3 )
const app = express()
const PORT = 3000;

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(PORT, () => console.log('Example app listening on port `${PORT}`!'))
```

Figure 14. Express example. (Express, [Ref: 30.10.2018].)

REST API. REST API also known as using Representational State Transfer Application. REST, or Restful API is predetermined styles and standards to follow, when performing HTTP requests in Web applications. These set of rules need to have the both client and the serve. Implementation of these standards happen in the client and the server. They are stateless which means they can work as separate operations from each other. It means that if the client has changes in its state or status, the server still works on its own and doesn't get affected from the client's changes. Or if the server goes through some changes, the client doesn't get affected by them. (Codeacademy 2018.)

Using REST you can make each side know if they are meant to be kept separated, or to send messages for them to be able to communicate with each other. Messages works as client sends requests to the server and it then sends responses back to the client's requests. (Codeacademy 2018.)

AJAX. AJAX also known as Asynchronous JavaScript and XML, is a method where object can communicate with the server using XMLHttpRequest. AJAX can send (POST) and receive (GET) data. With AJAX it is possible to exchange data and update a web page without refresh it. (MDN web docs 2018h.)

4 APPLICATION

Web applications have two sides. The client and the server. The “client” side is the visual part that works in browsers, like Google Chrome or Firefox. The “server” side works behind the client and sends data to the client and processes client’s requests.

Security is important when developing an application that will need to have access to user’s personal files. Security is also important when uploading and downloading content from the user’s device. Application uses HTTPS, UUID and function that is set to clear the uploaded and compressed folders from the server.

4.1 Application process

Figure 15 will visually explain the application process, to better understand the following:

- When application has been set up in the server, user will navigate the browser to the address that the application is located in.
- User will then choose either image compression or video-compression and file is uploaded in to the application.
- User will then choose the wanted compression options and will press the “compress files!” button.
- This will determine which routes folders compression route is to be executed. Lossless image compression will execute image_lossless.js – route. Otherwise image_lossy.js -route. Video compression is done in the video_compress.js – route.
- Process will then start transferring the file into the uploads folder. It will then start the selected compression process. When compression process is done the file will be transferred in to the compressed folder.
- The compressed file is then able to be downloaded with the users browser.

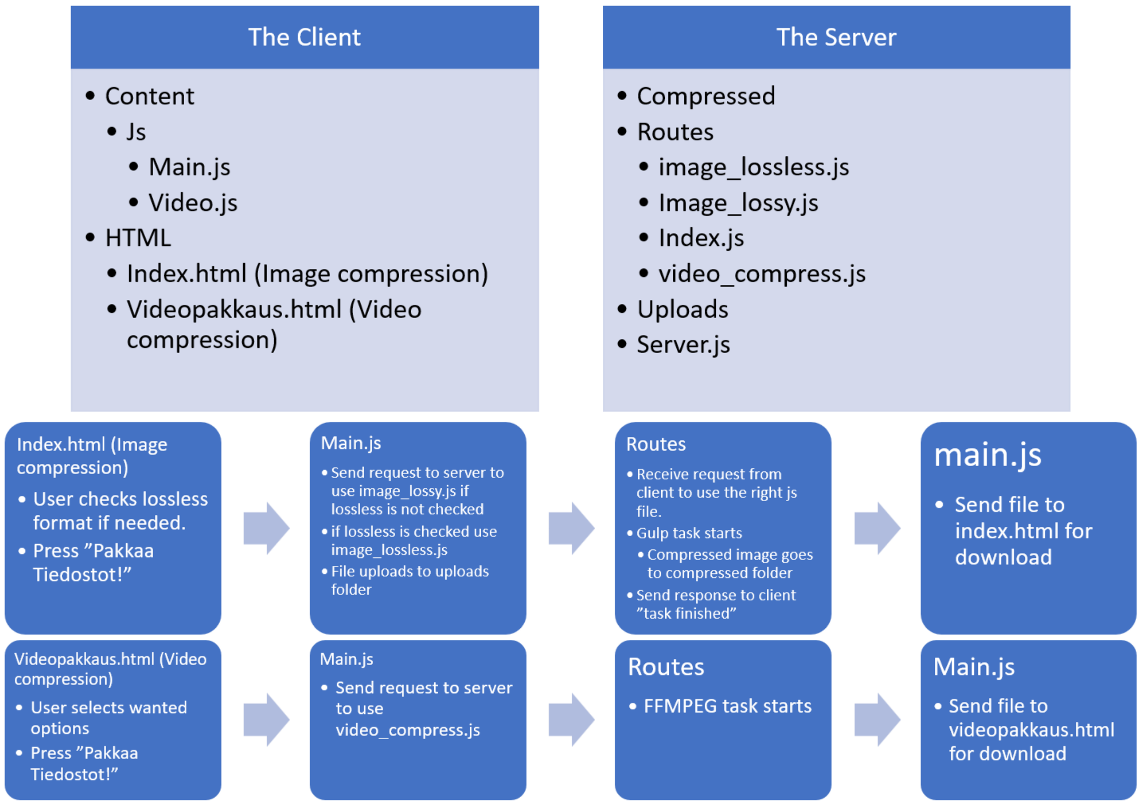
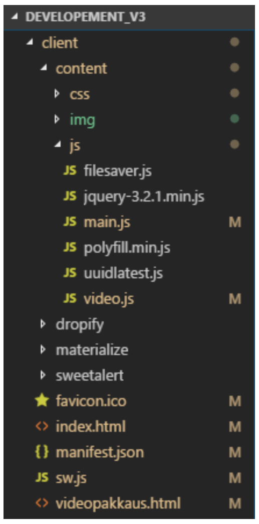

The Client
• Content
• Js
• Main.js
• Video.js
• HTML
• Index.html (Image compression)
• Videopakkaus.html (Video compression)
The Server
• Compressed
• Routes
• image_lossless.js
• Image_lossy.js
• Index.js
• video_compress.js
• Uploads
• Server.js
Index.html (Image compression)
• User checks lossless format if needed.
• Press "Pakkaa Tiedostot!"
Main.js
• Send request to server to use image_lossy.js if lossless is not checked
• if lossless is checked use image_lossless.js
• File uploads to uploads folder
Routes
• Receive request from client to use the right js file.
• Gulp task starts
• Compressed image goes to compressed folder
• Send response to client "task finished"
main.js
• Send file to index.html for download
Videopakkaus.html (Video compression)
• User selects wanted options
• Press "Pakkaa Tiedostot!"
Main.js
• Send request to server to use video_compress.js
Routes
• FFMPEG task starts
Main.js
• Send file to videopakkaus.html for download

Figure 15. Application process.

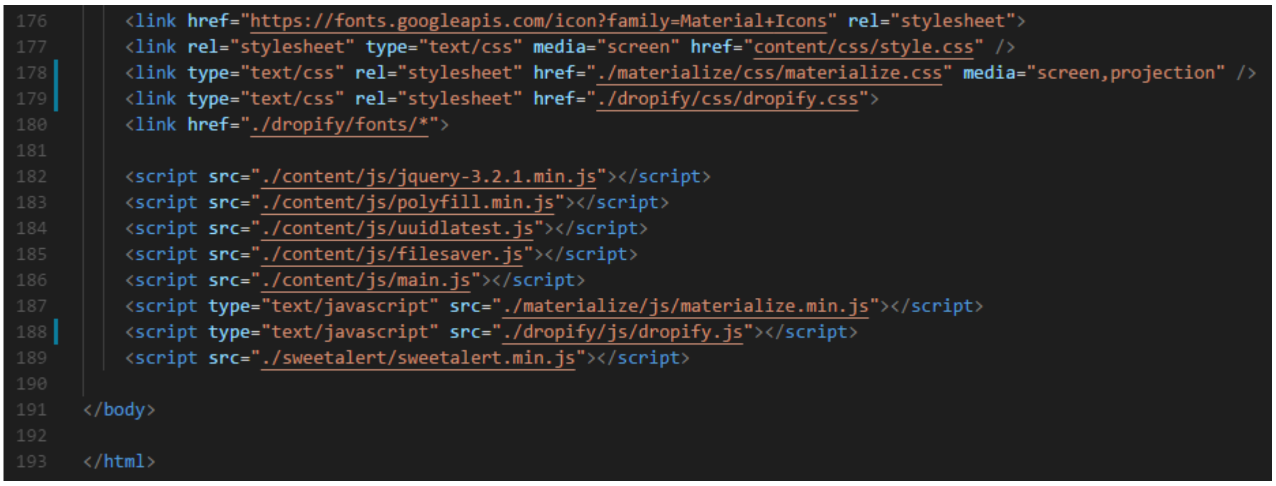
4.2 Client-side structure

Client-side consists of HTML, CSS and JavaScript. For the client to work it needs a browser that supports these techniques. Files are inserted to tree like structure like in the figure 16 and then they are accessed with written code in HTML. Figure 17 shows links and scripts that are accessed in the HTML.



```
DEVELOPEMENT_V3
  client
    content
      css
      img
      js
        filesaver.js
        jquery-3.2.1.min.js
        main.js                M
        polyfill.min.js
        uuidlatest.js
        video.js               M
    dropify
    materialize
    sweetalert
    favicon.ico                M
    index.html                 M
    manifest.json              M
    sw.js                      M
    videopakkaus.html          M
```

Figure 16. Client-side tree structure

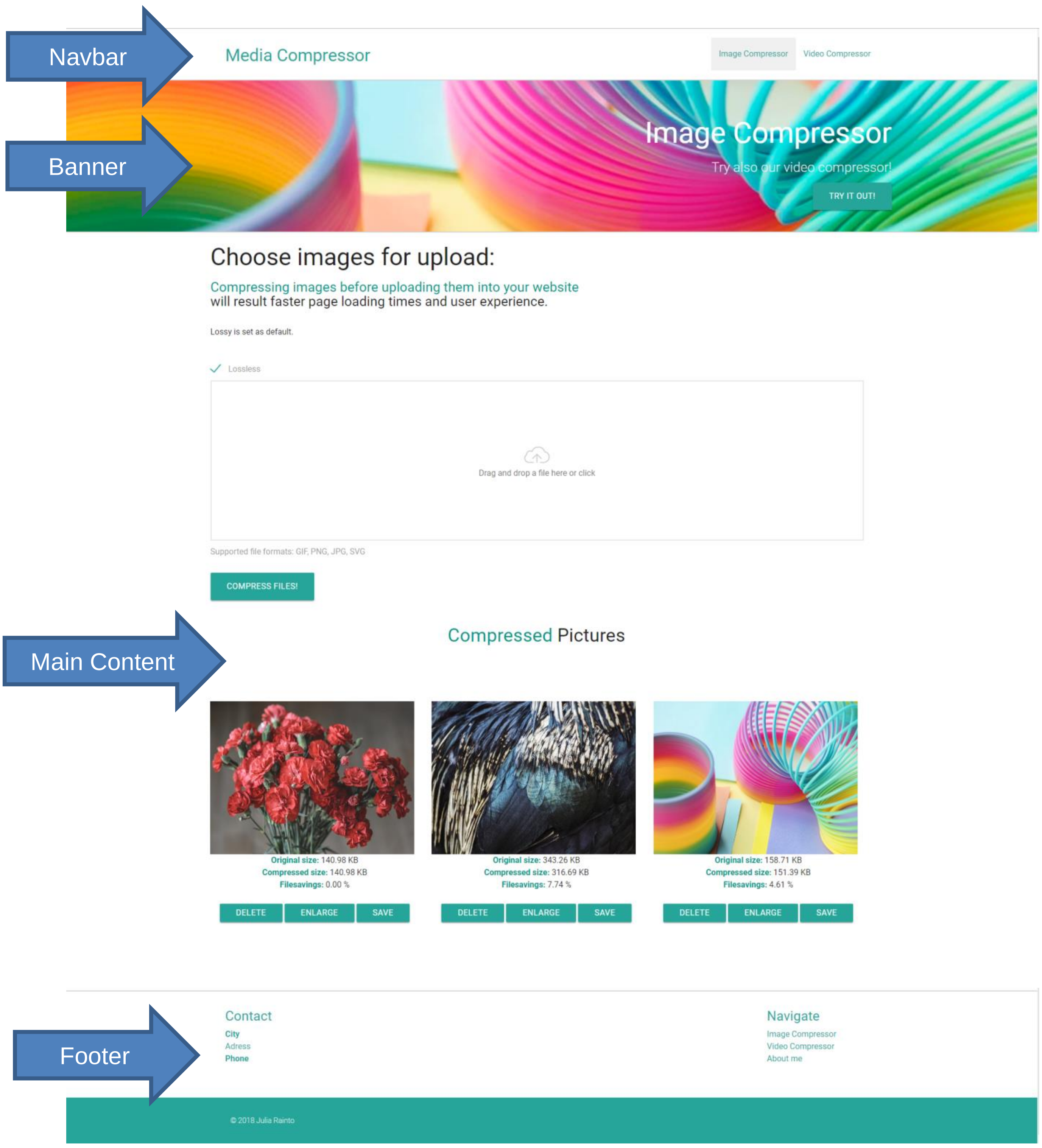


```html
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <link rel="stylesheet" type="text/css" media="screen" href="content/css/style.css" />
    <link type="text/css" rel="stylesheet" href="./materialize/css/materialize.css" media="screen,projection" />
    <link type="text/css" rel="stylesheet" href="./dropify/css/dropify.css">
    <link href="./dropify/fonts/*">

    <script src="./content/js/jquery-3.2.1.min.js"></script>
    <script src="./content/js/polyfill.min.js"></script>
    <script src="./content/js/uuidlatest.js"></script>
    <script src="./content/js/filesaver.js"></script>
    <script src="./content/js/main.js"></script>
    <script type="text/javascript" src="./materialize/js/materialize.min.js"></script>
    <script type="text/javascript" src="./dropify/js/dropify.js"></script>
    <script src="./sweetalert/sweetalert.min.js"></script>

</body>

</html>
```

Figure 17. Retrieving CSS and JS files.

The reason why <link> and <script> attributes are in the bottom of the <body>, and not in the <head>, is to optimize downloading content of the web page. This results to the <script> tags not stopping any parallel downloads during the process. If <script> tag is in the head of HTML file, it will stop any parallel downloads occurring in the web page until the script is done downloading. First meaningful paint is important especially in mobile view and <link> attribute is better to be placed at the bottom of the <body> if the performance is wanted to load more progressively. Other way is to do the opposite and place <link> tag inside the <head> tag. That will result for the web pages to appear faster in your web page, although performance will decrease.

Interface. Browser interface uses materializeCSS framework and two additional plugins, Dropify and SweetAlert. Application interface consist of navigation bar, banner, main content and a footer. Interface is also made responsive, so it works in different devices such as mobile phones, tablets and computers. It is also cross browser compatible and works in mobile and desktop browser. In the figure 18 you can see the browser view and in the figure 19 the mobile view.



Media Compressor

Image Compressor Video Compressor

Image Compressor

Try also our video compressor!

TRY IT OUT!

Choose images for upload:

Compressing images before uploading them into your website
will result faster page loading times and user experience.

Lossy is set as default.

✓ Lossless

Drag and drop a file here or click

Supported file formats: GIF, PNG, JPG, SVG

COMPRESS FILES!

Compressed Pictures

Original size: 140.98 KB
Compressed size: 140.98 KB
Filesavings: 0.00 %

DELETE ENLARGE SAVE

Original size: 343.26 KB
Compressed size: 316.69 KB
Filesavings: 7.74 %

DELETE ENLARGE SAVE

Original size: 158.71 KB
Compressed size: 151.39 KB
Filesavings: 4.61 %

DELETE ENLARGE SAVE

Contact
City
Adress
Phone

Navigate
Image Compressor
Video Compressor
About me

© 2018 Julia Rainto

Figure 18. View of the website.

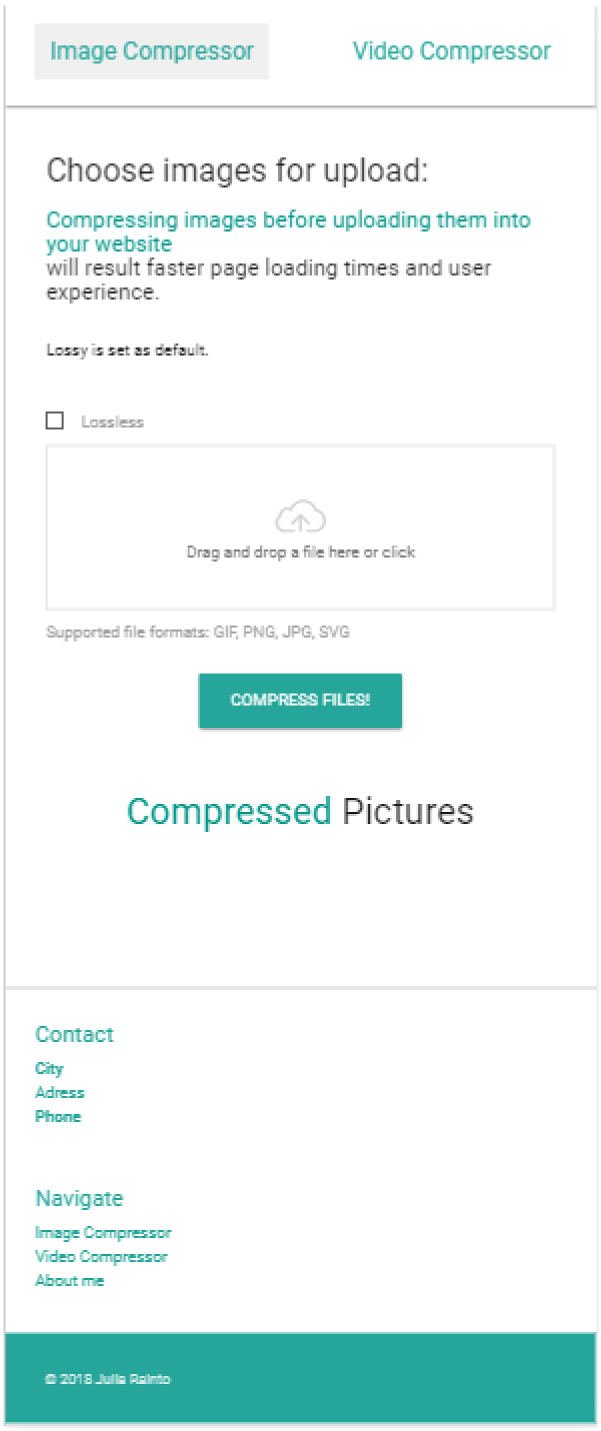


Image Compressor Video Compressor

Choose images for upload:

Compressing images before uploading them into your website
will result faster page loading times and user experience.

Lossy is set as default.

☐ Lossless

Drag and drop a file here or click

Supported file formats: GIF, PNG, JPG, SVG

COMPRESS FILES!

Compressed Pictures

Contact
City
Adress
Phone

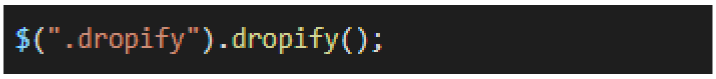
Navigate
Image Compressor
Video Compressor
About me

© 2018 Julia Rainto

Figure 19. Mobile view.

Plugins. Dropify and SweetAlert are additional plugins. That creates more fluent user experience with indicators and easier file upload.

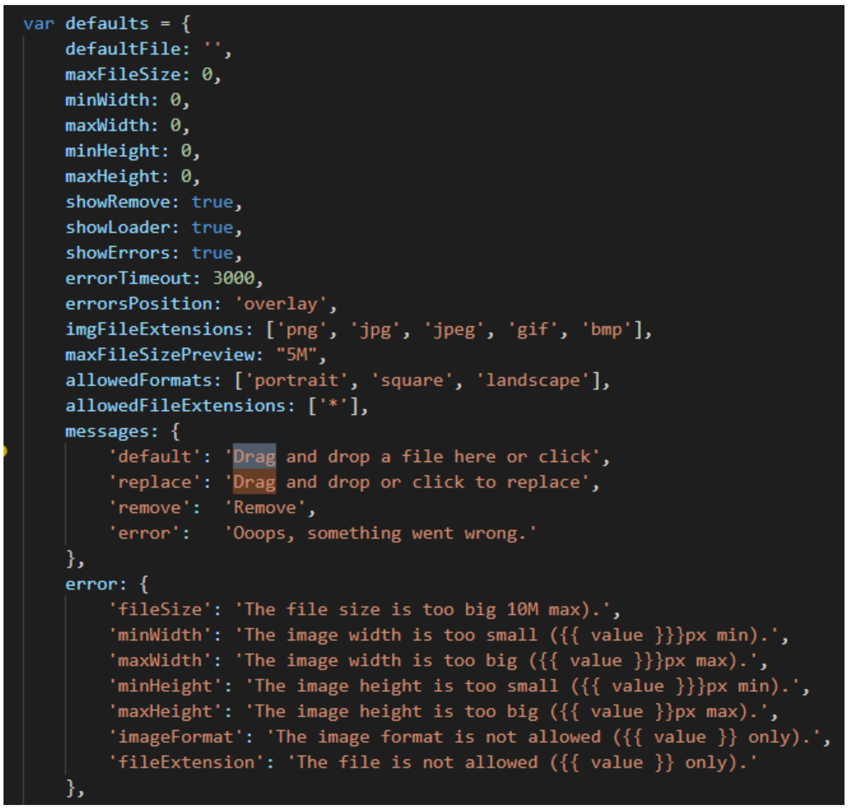
Dropify. Dropify is file input plugin that uses jQuery to work. It comes with CSS, fonts and JavaScript files. For it to work you need to include dropify's source packages to your project and then initialize the project (Figure 20).



```
$(".dropify").dropify();
```

Figure 20. Dropify jQuery.

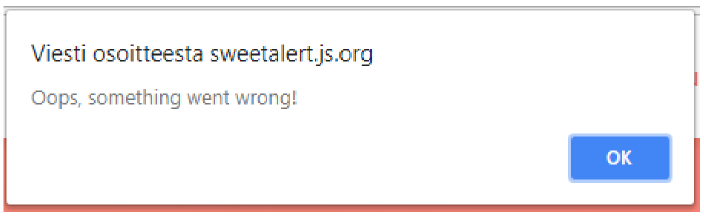
There are different kind of options to make dropify suitable for your needs. JavaScript configuration file is meant to be modified and then minified after development. Modifying JS file you can change the language or maximum file size of the uploaded document. Figure 21 shows how JS file can be modified.



```
var defaults = {
    defaultFile: '',
    maxFileSize: 0,
    minWidth: 0,
    maxWidth: 0,
    minHeight: 0,
    maxHeight: 0,
    showRemove: true,
    showLoader: true,
    showErrors: true,
    errorTimeout: 3000,
    errorsPosition: 'overlay',
    imgFileExtensions: ['png', 'jpg', 'jpeg', 'gif', 'bmp'],
    maxFileSizePreview: "5M",
    allowedFormats: ['portrait', 'square', 'landscape'],
    allowedFileExtensions: ['*'],
    messages: {
        'default': 'Drag and drop a file here or click',
        'replace': 'Drag and drop or click to replace',
        'remove':  'Remove',
        'error':   'Ooops, something went wrong.'
    },
    error: {
        'fileSize': 'The file size is too big 10M max).',
        'minWidth': 'The image width is too small ({{ value }}}px min).',
        'maxWidth': 'The image width is too big ({{ value }}}px max).',
        'minHeight': 'The image height is too small ({{ value }}}px min).',
        'maxHeight': 'The image height is too big ({{ value }}px max).',
        'imageFormat': 'The image format is not allowed ({{ value }} only).',
        'fileExtension': 'The file is not allowed ({{ value }} only).'
    },
```

Figure 21. Dropify JS file.

SweetAlert. SweetAlert is a decorative way to show alerts. The alert dialog is meant to make sure information comes for the user. Normal alert shows a message and additional OK button. SweetAlert makes the popup window more appealing. Figure 22 shows what the normal alert looks like and figure 23 shows what SweetAlert looks like.



Viesti osoitteesta sweetalert.js.org

Oops, something went wrong!

Figure 22. Normal alert.

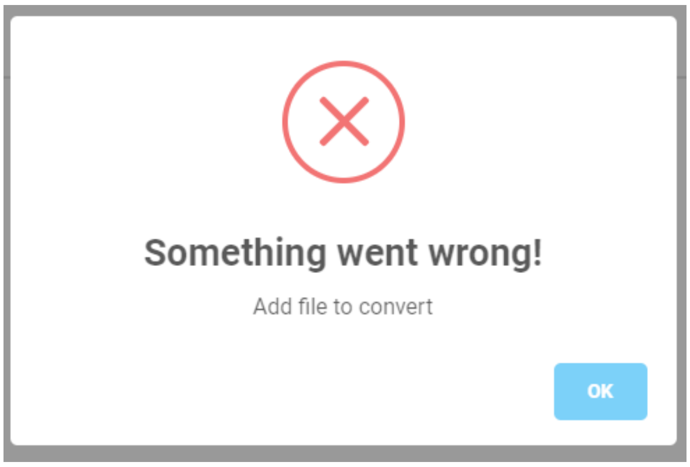
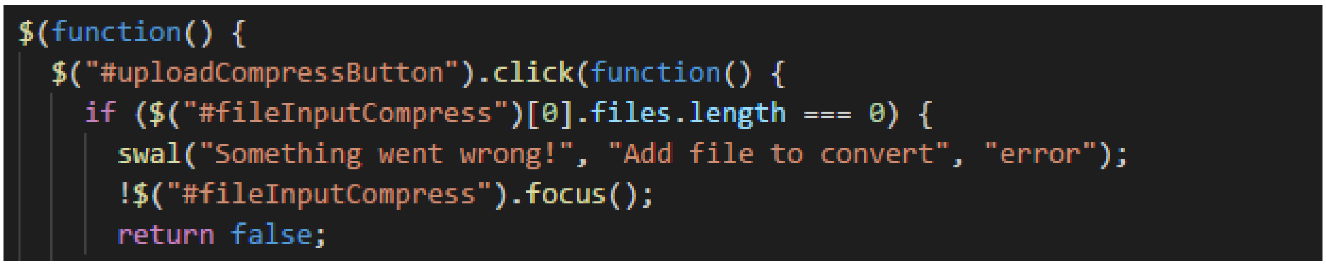


Figure 23. SweetAlert

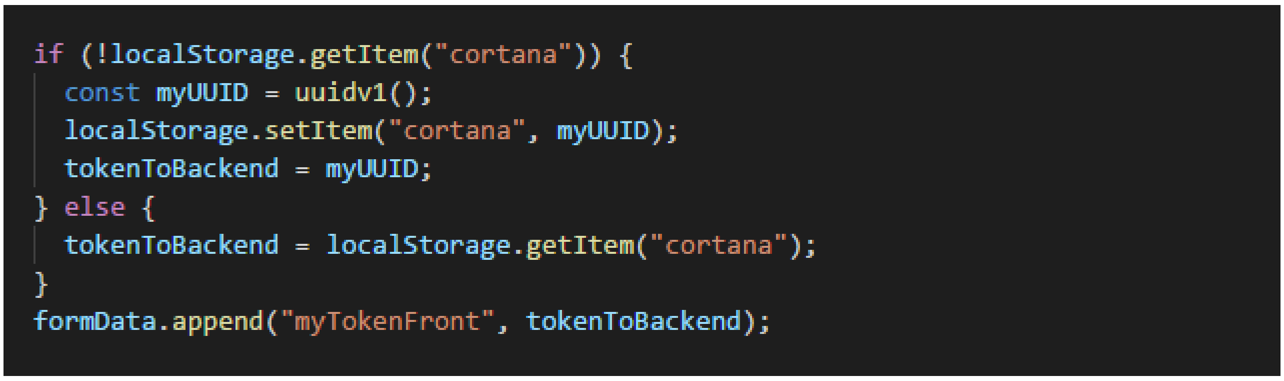
For the plugin to work you need to include sweetalert.js document to you HTML file. It is then initialized with jQuery in the main.js and video.js -files. Figure 24 shows that the jQuery will notify the user, if the user doesn't have permission to download the content.



```
$(function() {
  $("#uploadCompressButton").click(function() {
    if ($("#fileInputCompress")[0].files.length === 0) {
      swal("Something went wrong!", "Add file to convert", "error");
      !$("#fileInputCompress").focus();
      return false;
```

Figure 24. SweetAlert code.

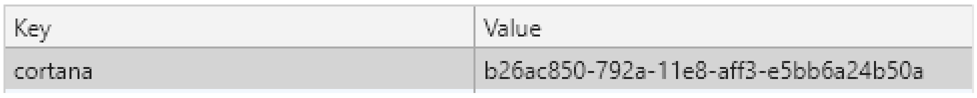
UUID. UUID is a way to generate random 128-bit value called token (Figure 32).



```
if (!localStorage.getItem("cortana")) {
  const myUUID = uuidv1();
  localStorage.setItem("cortana", myUUID);
  tokenToBackend = myUUID;
} else {
  tokenToBackend = localStorage.getItem("cortana");
}
formData.append("myTokenFront", tokenToBackend);
```

Figure 25. UUID token.

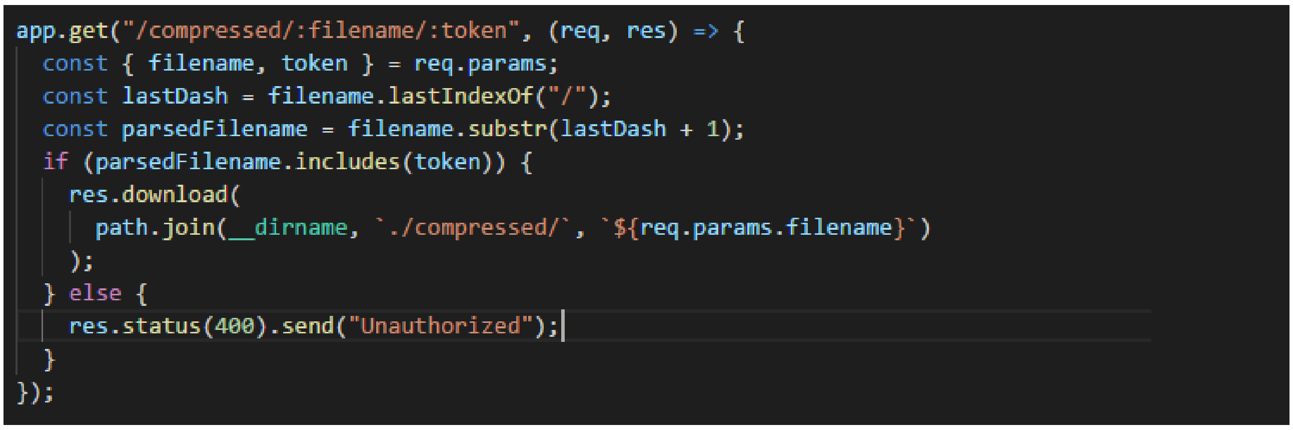
It is used for transactions and additional security and uniqueness. Application will generate personal token for the user and store it in local storage (Figure 33).



Key	Value
cortana	b26ac850-792a-11e8-aff3-e5bb6a24b50a

Figure 26. UUID generated token.

When uploading file to the applications server the token will be added in the file. When the compression task is complete, user can download content from the server. The server will then check that the token in the local storage and in the file will match. If they do, server allows the content to be downloaded from the server (Figure 34). If the token will fail to match the one in the local storage an error will occur. Unauthorized file will be then removed from the server after certain time period.

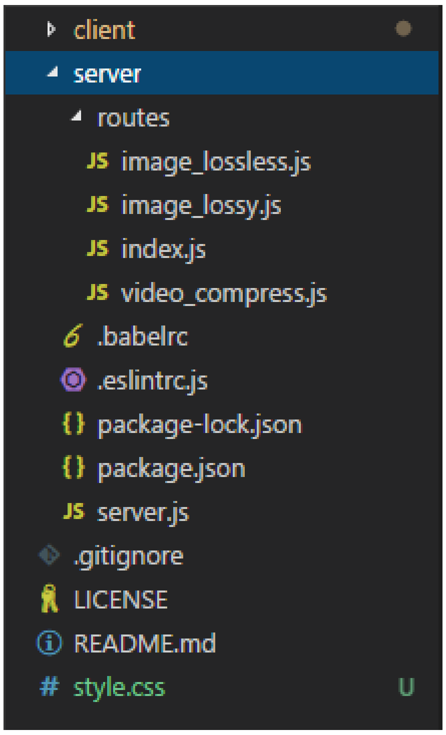


```
app.get("/compressed/:filename/:token", (req, res) => {
  const { filename, token } = req.params;
  const lastDash = filename.lastIndexOf("/");
  const parsedFilename = filename.substr(lastDash + 1);
  if (parsedFilename.includes(token)) {
    res.download(
      path.join(__dirname, `./compressed/`, `${req.params.filename}`)
    );
  } else {
    res.status(400).send("Unauthorized");
  }
});
```

Figure 27. Check UUID token.

4.3 Server-side structure

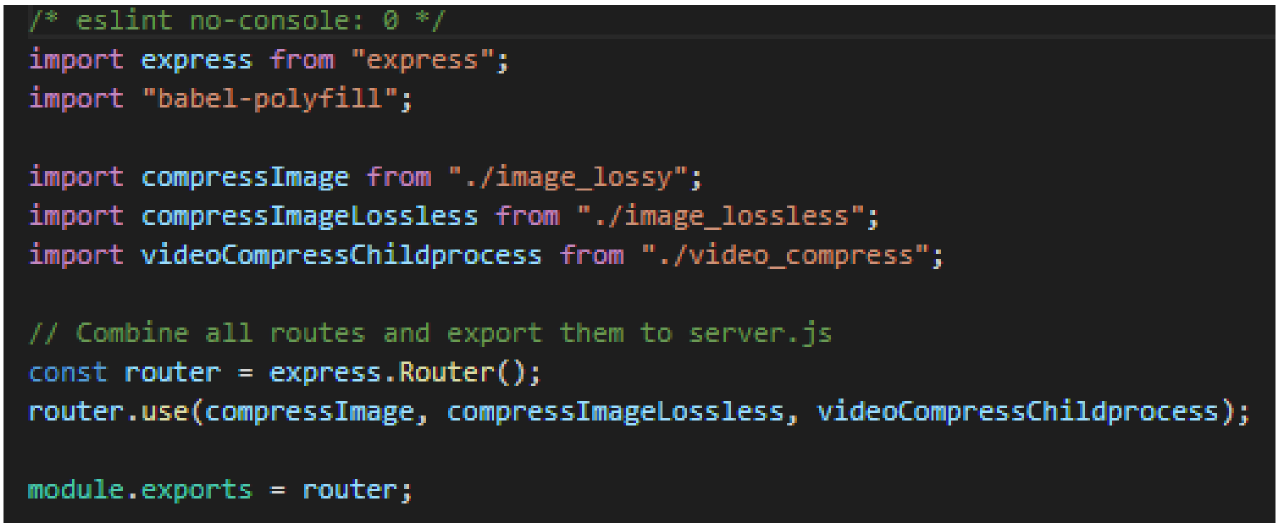
The server-side has its own tree structure shown in the figure 25. Server-side consists of server.js -file, folders for uploaded and compressed content and Routes -folder. Routes - folder includes image_lossless.js, image_lossy.js, index.js and video_compress.js -files. Uploads and compressed folders store media files. Uploaded media goes to the uploads folder and after compression process is done, the content is stored in the compressed folder. From there the content is ready to be downloaded to the user’s device.



```
client
server
  routes
    image_lossless.js
    image_lossy.js
    index.js
    video_compress.js
  .babelrc
  .eslintrc.js
  package-lock.json
  package.json
  server.js
.gitignore
LICENSE
README.md
style.css                U
```

Figure 28. Server-side tree structure.

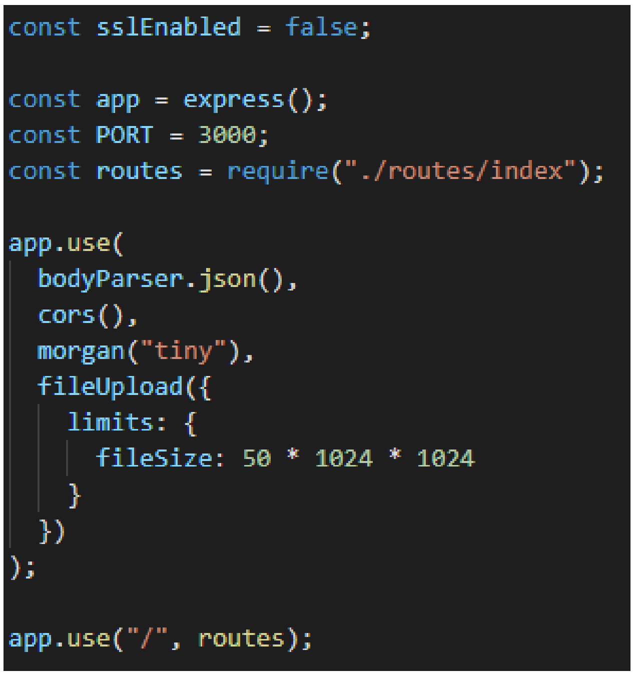
Figure 26 shows the index.js -file. Its purpose is to combine the routes in the file and pass them to server.js. This means the root path will then remain the same for all the endpoints. Import tags in the beginning are there to import functions, which are exported by external modules. Those can be the npm packages that has been installed or in this case the other JavaScript files to be routed.



```
/* eslint no-console: 0 */
import express from "express";
import "babel-polyfill";

import compressImage from "./image_lossy";
import compressImageLossless from "./image_lossless";
import videoCompressChildprocess from "./video_compress";

// Combine all routes and export them to server.js
const router = express.Router();
router.use(compressImage, compressImageLossless, videoCompressChildprocess);

module.exports = router;
```

Figure 29. index.js.

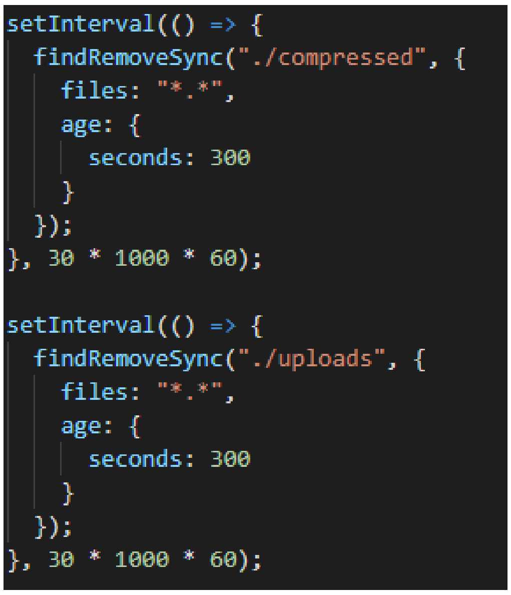
Server.js file has the HTTP server created by using express. Figure 27 will show how express is initialized in the server.js file.



```
const sslEnabled = false;

const app = express();
const PORT = 3000;
const routes = require("./routes/index");

app.use(
  bodyParser.json(),
  cors(),
  morgan("tiny"),
  fileUpload({
    limits: {
      fileSize: 50 * 1024 * 1024
    }
  })
);

app.use("/", routes);
```

Figure 30. Express initialized.

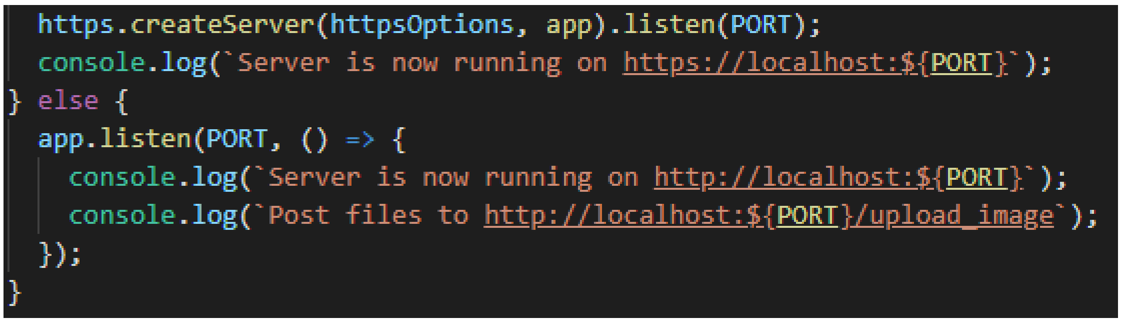
Server.js file also performs tasks like clearing uploaded and compressed files from their folders. In the figure 28 the function will remove files from the folders when the file has been in the server for 30 minutes. This secures that files don't get in the wrong hands and don't fill out the server from media files.



```
setInterval(() => {
  findRemoveSync("./compressed", {
    files: "*.*",
    age: {
      seconds: 300
    }
  });
}, 30 * 1000 * 60);

setInterval(() => {
  findRemoveSync("./uploads", {
    files: "*.*",
    age: {
      seconds: 300
    }
  });
}, 30 * 1000 * 60);
```

Figure 31. setInterval.

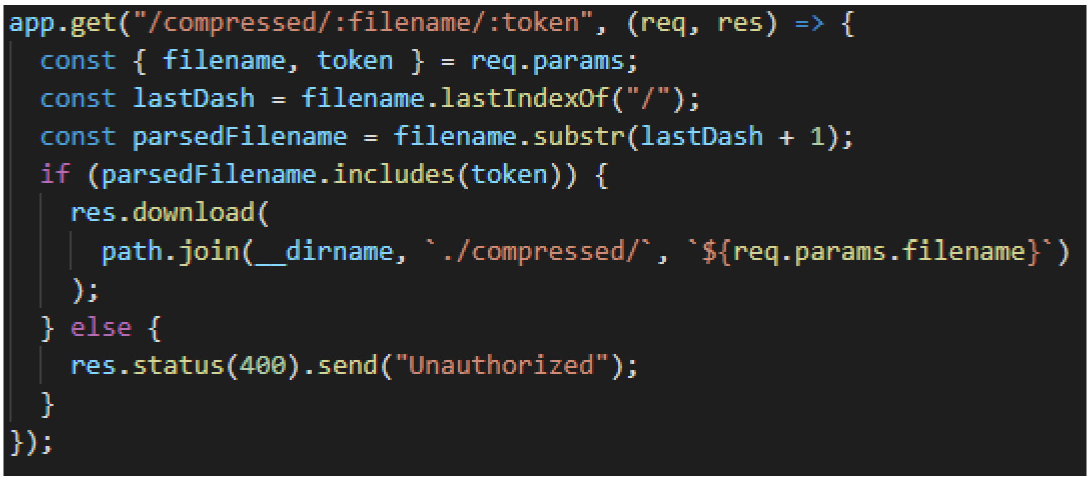
It also creates a local server when needed to run the application locally from your computer (Figure 29). This is usually used in development before applying the application to the actual server for production use.



```
  https.createServer(httpsOptions, app).listen(PORT);
  console.log(`Server is now running on https://localhost:${PORT}`);
} else {
  app.listen(PORT, () => {
    console.log(`Server is now running on http://localhost:${PORT}`);
    console.log(`Post files to http://localhost:${PORT}/upload_image`);
  });
}
```

Figure 32. Local server.

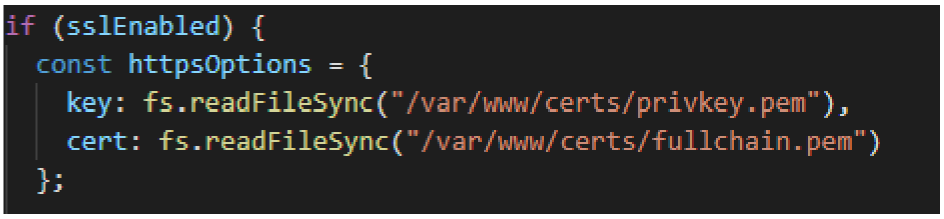
Other functions for the server.js -file are to check if the user downloading the content has the rights to download that file from the server (Figure 30).



```
app.get("/compressed/:filename/:token", (req, res) => {
  const { filename, token } = req.params;
  const lastDash = filename.lastIndexOf("/");
  const parsedFilename = filename.substr(lastDash + 1);
  if (parsedFilename.includes(token)) {
    res.download(
      path.join(__dirname, `./compressed/`, `${req.params.filename}`)
    );
  } else {
    res.status(400).send("Unauthorized");
  }
});
```

Figure 33. Checking for privileges.

Server.js -file also applies certification for secure SSL connection to the server. SSL certification is generated and then inserted to the wanted server path. Application will then call the path for certificates in the server and generate a secure connection. Figure 31 shows that the certifications are installed in the route: /var/www/certs/.



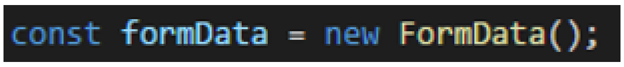
```
if (sslEnabled) {
  const httpsOptions = {
    key: fs.readFileSync("/var/www/certs/privkey.pem"),
    cert: fs.readFileSync("/var/www/certs/fullchain.pem")
  };
```

Figure 34. SSL Certificate access path.

4.4 Sending data to the server

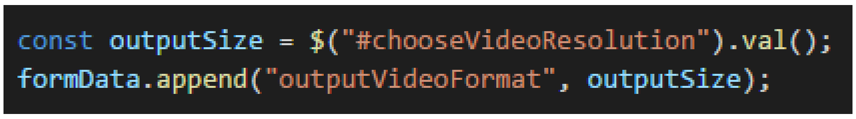
The data sent between the client and the server consists of uploaded medias and user chosen options. The technique used to send the data is AJAX. The application uses HTTP request (XHR) to send data to the server with a FormData -interface. FormData is the full package of information and data to be sent. FormData.append is a function call that adds more data to the package. XMLHttpRequest allows to send data to the server. Tässä saisi olla jonkinlainen johtolause alla olevaan luetelmaan.

1. FormData is formed (Figure 35).
2. Data is fetched from an html input and added to the formData by formData.append. (Figure 36.)
3. XMLHttpRequest is defined (Figure 37).
4. XMLHttpRequest events are formed (Figure 38).
5. XMLHttpRequest is given a address where to POST the formData -object. (Figure 39).
6. Data is sent to the server (Figure 40).



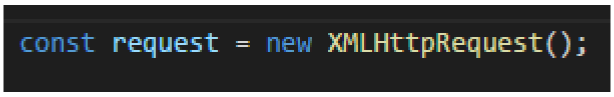
```
const formData = new FormData();
```

Figure 35. FormData.



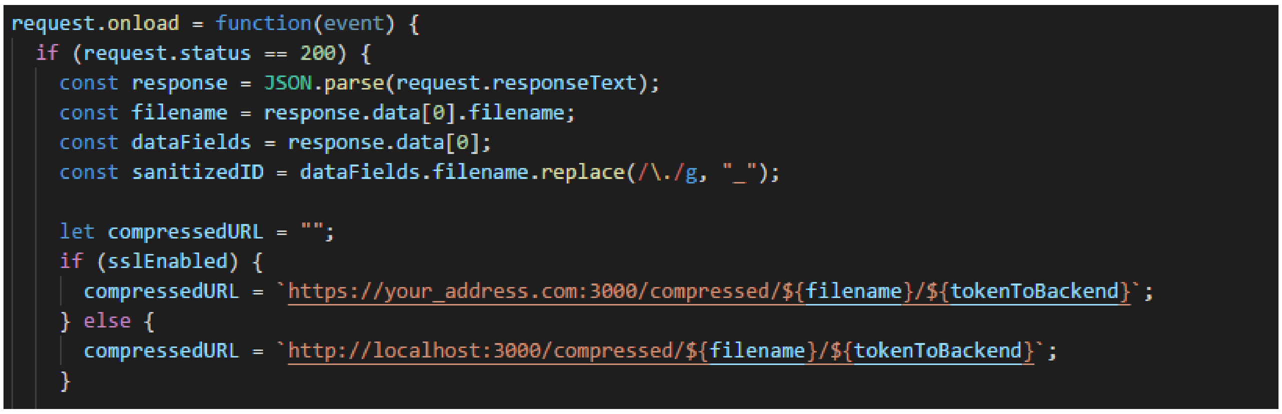
```
const outputSize = $("#chooseVideoResolution").val();
formData.append("outputVideoFormat", outputSize);
```

Figure 36. Formdata.append.



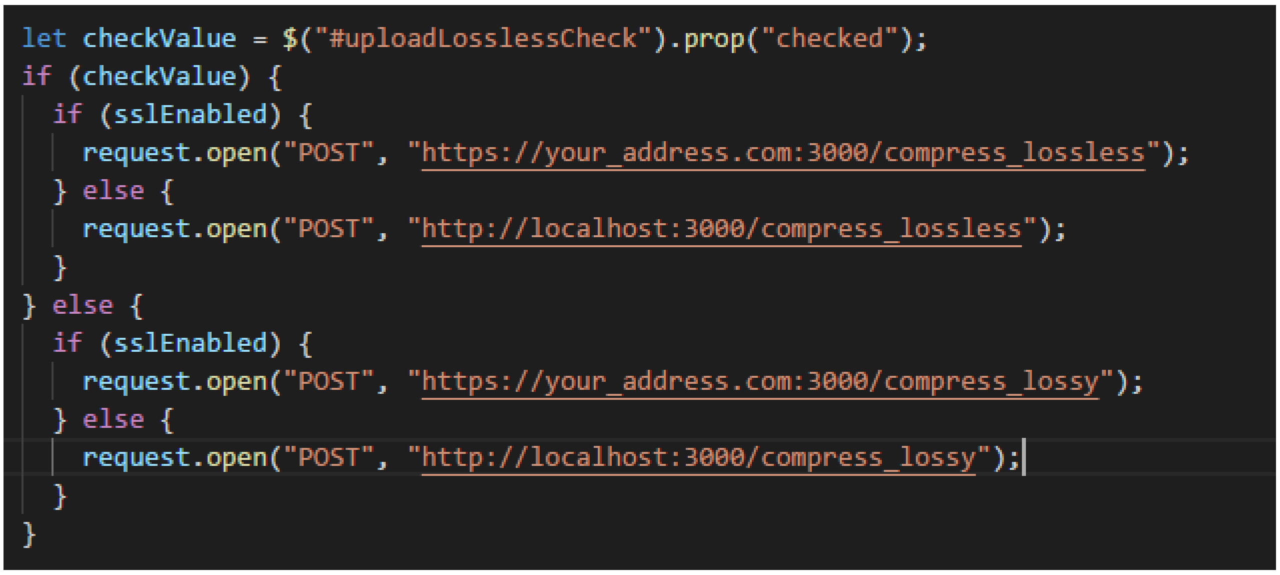
```
const request = new XMLHttpRequest();
```

Figure 37. XMLHttpRequest.



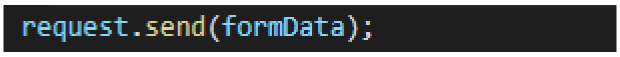
```
request.onload = function(event) {
  if (request.status == 200) {
    const response = JSON.parse(request.responseText);
    const filename = response.data[0].filename;
    const dataFields = response.data[0];
    const sanitizedID = dataFields.filename.replace(/\./g, "_");

    let compressedURL = "";
    if (sslEnabled) {
      compressedURL = `https://your_address.com:3000/compressed/${filename}/${tokenToBackend}`;
    } else {
      compressedURL = `http://localhost:3000/compressed/${filename}/${tokenToBackend}`;
    }
```

Figure 38. XMLHttpRequest event.



```
let checkValue = $("#uploadLosslessCheck").prop("checked");
if (checkValue) {
  if (sslEnabled) {
    request.open("POST", "https://your_address.com:3000/compress_lossless");
  } else {
    request.open("POST", "http://localhost:3000/compress_lossless");
  }
} else {
  if (sslEnabled) {
    request.open("POST", "https://your_address.com:3000/compress_lossy");
  } else {
    request.open("POST", "http://localhost:3000/compress_lossy");
  }
}
```

Figure 39. POST request.



```
request.send(formData);
```

Figure 40. Send formData.

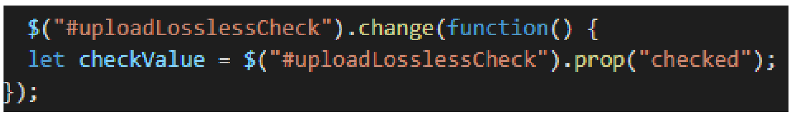
4.5 Image compression

The image compression uses Gulp as its toolkit to perform the image compression task. It automates a set of functions which are performed in the script. The Gulp acts as a task manager and the plugins as the tasks to be performed. The Gulp plugins used in the application are OptiPNG, PngQuant, Zopfli, Giflossy, Jpegtran, Mozjpeg, svgo. They are separate libraries with specific compression algorithms for different image formats. For example, PngQuant is meant for compressing PNG formats and Jpegtran is meant for compressing JPEG formats.

JPEG, PNG, GIF and SVG formats were used in this application, because the old software, which the company used, supported these formats.

Image compression. Image compression task happens in the Routes folder, in the image_lossless.js and image_lossy.js. The client-side determines which script is used, and which compression task will be performed. Figure 41 shows an option in the user interface and the user will check the CheckBox to perform either lossy or lossless compression. ChecBox needs to be initialized in JavaScript for it to work. Figure 42 shows how the checkbox button is initialized with jQuery.

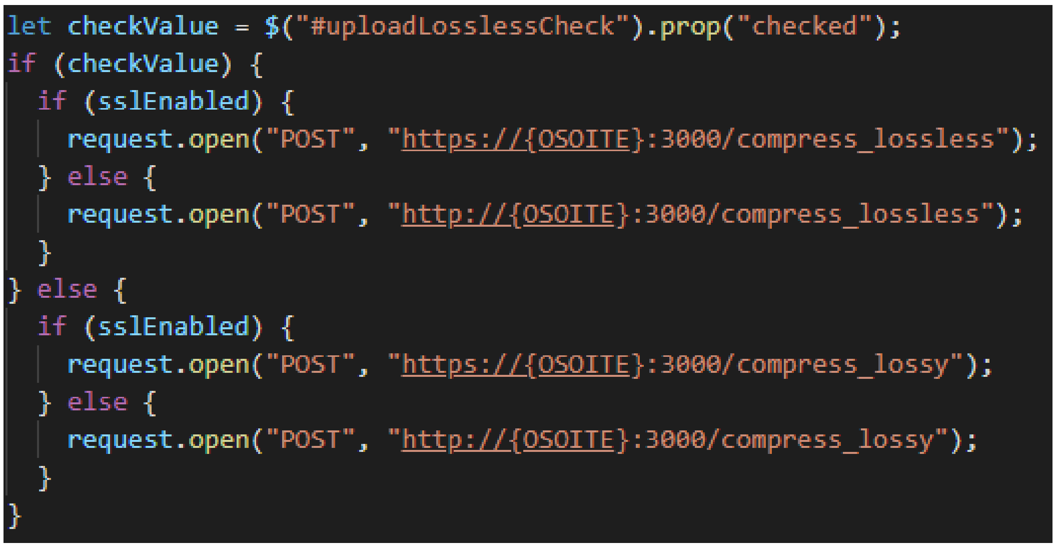
Figure 41. Checkbox.



```
  $("#uploadLosslessCheck").change(function() {
  let checkValue = $("#uploadLosslessCheck").prop("checked");
});
```

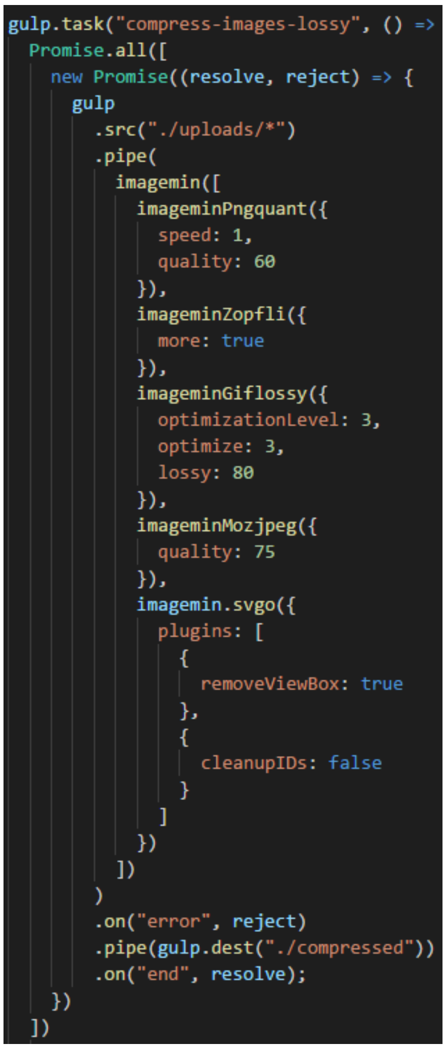
Figure 42. CheckValue.

When the user pushes the submit button, the application will check the value of the CheckBox button. Figure 43 shows the following: If the user doesn't check the lossless option, the POST request goes to image_lossy.js and it will then perform the gulp task shown in the figure 44. If the user checks lossless compression, the POST request goes to image_lossless.js and will perform the task shown in the figure 45.



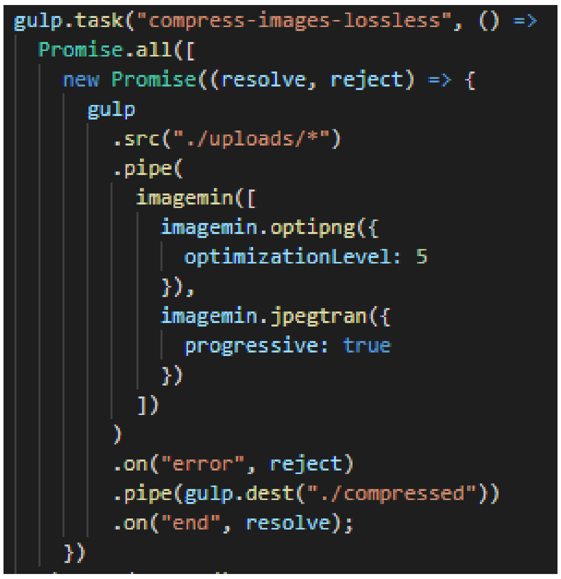
```
let checkValue = $("#uploadLosslessCheck").prop("checked");
if (checkValue) {
  if (sslEnabled) {
    request.open("POST", "https://{OSOITE}:3000/compress_lossless");
  } else {
    request.open("POST", "http://{OSOITE}:3000/compress_lossless");
  }
} else {
  if (sslEnabled) {
    request.open("POST", "https://{OSOITE}:3000/compress_lossy");
  } else {
    request.open("POST", "http://{OSOITE}:3000/compress_lossy");
  }
}
```

Figure 43. Send to route.



```
gulp.task("compress-images-lossy", () =>
  Promise.all([
    new Promise((resolve, reject) => {
      gulp
        .src("./uploads/*")
        .pipe(
          imagemin([
            imageminPngquant({
              speed: 1,
              quality: 60
            }),
            imageminZopfli({
              more: true
            }),
            imageminGiflossy({
              optimizationLevel: 3,
              optimize: 3,
              lossy: 80
            }),
            imageminMozjpeg({
              quality: 75
            }),
            imagemin.svgo({
              plugins: [
                {
                  removeViewBox: true
                },
                {
                  cleanupIDs: false
                }
              ]
            })
          ])
        )
        .on("error", reject)
        .pipe(gulp.dest("./compressed"))
        .on("end", resolve);
    })
  ])
```

Figure 44. image_lossy.js.



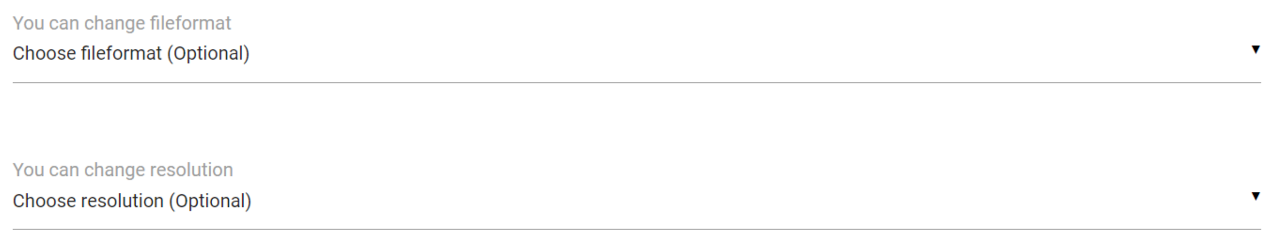
```
gulp.task("compress-images-lossless", () =>
  Promise.all([
    new Promise((resolve, reject) => {
      gulp
        .src("./uploads/*")
        .pipe(
          imagemin([
            imagemin.optipng({
              optimizationLevel: 5
            }),
            imagemin.jpegtran({
              progressive: true
            })
          ])
        )
        .on("error", reject)
        .pipe(gulp.dest("./compressed"))
        .on("end", resolve);
    })
```

Figure 45. image_lossless.js.

4.6 Video compression

Video compression task happens in the video-compress.js -file. Gulp has a support plugin called fluent-ffmpeg, but the plugin did not work well with the .mov -type files. For that purpose, another solution had to be found. The solution was to use FFmpeg together with the NodeJS child process. There are two child processes that can have multiple outcomes.

Video compression. In the figure 46 you can see that there are two option panels in the videopakkaus.html. The reason for this is that video compression has more options to choose from, than the image compression. Data travel between the client and the server happens in the same way as in image compression. The client-side sends chosen options as FormData to the server-side, where they are then processed.



You can change fileformat
Choose fileformat (Optional) ▾

You can change resolution
Choose resolution (Optional) ▾

Figure 46. Option panels.

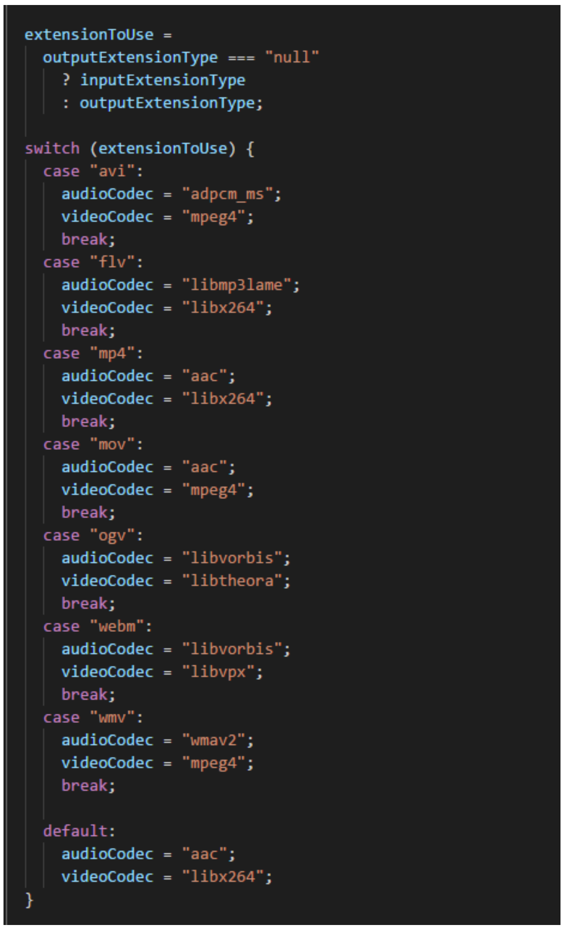
FFmpeg is based on terminal commands- In the figure 47 it can be seen what the conversion process command looks like in the terminal. The child process was used to automate the terminal commands in the video-compress.js -file.

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

Figure 47. FFmpeg in a command line tool (FFmpeg, [Ref: 6.7.2018].)

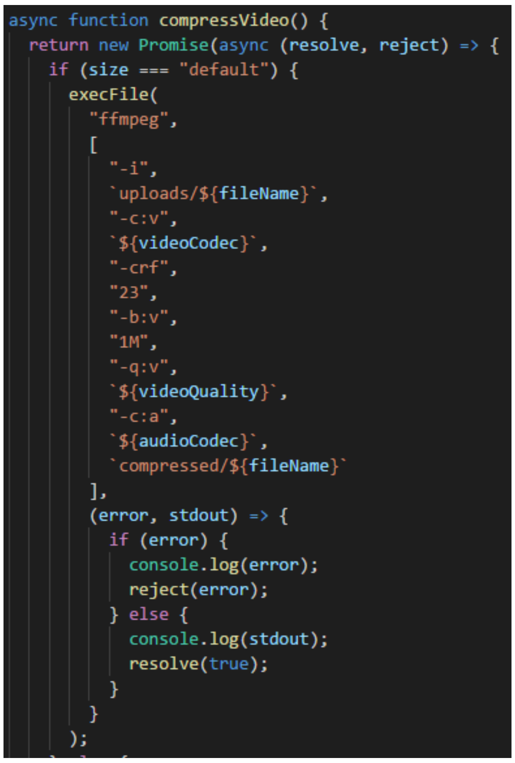
There are two different child processes for different file formats, because different file formats use different kind of codecs. Figure 48 shows file formats and their codecs.

The first process will be used when only compression is wanted to achieve (Figure 49). The second one is used when additional options are selected (Figure 50). These additional options include converting one format to another or changing video resolution. Both of these processes use compression in the process.



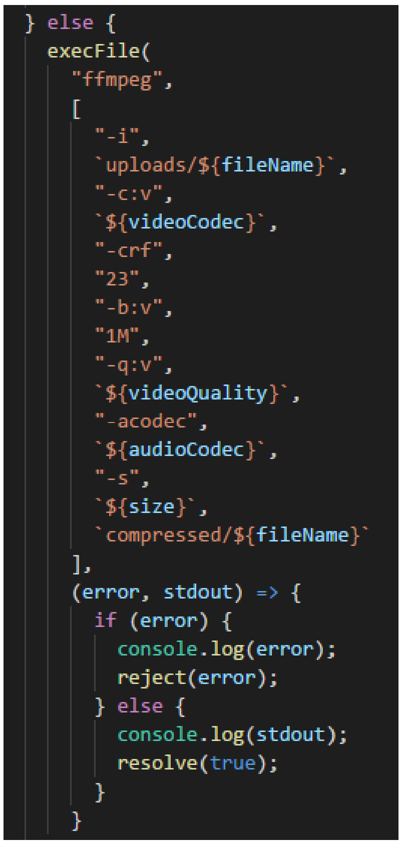
```
extensionToUse =
  outputExtensionType === "null"
    ? inputExtensionType
    : outputExtensionType;

switch (extensionToUse) {
  case "avi":
    audioCodec = "adpcm_ms";
    videoCodec = "mpeg4";
    break;
  case "flv":
    audioCodec = "libmp3lame";
    videoCodec = "libx264";
    break;
  case "mp4":
    audioCodec = "aac";
    videoCodec = "libx264";
    break;
  case "mov":
    audioCodec = "aac";
    videoCodec = "mpeg4";
    break;
  case "ogv":
    audioCodec = "libvorbis";
    videoCodec = "libtheora";
    break;
  case "webm":
    audioCodec = "libvorbis";
    videoCodec = "libvpx";
    break;
  case "wmv":
    audioCodec = "wmav2";
    videoCodec = "mpeg4";
    break;

  default:
    audioCodec = "aac";
    videoCodec = "libx264";
}
```

Figure 48. Codecs.



```
async function compressVideo() {
  return new Promise(async (resolve, reject) => {
    if (size === "default") {
      execFile(
        "ffmpeg",
        [
          "-i",
          `uploads/${fileName}`,
          "-c:v",
          `${videoCodec}`,
          "-crf",
          "23",
          "-b:v",
          "1M",
          "-q:v",
          `${videoQuality}`,
          "-c:a",
          `${audioCodec}`,
          `compressed/${fileName}`
        ],
        (error, stdout) => {
          if (error) {
            console.log(error);
            reject(error);
          } else {
            console.log(stdout);
            resolve(true);
          }
        }
      );
```

Figure 49. First child process.



```
} else {
  execFile(
    "ffmpeg",
    [
      "-i",
      `uploads/${fileName}`,
      "-c:v",
      `${videoCodec}`,
      "-crf",
      "23",
      "-b:v",
      "1M",
      "-q:v",
      `${videoQuality}`,
      "-acodec",
      `${audioCodec}`,
      "-s",
      `${size}`,
      `compressed/${fileName}`
    ],
    (error, stdout) => {
      if (error) {
        console.log(error);
        reject(error);
      } else {
        console.log(stdout);
        resolve(true);
      }
    }
```

Figure 50. Second child process.

File format change. This functionality is tied to browsers because HTML5 doesn’t support all existing video formats. When developing videos with macOS, video formats are created as .mov files and those are later changed to a different format suited for web. It is good to remember that different formats use different codecs. This means that the file sizes of these formats are different and changing a format will affect the file size as well. Changing the resolution of the original files can result in having smaller or even bigger file sizes. Figure 51 shows the file formats of the application that can be changed.

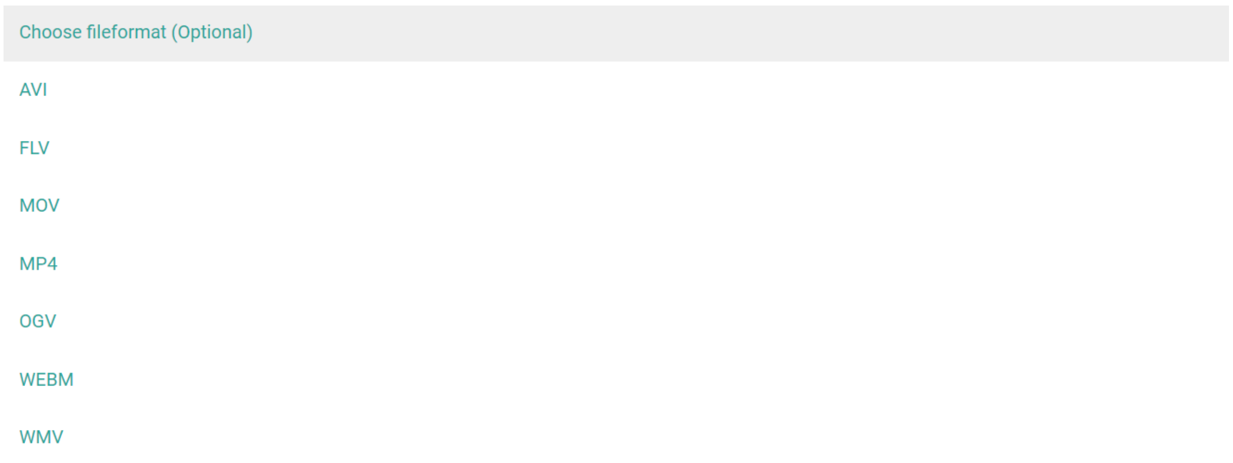


Figure 51. File format change.

Resolution changes improve compression. Changing the resolution of a video to a smaller one will result in smaller file sizes. However, it is not practical to upload a 4K resolution video to your website even with compression applied. Compressing the file size of a 4K video to the bare minimum would result in a poor quality video. Thus, it is better to downsize the resolution of a video from 4K to 080p or smaller. This way it is possible to achieve smaller file sizes and still maintain the good video quality suited for web.

Resolution change in the application. In the application there are two ways to change the resolution. First there are predefined options for the user to choose from. If the user doesn't want any specific resolution, it can be chosen from predefined options. Figure 52 shows the predefined option panel. The second way is to choose the option "other". It is an input where the user can insert the wanted resolution. Figure 53 shows the inputs of the second options. After the user has chosen the resolution, user can choose which aspect ratio to use. Figure 54 shows horizontal- and vertical aspect ratio options.

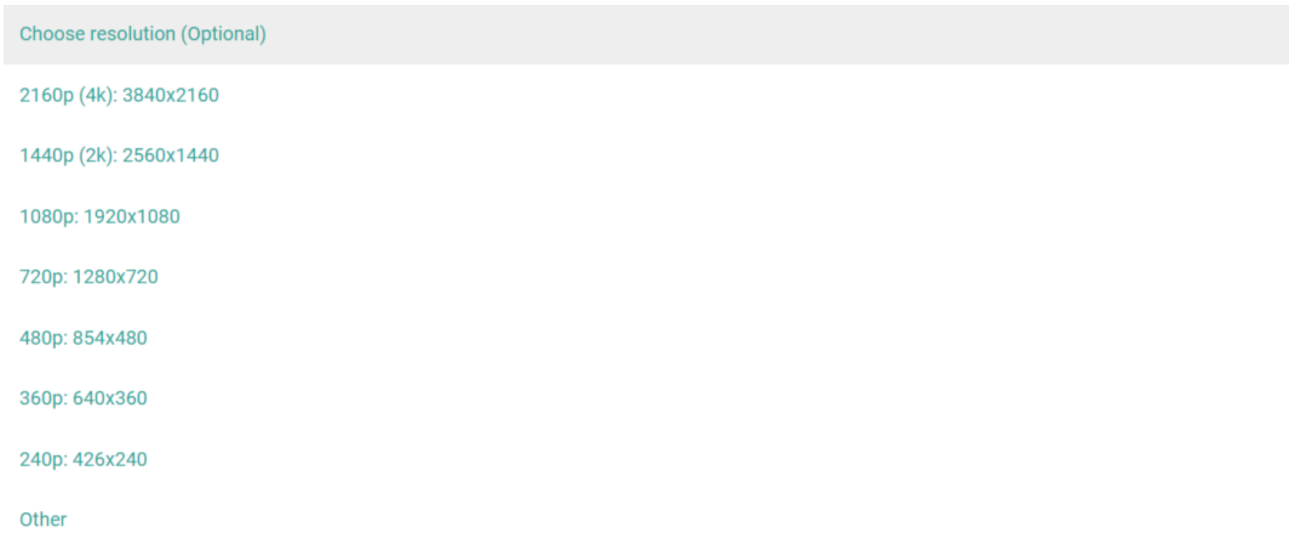

Choose resolution (Optional)
2160p (4k): 3840x2160
1440p (2k): 2560x1440
1080p: 1920x1080
720p: 1280x720
480p: 854x480
360p: 640x360
240p: 426x240
Other

Figure 52. Premade options.

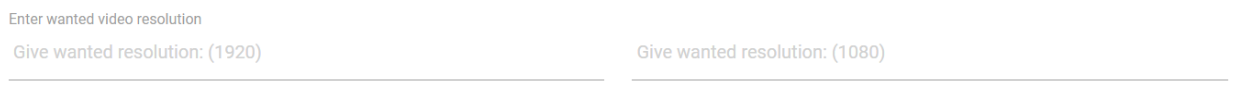

Enter wanted video resolution
Give wanted resolution: (1920)
Give wanted resolution: (1080)

Figure 53. User options.

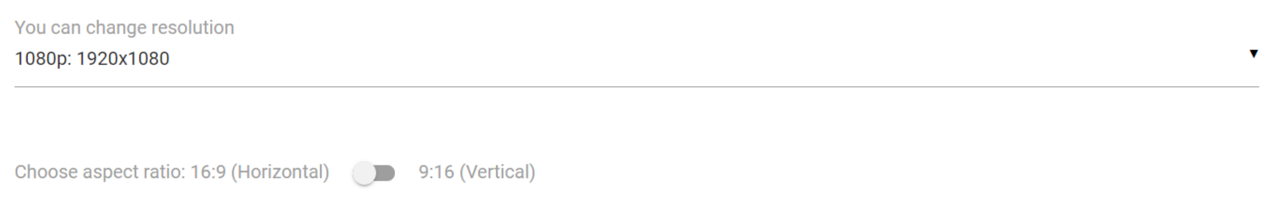
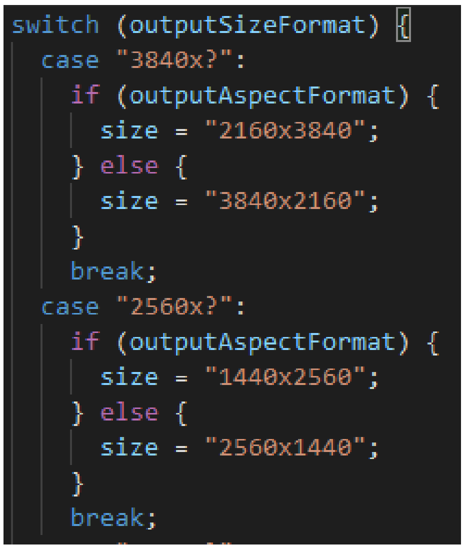

You can change resolution
1080p: 1920x1080
Choose aspect ratio: 16:9 (Horizontal)
9:16 (Vertical)

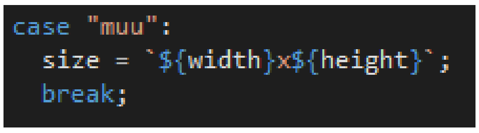
Figure 54. Aspect ratio.

Figure 55 shows the code behind “Choose resolution (Optional)” premade command inputs and aspect ratio. If the user will choose a premade option, the code will execute the given resolution in the compression process. Figure 56 shows what will be printed in the “other” options to the child process later to be executed. This option doesn’t need the aspect ratio slider, because the user will determine the resolution.



```
switch (outputSizeFormat) {
  case "3840x?":
    if (outputAspectFormat) {
      size = "2160x3840";
    } else {
      size = "3840x2160";
    }
    break;
  case "2560x?":
    if (outputAspectFormat) {
      size = "1440x2560";
    } else {
      size = "2560x1440";
    }
    break;
```

Figure 55. Premade option.



```
case "muu":
  size = `${width}x${height}`;
  break;
```

Figure 56. User options.

5 CONCLUSIONS

The main goal of this study was achieved. Image compression works and gives better compression ratios than the previously used application. Video compression functions as it is supposed to, and the conversion and resolution solutions have functioned without any problems. The program has been developed and tested on a separate server. At the moment, the application is used in production and it has been installed on the company's own server.

What could still be done in the future, is to test if the application can also be implemented on the company’s publishing system Webio. This would improve the compression ratios when the users are uploading images and videos to their own websites using the Webio – publishing system.

More features could have been implemented, like adding more features to the image- and video compression. Image compressor could have included a photo editor that could add text or filters in to the images. The video compressor could have been expanded to support more file formats and to have a video editor for video editing and compiling.

At the start of this project I only knew the basics of HTML and CSS. I did not have any former experience in JavaScript or web development. During the project I became aware of how to create full stack applications with mainly JavaScript. Many new programming principles and methods became familiar to me and I also learned how to develop an application that would be suitable for the needs of a company. Now I have more knowledge and confidence to continue my path as a full-stack web developer.

BIBLIOGRAPHY

Ansoft Inc. 2018. Tutorial Video Codec. [Web Page]. Ansoft Inc. [Ref: 24.9.2018]. Available at: https://www.any-video-converter.com/mac-tutorial/video-codec.php

Blelloch, G. 2013. Introduction to Data Compression. [Online publication]. Carnegie Mellon University. [Ref: 14.6.2018]. Available at: https://www.cs.cmu.edu/~guyb/realworld/compression.pdf

Codeacademy. 2018. What is REST?. [Online Article]. Codeacademy. [Ref: 19.6.2018]. Available at: https://www.codecademy.com/articles/what-is-rest

Comodo CA Limited. 2018. What is HTTPS?. [Online publication]. Imagify. [Ref: 19.7.2018]. Available at: https://www.instantssl.com/ssl-certificate-products/https.html

Dadfar, K. 2018. Understanding all the Different Image File Formats [Online Article]. Digital Photography School. [Ref: 21.9.2018]. Available at: https://digital-photography-school.com/understanding-all-the-different-image-file-formats/

Ecma International. 2018. 4 Overview. [Online publication]. Ecma International. [Ref: 15.6.2018]. Available at: https://tc39.github.io/ecma262/#sec-overview

Edwards, T. No date. Documentation, SweerAlert. [Web Page]. [Ref: 14.6.2018]. Available at: https://sweetalert.js.org/

Express. No date. Documentation, Hello world example. [Online publication]. [Ref: 30.10.2018]. Available at: https://expressjs.com/en/starter/hello-world.html

FFmpeg. No date. Documentation, About FFmpeg. [Web Page]. FFmpeg [Ref: 5.7.2018]. Available at: https://www.ffmpeg.org/about.html

FFmpeg. No date. Documentation, ffmpeg Documentation. [Documentation]. FFmpeg [Ref: 6.7.2018]. Available at: https://www.ffmpeg.org/ffmpeg.html

Official FFmpeg Wiki. 2018. Documentation, FFmpeg. [Wiki]. Official FFmpeg Wiki [Ref: 24.9.2018]. Available at: https://trac.ffmpeg.org/wiki

Giltsoff, J. No date. Documentation, SVG ON THE WEB - A Practical Guide. [Web Page]. [Ref: 21.9.2018]. Available at: https://svgontheweb.com/

Google Developers. 2018a. Enable Compression. [Online publication]. Google Developers. [Ref: 14.6.2018]. Available at: https://developers.google.com/speed/docs/insights/Enable-Compression

Google Developers. 2018b. Progressive Web Apps. [Online publication]. Google Developers. [Ref: 20.6.2018]. Available at: https://developers.google.com/web/progressive-web-apps/

GSP. 2014. Manual Reference Pages - JPEGTRAN (1). [Online publication]. GSP. [Ref: 20.6.2018]. Available at: http://gsp.com/cgi-bin/man.cgi?topic=jpegtran

Guillaume, C. 2015. Optimising SVG images [Online publication]. Moz://a HACKS. [Ref: 24.9.2018]. Available at: https://hacks.mozilla.org/2015/03/optimising-svg-images/

Gulpjs. No date b. Documentation, gulp Documentation. [Documentation]. Gulp. [Ref: 24.9.2018]. Available at: https://gulpjs.com/

Grigorik, I. 2018. Optimizing Encoding and Transfer Size of Text-Based Assets. [Online publication]. Google Developers. [Ref: 14.6.2018]. Available at: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#text-compression-with-gzip

Jessier, M. 2017. Lossy vs Lossless Image Compression. [Online publication]. Imagify. [Ref: 14.6.2018]. Available at: https://imagify.io/blog/lossless-vs-lossy-image-compression/

kornelski/giflossy. No date. GitHub source code repository: kornelski/giflossy. [Text document]. Giflossy. [Ref: 24.9.2018]. Available at: https://github.com/kornelski/giflossy

Material Design. 2018. Documentation, Material Design. [Web Page]. Google. [Ref: 18.6.2018]. Available at: https://material.io/design/introduction/#principles

Materialize. 2017. Documentation, Materialize. [Web Page]. Materialize. [Ref: 15.6.2018]. Available at: http://archives.materializecss.com/0.100.2/

MDN web docs. 2018a. HTML: HyperText Markup Language. [Online publication]. MDN web docs moz://a. [Ref: 15.6.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTML

MDN web docs. 2018b. Compression in HTTP. [Online publication]. MDN web docs moz://a. [Ref: 14.6.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Compression

MDN web docs. 2018c. JavaScript. [Online publication]. MDN web docs moz://a. [Ref: 15.6.2018]. Available at: https://developer.mozilla.org/bm/docs/Web/JavaScript

MDN web docs. 2018d. Hypertext Transfer Protocol (HTTP). [Online publication]. MDN web docs moz://a. [Ref: 18.6.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP

MDN web docs. 2018e. Express/Node introduction [Online publication]. MDN web docs moz://a. [Ref: 18.6.2018]. Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

MDN web docs. 2018f. Media formats for HTML audio and video [Online publication]. MDN web docs moz://a. [Ref: 24.9.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats

MDN web docs. 2018g. Web App Manifest [Online publication]. MDN web docs moz://a. [Ref: 20.6.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/Manifest

MDN web docs. 2018h. Getting Started [Online publication]. MDN web docs moz://a. [Ref: 30.10.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started

MDN web docs. 2018i. HTTP request methods [Online publication]. MDN web docs moz://a. [Ref: 31.10.2018]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

mozilla/mozjpeg. No date. GitHub source code repository: mozilla/mozjpeg. [Text document]. Mozilla. [Ref: 14.6.2018]. Available at: https://github.com/mozilla/mozjpeg

Nelson, M. & Gailly, J-L. 1995. The Data Compression Book. [E-Book]. Hoboken: Wiley. [Ref: 02.10.2018]. Available at : Google Books.

Node.js Foundation. 2018a. Documentation, Child Processes. [Web Page]. Node.js Foundation [Ref: 5.7.2018]. Available at: https://nodejs.org/api/child_process.html#child_process_child_process_exec_command_options_callback

Node.js Foundation. 2018b. Documentation, Overview of Blocking vs Non-Blocking. [Web Page]. Node.js Foundation [Ref: 18.6.2018]. Available at: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

Node.js v10.11.0 Documentation. No date, Child Process. [Text document]. Node.js v10.11.0 Documentation [Ref: 25.9.2018]. Available at: https://nodejs.org/api/child_pro-cess.html#child_process_child_process

npmjs.com. 2018. What is npm?. [Web Page]. npmjs.com. [Ref: 18.6.2018]. Available at: https://docs.npmjs.com/getting-started/what-is-npm

Patel P. 2018. What exactly is Node.js?. [Online publication]. FreeCodeCamp. [Ref: 18.6.2018]. Available at: https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5

Pngquant. No date. libimagequant - Image Quantization Library. [Text document]. pngquant. [Ref: 24.9.2018]. Available at: https://pngquant.org/lib/

Techopedia Inc. 2018. Scripting Language [Web Page]. Techopedia Inc. [Ref: 12.9.2018]. Available at: https://www.techopedia.com/definition/3873/scripting-language

Truţa, C. 2008. A guide to PNG optimization. [Text document]. Optipng [Ref: 24.9.2018]. Available at: http://optipng.sourceforge.net/pngtech/optipng.html

w3schools.com. 2018a. HTML introduction. [Online publication]. w3schools.com. [Ref: 15.6.2018]. Available at: https://www.w3schools.com/html/html_intro.asp

w3schools.com. 2018b. HTML <meta> tag. [Online publication]. w3schools.com. [Ref: 15.6.2018]. Available at: https://www.w3schools.com/Tags/tag_meta.asp