[1 Requirements 3](#_Toc485752189)

[1.1 Upload 3](#_Toc485752190)

[1.2 Administer Images 3](#_Toc485752191)

[1.3 Provide images for further processing 3](#_Toc485752192)

[2 Architecture 4](#_Toc485752193)

[3 Implementation 5](#_Toc485752194)

[3.1 Image Administration 5](#_Toc485752195)

[3.1.1 Filesystem 5](#_Toc485752196)

[3.1.2 Upload 5](#_Toc485752197)

[3.1.3 Download 6](#_Toc485752198)

[3.1.4 Deletion 6](#_Toc485752199)

[3.2 Further Processing 6](#_Toc485752200)

[3.3 Frontend 7](#_Toc485752201)

[3.4 Frameworks 7](#_Toc485752202)

[3.4.1 NodeJS and npm 7](#_Toc485752203)

[3.4.2 Electron 7](#_Toc485752204)

[3.4.3 Bootstrap 7](#_Toc485752205)

[3.5 Compression 7](#_Toc485752206)

[3.6 Instructions 7](#_Toc485752207)

Design and Implementation

Objective of this paper is to design and implement a solution for uploading medical images to a distributed filesystem so that they are available for further processing by another software component. The following chapters give an overview of existing requirements as well as architecture and implementation details.

# Requirements

The solution for uploading medical images is meant to be part of an online game which consists in segmenting such images and getting a score depending on precision. People who own such medical images are invited to share those images with the players by donating them to the gaming platform. Hence there are several requirements concerning this process of donating images which are described more closely in the following sections.

## Upload

* The donor finds an appealing user interface for uploading his images
* The donor finds the user interface to be self-explanatory
* The donor can upload images directly from his local filesystem
* The time, the donor must wait while is image is uploading, is limited to a minimum
* The upload should include compression of the image

## Administer Images

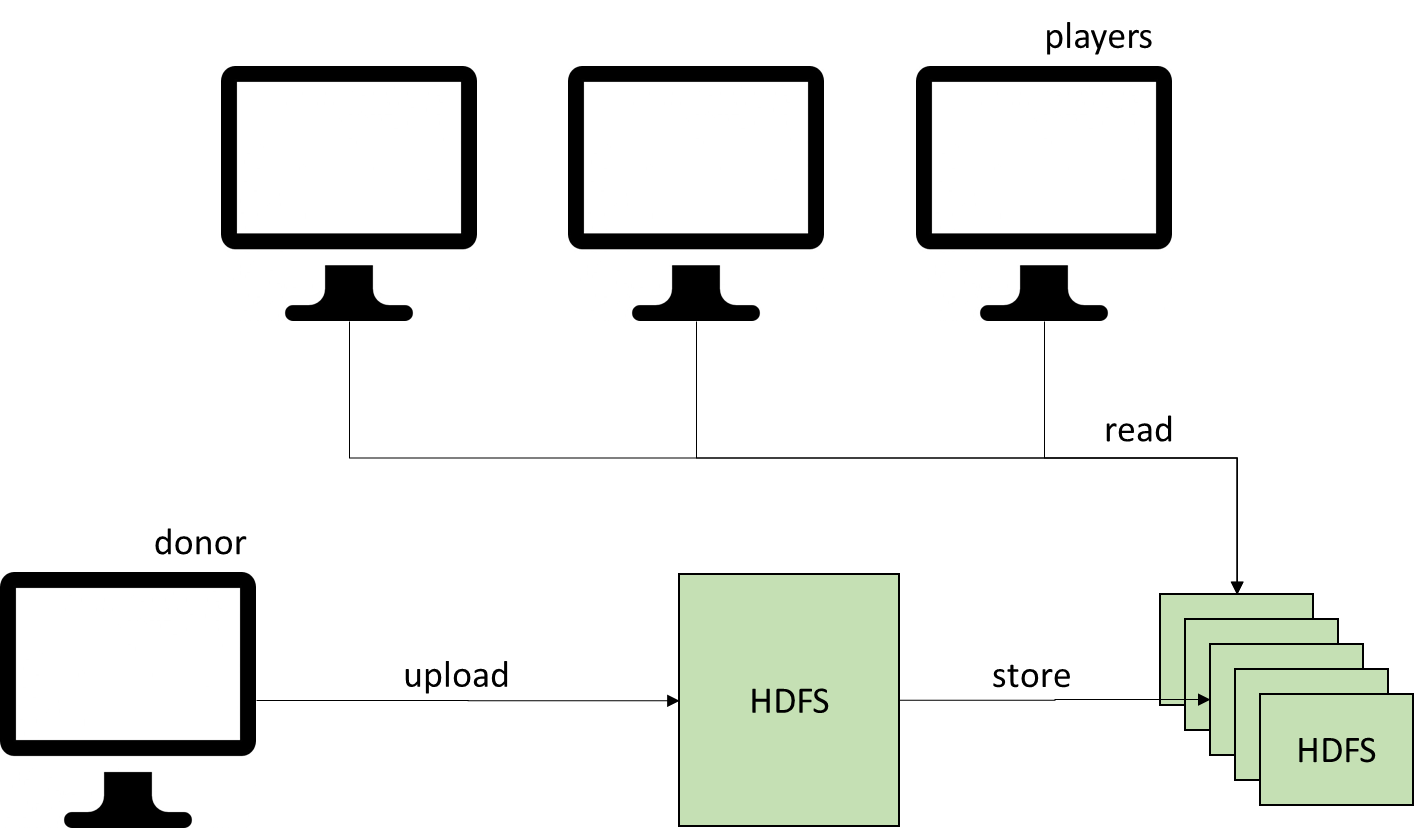
* The donor can view his uploaded images
* The donor can download his own images
* The donor can delete his own image

## Provide images for further processing

* The filesystem must provide the uploaded images for further processing by the players
* The images need to be accessible by a web interface
* The quality of the uploaded image must be maintained on a level so that it’s possible for the players to determine certain body parts or organs (ensuring that the donated images themselves fulfill this requirement is not part of this paper)

# Architecture

The idea is to have a lightweight frontend-solution which directly interacts with HDFS.



# Implementation

All software-modules are written in JavaScript and can easily be integrated into a web-frontend. This way, the requirement of the project owner to have a standalone solution can be fulfilled whilst ensuring future-proofness.

## Image Administration

### Filesystem

As the objective of this paper demands, the images need to be stored to a distributed filesystem. The filesystem only needs to provide the image for further processing, and isn’t the platform for the processing itself. Most likely, compared to the number of uploads per day, there will be much more active players. Therefore, the focus needs to be set on quick read operations. In addition, the stored images should be easily accessible via the web interface of the online game.

The Hadoop Distributed Filesystem is used in

A Hadoop WebHDFS REST API client library[[1]](#footnote-1) is used to implement the interaction with the HDFS. This library can be used in NodeJS and is available as npm-package. It provides basic functionalities like read and write operations, which are necessary for storing the donated images to the HDFS.

### Upload

To speed up the upload of donated images, the images are cut in tiles. This enables the parallelization of multiple processes which all compress and upload one single tile of the original image. To be able to dynamically determine the necessary number of tiles, there’s a fixed tile size.

The tile size itself can be configured to any suitable number of pixels. Yet, as some compression libraries have limitations in how large an original image can be, this aspect should be taken into consideration when configuring the tile size.

Not all images will have a multiple of the configured tile size as their number of pixels. Therefore, the processing is designed in a way so that there will be as much full tiles as possible, the rest of the image will be processed as rectangles. Uploaded to the HDFS, the tiles will be stored in a folder which has the name of the original image.

As part of the upload, a preview image will be generated and stored alongside the tiles. This image will be created by resizing it to a predefined size and lower quality. It is used in the administrative user-interface, to let the donor preview his uploaded images without the need of downloading the full-sized image.

### Download

The donor should have the possibility to download his uploaded images.

For the download, it’s necessary to recompose it from the tiles which are stored in the image folder. Therefore, it’s compulsory to know the original position of the tile in the image. To keep this information, the name of each tile includes the position expressed in pixels.

* Recomposition

### Deletion

As the donor still has the owner rights of his images, it is necessary that he has the possibility to delete them from the game. Therefore, it is necessary that the entire folder, including the tiles as well as the preview image, is removed from the HDFS.

## Further Processing

The players are meant to segment the image in a web interface. The determined segments will then be stored in another database. The segmentation itself, which is objective of another paper, will be realized with an extra layer which has its position over the image. The image therefore needs to be loaded from the HDFS.

This requires the same recomposing process as the download of an image, which is described in chapter xxx. The methods for recomposing the images are developed as a part of this paper and will be reused in the other paper which was mentioned to have the segmentation as its objective.

## Frontend

The frontend is meant to provide a well-structured entry point for the user. Therefore, all functionalities which are necessary for uploading or administering images need to be easy to find. The usage should be self-explanatory. Therefore, where it is possible, the frontend follows the design of a dialog in the local filesystem.

## Frameworks

### NodeJS and npm

### Electron

To develop a standalone application which can be turned into a web application without major effort Electron is used as framework. Electron is a framework for building cross platform applications with JavaScript. It is open source; the development and maintenance is carried out by GitHub.

<https://electron.atom.io/>

### Bootstrap

Bootstrap is a pervasive framework for developing responsive web interfaces.[[2]](#footnote-2) It covers HTML, CSS and JS and is used to design the user interface. Bootstrap offers a grid layout, which makes it easy to position elements. It also includes different designs for buttons and tables.

## Compression

As compression libary webp is used.[[3]](#footnote-3) It provides both, lossy and loss-free compression.[[4]](#footnote-4) It allows an image size of xxx pixels. The quality of the output image can be configured. Similar to the HDFS client library, the utilized converter is available as npm-package.

## Instructions

The instructions on how to run the developed application can be found in the readme file. Nevertheless, the following paragraph gives an overview of the pre-requirements which need to be fulfilled to run the application.

The application is designed to run against a pseudo distributed node of HDFS and would also work with a usual HDFS architecture. To have an instance of HDFS available during development, it was decided to use a Docker image. These are available online at no cost and provide an effortless way of testing against an actual instance of HDFS. The pseudo distributed node provides the possibility to work with multiple java processes, so it’s closer to the production environment than a usual single node of HDFS.

If the testing environment is set up with the same Docker image and the other Docker configurations which are included in the project, there’s also no need to change port-numbers or other information.

1. https://www.npmjs.com/package/webhdfs [↑](#footnote-ref-1)
2. http://getbootstrap.com/ [↑](#footnote-ref-2)
3. <https://developers.google.com/speed/webp/docs/webp_study> [↑](#footnote-ref-3)
4. https://developers.google.com/speed/webp/ [↑](#footnote-ref-4)