HTBLA Grieskirchen
4710 Grieskirchen, Parzer Schulstraße 1

Specialization Medical Information Technology

Academic year 2017/2018

DIPLOMA THESIS

Project

Layout Generator

Conducted by:

Andreas Resch, 5AHBGM-12

Felix Kirchweger, 5AHBGM-5

Supervisor:

Mag. Dr. Bernhard Mayr, MBA

Grieskirchen, 01.04.2018

Hand in annotation:
Date:

Supervisor:

Declaration

„I declare that I have developed and written the enclosed Diploma Thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked.”

______________________________ ______________________________

Grieskirchen, 01.04.2018 Andreas Resch

____

Felix Kirchweger

Acknowledgment

The success and outcome of this project required guidance and assistance from many people, and we would like to use this page to thank them.

First, we would like to thank the DCCS IT Business Solutions and especially Stephan Meißner for giving us the opportunity to work on this project and their continuous support during this project.

We are also grateful for the assistance of our advising teacher Mag. Dr. Bernhard Mayr, MBA, whose knowledge and experience in the fields of data science and machine learning were of utmost value to us.

Furthermore, Magdalena Meindlhumer and Judith Resch supported this project by proofreading our written thesis and providing feedback.

	HTBLA Grieskirchen
	Fachrichtung: Medizininformatik

DIPLOMARBEIT
DOKUMENTATION

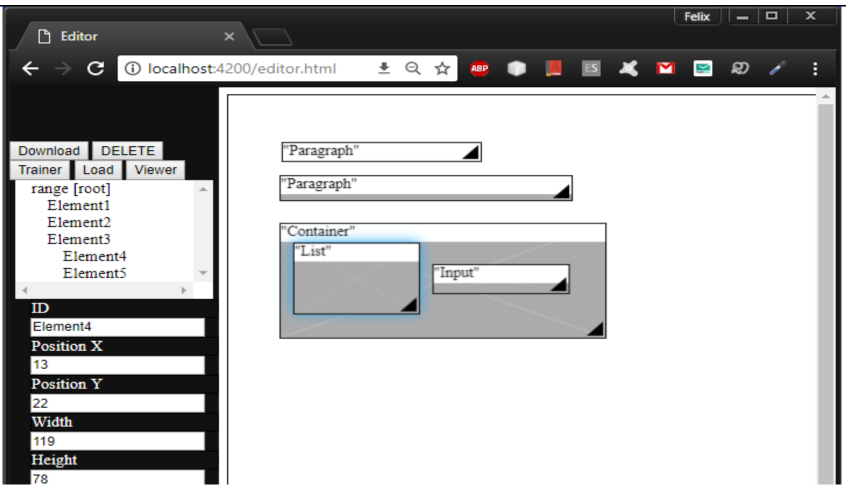
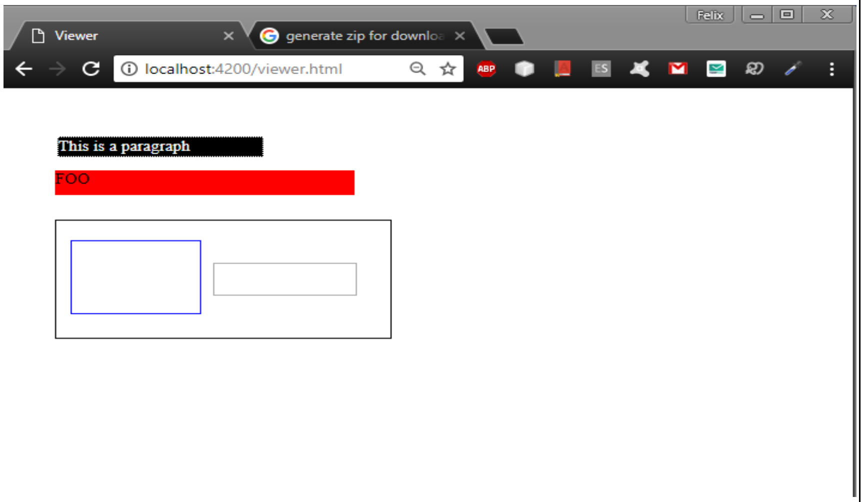
Namen der Verfasser/innen	Andreas Resch Felix Kirchweger
Jahrgang **Schuljahr**	5. Jahrgang 2017/2018
Thema der Diplomarbeit	Layout Generator
Kooperationspartner	DCCS IT Business Solutions

Aufgabenstellung	Die Aufgabenstellung beinhaltet die Benutzung von technisch anspruchsvollen Konzepten, namentlich Maschinelles Lernen mittels eines Neuronalen Netzwerkes sowie das Entwickeln eines grafischen Editors für HTML Layouts unter Zuhilfenahme eines MVVM-Patterns. Vorgaben seitens des Auftraggebers beinhalten die Verwendung folgender Libraries und Technologien: • NodeJS als Servertechnologie • Einen REST Service zur Kommunikation mit einer SQLite3 Datenbank • SocketIO als Socketverbindung des Websitehosts mit der Clientanwendung • ConvnetJS als Implementierung eines Neuronalen Netzwerks • KnockoutJS als Implementation des MVVM-Patterns im Editor

Realisierung	Das Neuronale Netzwerk soll benutzergezeichnete Symbole klassifizieren und dadurch einem Layout-Elementtyp zuordnen. Weiters soll das Netzwerk vom Benutzer trainiert werden können und dem Administrator Auskunft über die Genauigkeit des Systems geben können. Der zweite Teil nutzt die Fähigkeiten des Neuronalen Netzwerkes, um mit dessen Hilfe eine Layout-Generierung durch einen grafischen Editor zu ermöglichen.

Ergebnisse	Das Ergebnis besteht aus einem lauffähigen System aus drei Node.js-Applikationen und einer Webapplikation . Die NodeJS-Anwendung sind: • ein Rest-Service mit Datenbankanbindung • ein Websitehost • eine Trainings- und Test-Applikation für das Neuronale Netzwerk Die Webapplikation bietet: • eine Login-Seite • einen grafischen Editor zur Erstellung von HTML-Files und eine Downloadmöglichkeit des HTML-Files • einer Seite auf der das aktuelle HTML-File live angezeigt wird • einer Webpage zur Trainings- und Testdaten-Erstellung für das Neuronale Netzwerk

	HTBLA Grieskirchen
	Fachrichtung: Medizininformatik

Typische Grafik, Foto etc. (mit Erläuterung)	 Editor localhost:4200/editor.html Download DELETE Trainer Load Viewer range [root] Element1 Element2 Element3 Element4 Element5 ID Element4 Position X 13 Position Y 22 Width 119 Height 78 "Paragraph" "Paragraph" "Container" "List" "Input" Hauptfenster des Editors mit vier Element-Objekten  Viewer generate zip for downlo localhost:4200/viewer.html This is a paragraph FOO Darstellung der obigen Element-Objekte als DOM-Elemente in der Viewer.html
Teilnahme an Wettbewerben, Auszeichnungen	
Möglichkeiten der Einsichtnahme in die Arbeit	Die Diplomarbeit kann öffentlich eingesehen werden.

Approbation (Datum/Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin

	HTBLA Grieskirchen
	Department: Medical Information Technology

DIPLOMA THESIS
DOCUMENTATION

Author(s)	Andreas Resch Felix Kirchweger
Form **Academic year**	5th form 2017/2018
Topic	Layout Generator
Co-operation partners	DCCS IT Business Solutions

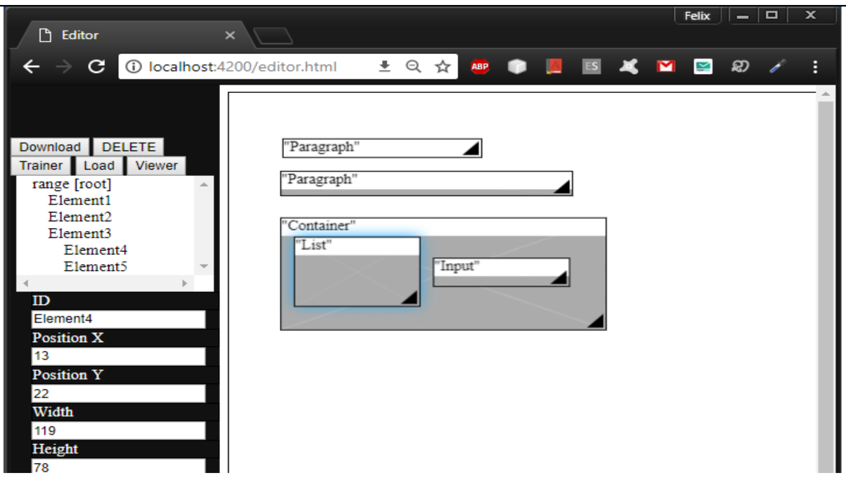
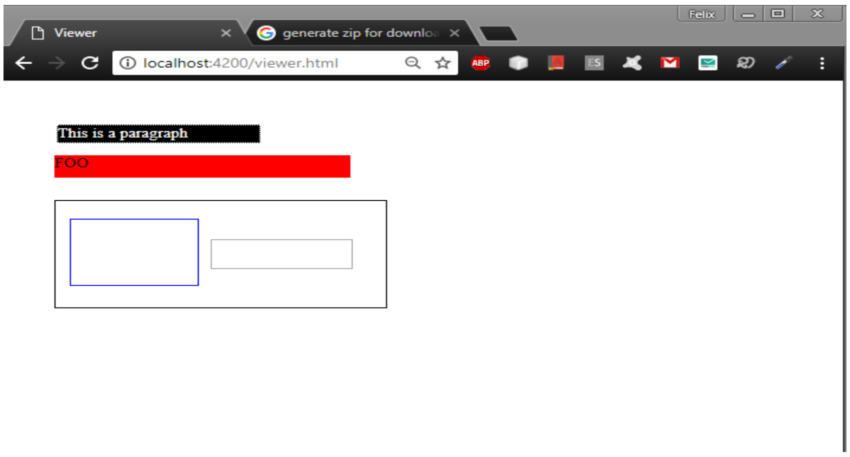
Assignment of tasks	The assignment requires the use of advanced technical concepts, those being machine learning using a neuronal network, as well as the development of a graphical HTML editor. The editors event handling and object management is to be achieved by the implementation of a MVVM pattern. Requirements of the contractor include: • NodeJS as server technology • A REST service to establish communication with an SQLite3 database • SocketIO to implement a socket-driven connection between the website host and the web application • ConvnetJS to create and train a neuronal network. • KnockoutJS to realize the MVVM pattern in the editor

Realisation	The neuronal network has to be able to classify user-drawn symbols and assign them to a specified layout element. Furthermore, the user must be given the ability to train this network. Test protocols have to be provided for the administrator to evaluate the prediction accuracy. The web application uses the networks abilities to facilitate the graphical editor's layout generating functions.

Results	The result is an executable system of three NodeJS applications and one web application. The Node.js applications are:

	• a REST service with connection to a database • a website host • a training and testing application for the neuronal network The web application offers: • a login page • a graphical editor for the creation of HTML files and download possi-bilities of those • a page which displays the generated HTML file during editing • a website to generate training and testing data for the neuronal net-work

	HTBLA Grieskirchen
	Department: Medical Information Technology

Illustrative graph, photo **(incl. explanation)**	 Editor localhost:4200/editor.html Download DELETE Trainer Load Viewer range [root] Element1 Element2 Element3 Element4 Element5 "Paragraph" "Paragraph" "Container" "List" "Input"ID: Element4 Position X: 13 Position Y: 22 Width: 119 Height: 78 Editor's main window with four element-objects  Viewer generate zip for downlo localhost:4200/viewer.html This is a paragraph FOO The four element-object from above represented as HTML-DOM-Elements in the viewer.html

Participation in competitions **Awards**	

Accessibility of **diploma thesis**	The diploma thesis can be viewed publicly.

Approval **(date/signature)**	Examiner	Head of College

Table of Content

1. Introduction

1.1. Involved Persons

1.1.1. Team

The team consists of following persons:

- Felix Josef Kirchweger
- Andreas Resch

All team members currently attend the 5AHBGM at the HTBLA Grieskirchen.

1.1.2. Supervisor

Mag. Dr. Bernhard Mayr, MBA functioned as our supervisor for this diploma thesis. His experience with Machine Learning and Data Science were of great support to us.

1.1.3. Contractor

Our contractor was the DCCS IT Services GmbH. We were in contact with Stephan Meißner (Software Development Manager), whose knowhow in the area of web-development assisted us a lot.

1.2. Conceptual Formulation

This project was of exploratory nature. The request made by our contractor was that we implemented a method of recognizing and categorizing user-drawn symbols and to use this data to generate a HTML layout file and also provide a web application to edit and style these layout files.

We used two main technologies:

1.2.1. Machine Learning

Machine learning allows programs to automatically learn and improve without being explicitly programmed.

1.2.2. MVVM

MVVM is a software architectural pattern used to manage data and its presentation and alteration.

The contractor advised us to use:

- NodeJS as server technology
- A REST service to establish communication with an SQLite3 database
- SocketIO to implement a socket-driven connection between the website host and the web application
- ConvnetJS to create and train a neuronal network.
- KnockoutJS to realize the MVVM pattern in the editor

1.3. Individual Task Assignment

1.3.1. Andreas Resch

Generating and training a convolutional neuronal network to recognize user-drawn symbols obtained from the client application and assigning them to a specified layout element. Providing a database and a method to communicate with the client and the training and testing application.

1.3.2. Felix Kirchweger

Developing a website which enables the user to draw symbols to generate test and training data and also provides the user with a graphical editor which relies on the neuronal networks data for generating HTML-layouts as well as providing the means of styling and downloading those layout-files.

1.4. Milestones

8/18/2017	**Sketch Pad finished**	Felix Kirchweger
9/1/2017	**Conv Network finished**	Andreas Resch
9/1/2017	**DB Server finished**	Andreas Resch
9/6/2017	**Editor finished**	Felix Kirchweger
9/6/2017	**Client finished**	Andreas Resch
9/7/2017	**Console app for Custom Training finished**	Andreas Resch
9/8/2017	**Merging the Editor and the Sketch Pad**	Team
1/7/2018	**Automated Testing of Neuronal Network**	Andreas Resch
2/28/2018	**Refitting data structure to support CSS strings**	Felix Kirchweger
3/16/2018	**Download of layout files**	Felix Kirchweger

3/23/2018	**Submitting diploma thesis to supervisor for correction**	Team
4/5/2018	**Submitting diploma thesis for grading**	Team

Table 1: Milestones

2. Machine Learning

"The capacity of a computer to learn from experience, i.e. to modify its processing on the basis of newly acquired information." https://en.oxforddictionaries.com/definition/machine_learning [5.11.2017]

Machine learning allows programs to automatically learn and improve without being explicitly programmed. Thanks to machine learning we have self-driving cars, speech and handwriting recognition, more effective web search, customer related advertising, improvement in understanding the human genome and much more. Machine learning uses statistical analysis to predict an output value based on input data. This can either be a value for example the price of a house or a percentage for instance if a specific e-mail is a spam e-mail or not. There are two ways to train a machine to learn algorithm, supervised and unsupervised learning, which will be described later. Cf. http://www.expertsystem.com/machine-learning-definition/ [5.11.2017], http://whatis.techtarget.com/definition/machine-learning [5.11.2017]

2.1. Neuronal Network

Neuronal networks are built out of layers of neurons.

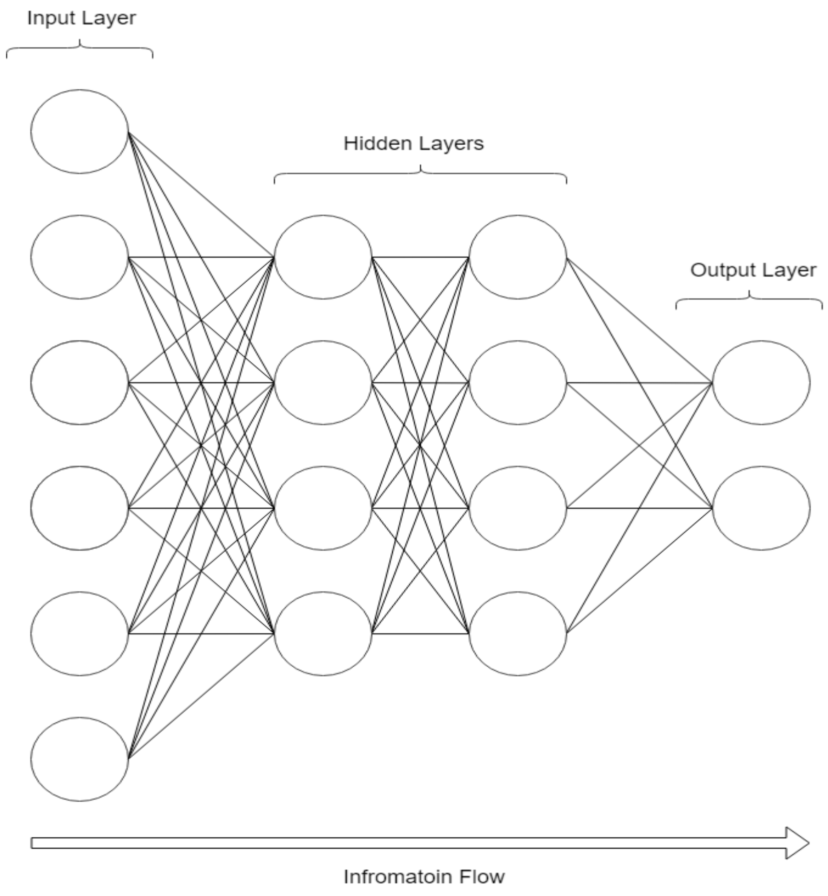


Figure 1: Visualization of a Deep Neuronal Network

2.1.1. Input Layer

Every neuronal network has exactly one input layer. Each neuron of an input layer receives one input, e.g. the size of a house in square meter, the number of bathrooms, the size of the garden and so on, to predict the price of a house. For each nonlinear depending parameter in the training data there is one neuron in the input layer. If you compare a Neuronal Network to a human body, the input layer would be our senses trough which we acquire information in and outside our body.

2.1.2. Output Layer

There is only one output layer in a neuronal network. The number of neurons in the output layer depends on the type of problem which the neuronal network should solve, e.g. if it is a regression problem then there is one neuron, if it is a classification problem there is one neuron for each class. In a human body the output layer would be the actions we take because of the information we got from our senses.

2.1.2.1. Difference of Regression and Classification

- "Classification is the task of predicting a discrete class label.
- Regression is the task of predicting a continuous quantity. […]
- A classification algorithm may predict a continuous value, but the continuous value is in the form of a probability for a class label.
- A regression algorithm may predict a discrete value, but the discrete value in the form of an integer quantity."

https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/ [24.2.2018]

For the prediction of something's price a regression algorithm would be used or to determine if a tumor is a benign one or a malignant one a classification method would find its use. In this diploma thesis multi-class classification is being used. The difference between a classification and a multi-class classification is that the multi-class classification has more possible classes than just two. An example for multi-class classification is to determine if there is an apple, an orange, a banana or a melon in a picture, or in our case if the symbol represents a button, an image, a link, or a different layout object.

2.1.3. Hidden Layers

The reason why these layers are called “hidden layers” is, that they receive data from a layer and give data to another layer. This means a hidden layer is neither visible as an input layer, nor as an output layer, therefore it is “hidden”. These hidden layers can be of a different type and have different activation functions. Their type can be fully connected, convolutional or pooling layers. Activation functions are functions which calculate the output of the layer. In comparison to the human body the hidden layers represent the brain. Cf. http://standoutpublishing.com/g/hidden-layer.html [12.3.2018]

2.2. Training the neuronal network

The learning rate defines how aggressive the learning algorithm changes the parameters of the neuronal network. If it is set too aggressive the network will not be able to produce good predictions. If the learning rate is set too passively it will take a lot of training iterations for the network to generate meaningful predictions.

The batch size defines the amount of training examples which propagate through the network before the parameters of the neuronal network are adapted.

2.2.1. Supervised Learning

The supervised learning uses labelled training data. These learning systems provide learning algorithms which slowly improve the parameters of the prediction algorithm to finally generate similar output values of the prediction algorithm as the output values from the labelled training data. Supervised learning is used in every speech automated system such as Siri and Cortana, OCR systems and in much more applications. Cf. http://whatis.techtarget.com/definition/supervised-learning [5.11.2017]

2.2.2. Unsupervised Learning

The unsupervised learning uses unlabelled training data. The task of the unsupervised learning algorithm is to find a structure in the data. This can be accomplished by a clustering algorithm. This type of algorithm tries to find similar sets of parameters and puts them into a cluster. The sets of parameters in a cluster must be similar to each other but dissimilar from the sets of parameters in other clusters. Cf. https://web.stanford.edu/class/cs345a/slides/12-clustering.pdf [8.3.2018]

2.3. The Used Neuronal Network

The JavaScript library convnetjs is used for the implementation of our Convolutional Neuronal Network. To create a new Network an array of layer definitions must be created. Each layer definition is a JSON-Object with multiple parameters. This array of definitions is then passed to the network trough the method makeLayers().

Our network is constructed after the example for prediction of handwritten numbers (the MNIST training data set) on the convnetjs homepage (https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html [18.3.2018]). We just added the first pooling layer to improve the performance by reducing the number of inputs and added a dropout probability (drop_prob) of 0.1 to the convolutional layers to prevent overfitting due to our small amount of training samples.

```
var types = Number.parseInt(row.count);
var net = new convnetjs.Net();
var Layer_Defs = [];
Layer_Defs.push({ type: 'input', out_sx: 100, out_sy: 100, out_depth: 1 });
Layer_Defs.push({ type: 'pool', sx: 4, stride: 4 });
Layer_Defs.push({
    type: 'conv', sx: 5, filters: 8, stride: 1, pad: 2,
    activation: 'relu', drop_prob: 0.1
});
Layer_Defs.push({ type: 'pool', sx: 2, stride: 2 });
Layer_Defs.push({
    type: 'conv', sx: 5, filters: 16, stride: 1, pad: 2,
    activation: 'relu', drop_prob: 0.1
});
Layer_Defs.push({ type: 'pool', sx: 3, stride: 3 });
Layer_Defs.push({ type: 'softmax', num_classes: types });
net.makeLayers(Layer_Defs);
```

Code-Snippet 1: Setput of used neuronal network

2.3.1. Input Layer

- "type": specifies the type of the layer
- "out_sx": number of inputs in x direction
- "out_sy": number of inputs in y direction
- "out_depth": number of inputs in z direction also called slices

The number of inputs is calculated trough *NumberOfInputs = out_sx * out_sy * out_depth*. In our case we have an input layer with 100*100*1 because the pictures consist of 100*100 binary values.

2.3.2. Hidden Layers

2.3.2.1. Pooling Layer

- "type": specifies the type of the layer
- "sx": number of affected Neurons in x and y direction for one calculation
- "stride": number of inputs to be passed after every calculation

Convnetjs utilizes max-pooling in a pooling layer.

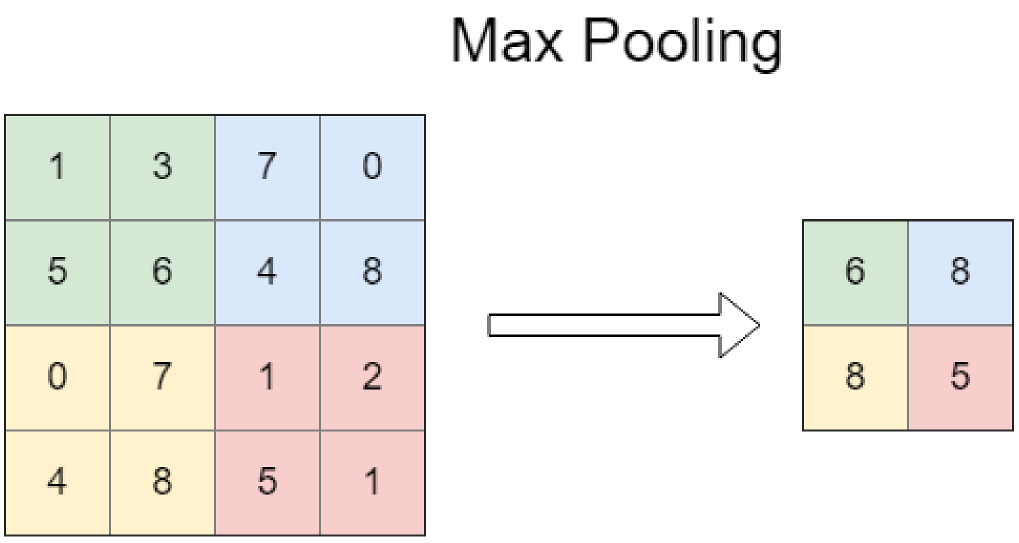


Figure 2: Max-Pooling example

The example shows a max-pooling with the parameters sx = 2 and stride = 2. If the stride and sx are the same, it means that the individual applications of max-pooling do not overlap themselves. Max-Pooling takes every pixel in the given range and returns its maximum.

> "… In general, they are used after multiple stages of other layers (i.e. convolutional and non-linearity layers) in order to reduce the computational requirements progressively through the network as well as minimizing the likelihood of overfitting. " https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network#Lay-ersofaConvolutionalNeuralNetwork-PoolingLayer [18.3.2018]

In our case the first pooling layer is used to reduce the input volume from 100x100x1 to 25x25x1 to improve the performance. The second and the third ones are to improve the performance and prevent overfitting, which is a big problem, because of our small amount of training data.

2.3.2.2. Convolutional Layer

- "type": specifies the type of the layer
- "sx": size of one filter
- "filters": number of filters
- "stride": number of inputs to be passed after one filter
- "pad": number of ceros inserted around the input
- "activation": type of function which calculates the output of the layer
- "drop_prob": The percentage of how much units are deactivated per propagation to prevent overfitting. The units which are deactivated are randomly chosen.

> "… The primary purpose of Convolution […] is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. …" https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [18.3.2018]

We decided to use ReLU as activation function, because it is the most common one used in convolutional neuronal networks and deep learning. Cf. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 [18.3.2018]

$ReLU$: $f(x) = \max(x, 0)$

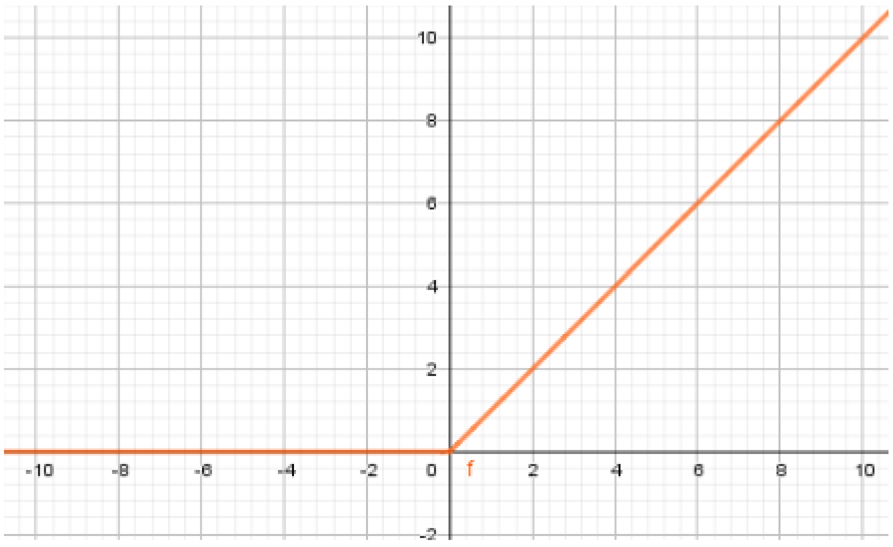


Figure 3: ReLU Function

If the input is positive the output is the same as the input. If the input is negative, the input is zero. A big problem of ReLU functions is that if the input is consistently zero, the backpropagating algorithm of the training multiplies zero with the error of the later layers, so no error signal gets ever passed to the previous layers. If this happens the neuron is considered as dead. To prevent this from happening a leaky ReLU function is used. It only returns zero if the input was zero. Otherwise a small decreasing negative number is returned. Cf. https://www.quora.com/What-are-the-advantages-of-using-Leaky-Rectified-Linear-Units-Leaky-ReLU-over-normal-ReLU-in-deep-learning?share=1 [2.4.2018]

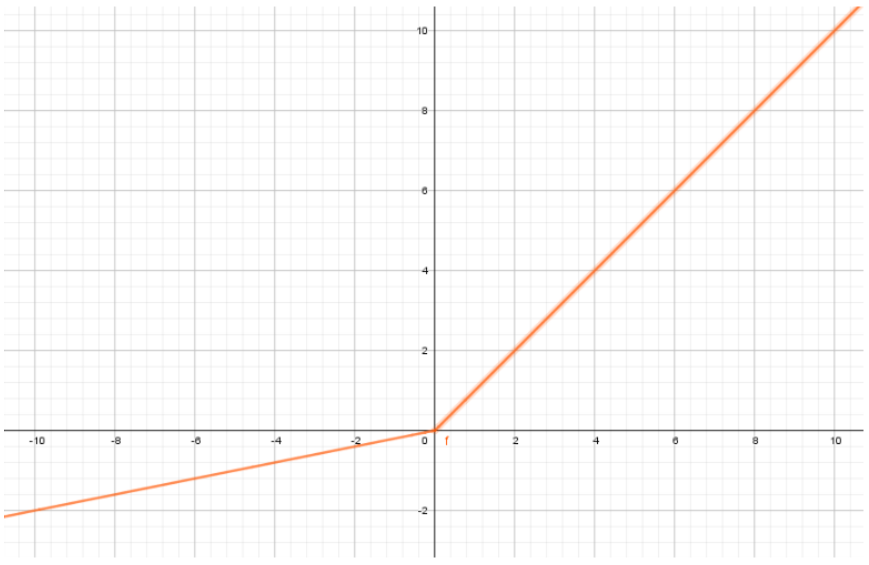


Figure 4: Leaky ReLU

2.3.3. Output Layer

- “type”: specifies the type of the layer
- “num_classes”: the number of classes we want to predict with our Network

We choose softmax as the type of our output layer. It returns a probability for each class which all sum up to one. When we create a network the parameter “num_classes” equals the number of types we have in our database.

2.3.4. Training

```
var trainer = new convnetjs.Trainer(
        net,
        { method: 'adadelta', batch_size: 20, l2_decay: 0.001 }
    );
```

Code-Snippet 2: Setput of used trainer

- “method”: different training methods implemented by convnetjs
- “batch_size”: How many examples the trainer passes through during training, before changing the weights. This helps the network to converge but also slows the process down. If the batch_size is set to one, after every training sample the weights would be changed. If the batch_size is set to 20 as in our case the trainer passes 20 trainings examples through before changing the weights.
- “l2_decay”: The higher this parameter goes, the better the network will be regularized, which is the counterpart to overfitting.

We used the trainings settings as suggested for beginners at the convnetjs Documentation.

> “Q: I don't care about anything, just tell me what to use. A: okay, the 2nd example above (with 'adadelta') is probably good to try. “ https://cs.stanford.edu/people/karpathy/convnetjs/docs.html [18.3.2018]

2.3.4.1. Training Algorithm

```
var trainer = new convnetjs.Trainer(
    net,
    { method: 'adadelta', batch_size: 20, l2_decay: 0.001 }
);
var loss = [];

//At maximum passes every picture 100 times through the net
for (let z = 0; z < 100 * pic.length; z++) {

    //Training magic
    var pic = pics[z % pics.length];
    var vol = generateVol(pic.data);
    var x = trainer.train(vol, parseInt(pic.type_id));

    //Checking the average loss after one iteration over all pictures
    if (z % pics.length == 0 && z != 0) {
        //If loss falls below a certain value the training is automatically stoped
        //to save time and prevent unnecessary training
        if (avg(loss))
            break;
        loss = [];
    }
    //saves the result of the training in the array
    //at the index of the trained picture in the pics array
    loss[z % pics.length] = x;
}
```

Code-Snippet 3: Training algorithm

At first, we initialize the trainer with the parameters we explained above. Then we define an array in which the results of the loss will be stored. At maximum every picture will be passed through the net at maximum 100 times. The method “generateVol()” converts the picture data into a “convnetjs.Vol” object, which is then used as the first parameter for the method “trainer.train()”. The second parameter is the number of the class of the picture. The method returns an object which contains the loss of the neuronal network. The following if-condition checks if every picture has been passed through the network once. If it is true the “loss” array is given to the “avg()” function, which checks if the average loss over all pictures is under a certain number and returns a Boolean, weather it is true or not. If it is true, the training is aborted to prevent unnecessary training. If not the “loss” array is emptied again.

Our dataset for the training contains 50 pictures for every of the 10 classes.

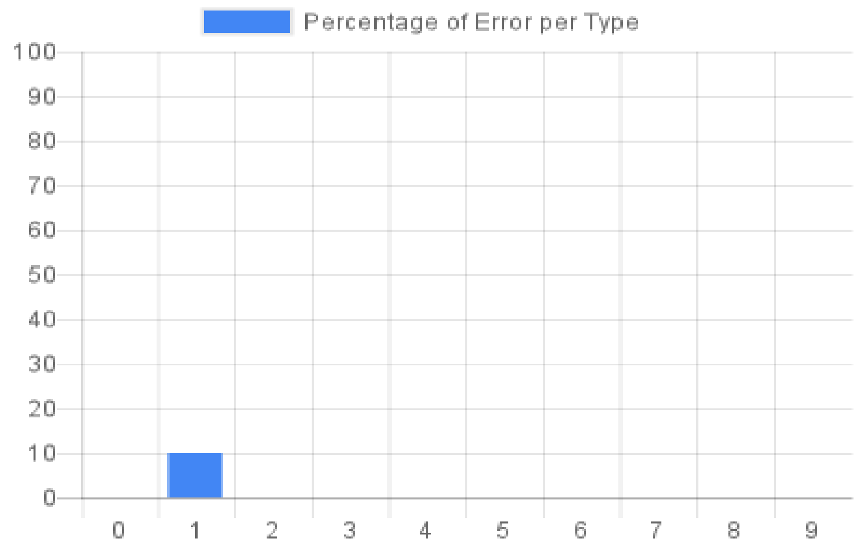
2.3.5. Testing

```
var errorcount = new Int32Array(types.length);
var picspertype = new Int32Array(types.length);

//forward all pics of test data
for (let i = 0; i < pics.length; i++) {
    let pic = pics[i];
    //converts the pic.data into a convnetjs.Vol object
    let vol = generateVol(pic.data);
    let result = net.forward(vol);
    let max = 0;
    let max_id = 0;

    //get the type with the highest percentage
    for (let x = 0; x < result.depth; x++) {
        var per = result.w[x];
        if (per > max) {
            max = per;
            max_id = x;
        }
    }

    picspertype[pic.type_id] += 1;
    //check if right type has been predicted
    if (max_id != pic.type_id)
        errorcount[pic.type_id] += 1;
}
```

Code-Snippet 4: Testing algorithm

To test a neuronal network, the training and testing application passes every dataset, which is marked for testing, once through the network and compares the predicted class with the attached id of the actual type of the dataset. Then the value in the array “picspertype” at the index of the id from the linked type is increased by one. If it is wrong the number in the array “errorcount” at the position of the id of the type is increased by one. These two arrays can be used to evaluate the prediction accuracy.

2.3.6. Prediction Accuracy



0:	Button
1:	Paragraph
2:	Container
3:	Input
4:	Image
5:	Selection
6:	Heading
7:	Link
8:	Table
9:	List

Figure 5: Prediction Accuracy

Our dataset for training contains 10 pictures for every of the 10 types.

The graph shows that only one of these pictures has not been recognized by our current neuronal network, which makes an overall prediction accuracy of 99%.

3. Backend

3.1. Node.js

"Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux. " https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm [21.3.2018]

It also has the largest ecosystem of open source libraries with its package system "npm". Cf. https://nodejs.org/en/ [21.3.2018]

Node.js runs under the MIT license.

3.1.1. Advantages

3.1.1.1. Asynchronous and Event Driven

Every API that is made for Node.js is asynchronous, which is essential for the concept of Node.js to work, because the server never waits for an API to return data, so it can handle the next request. It just gives the API a callback function (event) which is executed when the API is finished.

3.1.1.2. Performance

Node.js is built on Google Chrome’s V8 JavaScript Engine, which makes it very fast in code execution

3.1.1.3. Scalability and Single Thread

The Node.js Application makes use of a single threaded model with event looping. This results in the high scalability of Node.js applications in comparison to traditional server techniques which create a limited number of threads to handle requests.

3.1.1.4. Data output in chunks

The data never gets buffered in Node.js applications, it just gets returned in chunks.

Cf. https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm [21.3.2018]

3.1.2. Disadvantages

3.1.2.1. Uncontrollable quality of APIs

Everyone is able to write and publish APIs for Node.js or in general JavaScript. The absence of a uniform controlling system makes it hard to assess the quality and stability of APIs before trying them out by yourself.

3.1.2.2. Stringly-typed Programming

The reliance on strings instead of proper data structures is a general problem of JavaScript, which leads to a problem of Node.js. This often causes problems in combination with the use of APIs.

Cf. https://www.quora.com/What-are-the-disadvantages-of-using-Node-js [21.3.2018]

3.1.2.3. Node.js’ Rapidly Changing

Node.js’ API often changes in a backward-incompatible way. This lack of consistency leads to a lot of work for developers to keep their applications running with the latest Node.js API.

Cf. https://www.netguru.co/blog/pros-cons-use-node.js-backend [21.3.2018]

3.2. Network Communication

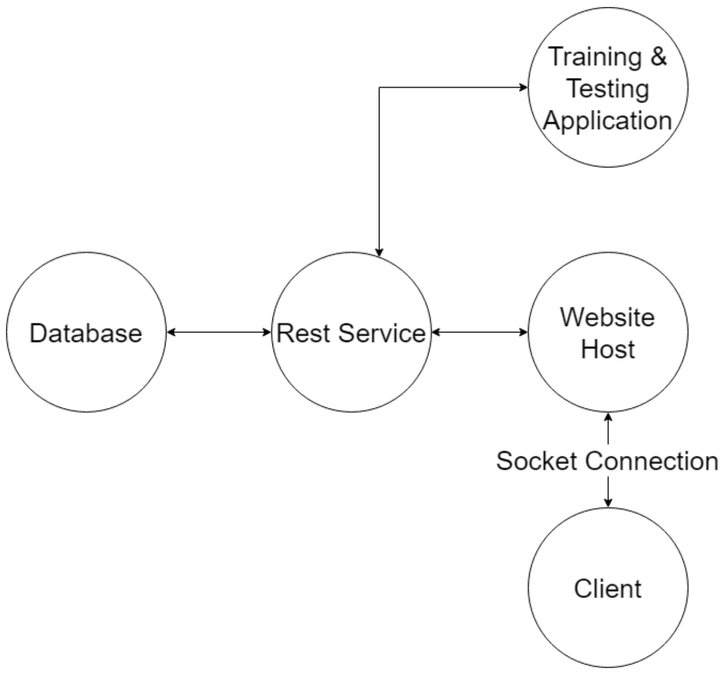


Figure 6: Network Communication

Briefly summarized the database stores all the necessary data. The communication with the database takes place via a REST service. The testing and training application receives and sends the data directly to the REST service, as does the website host. The communication with the individual websites is established through a socket connection.

3.3. Database

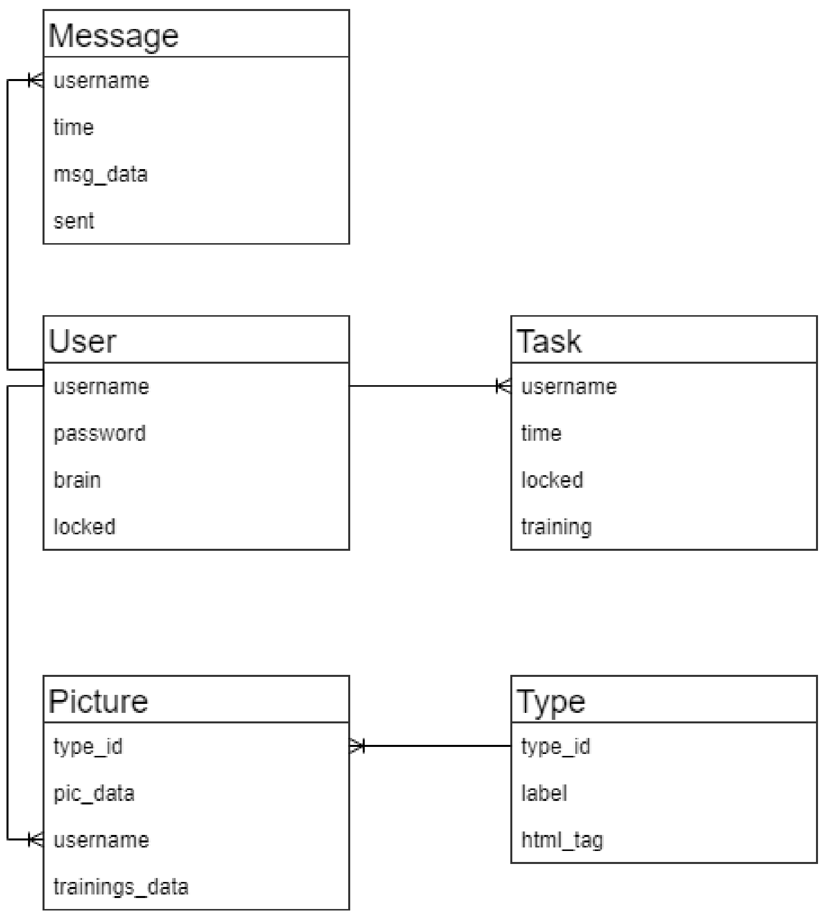


Figure 7: Database model

The table “Type” contains all supported types. Every type has a type_id, a label, which is the name of the type and a html_tag, which contains the HTML tag of the type, for example “p” for a paragraph.

In the table “User” every user is stored. Each user has a username, a password and his or her own neuronal network, which he or she can adept to his or her drawing style through custom training. When a new user is created, the default user’s network is taken over as his or her new network.

A user can create a task through the training page on the website which also will be stored in the database in the table “Task” until the task is finished by the training and testing application. A task can either be a training task to improve the users brain or a testing task to find out how accurate the neuronal network is able to predict the types of sketches. This is saved in the field “training” in the table Task.

During creating a task, a user is able to draw pictures, which are saved in the table “Picture”. If the pictures are drawn for a testing task, the field “training data” of the pictures is set to false, so these pictures will not be used for training. A type from the table “Type” must be attached to every drawn picture, because the training application uses supervised learning and the testing application needs the type to calculate the prediction error. The “type_id” in the picture must contain the same value as the “type_id” form the associated type. The length-encoded data of a picture is saved in the field “pic_data”.

The table “Message” stores messages, which are created in the training and testing application to notify a user if the training or testing has been finished. Every message consists of a “username”, which specifies the user whom message is for, the “time” when it has been saved, the “msg_data”, which holds the message itself and the attribute “sent”, which states if the messages has already been delivered to the user.

3.3.1. SQLite3

SQLite3 is an API which provides a light weight relational database management system in means of required resources, setup and database administration, which makes it perfect for our diploma thesis.

The SQLite3 API makes it possible to save queries as strings with “?” as placeholder which will be exchanged with a given value before execution. To execute statements there are three functions implemented by the SQLite API:

- Get()
- All()
- Run()

“Get()” and “All()” are used to read from the database (“Get()” if only one dataset is needed, “All()” if multiple dataset are needed). “Run()” is used to write on the database, for instance to insert, update or delete datasets from the database.

All three of these functions take three parameters:

1. a string with the statement
2. an array of values which are replacing the "?" in the statement
3. a callback function for when the API returns the data

Only the first parameter has to be set. The other ones can be left out if they are not needed, for example if there are no "?" to replace.

3.3.1.1. Example of a SQLite3 Database with Node.js

```
const sqlite3 = require('sqlite3').verbose();

//Connect to the database file
//If it does not exist it is created automatically
let db = new sqlite3.Database('../data/queue_application.db', (err) => {
    if (err)
        console.log(err);
    console.log('Connected to queue_application.db');
});

//Statements which create the tables in the database
//if they do not exit yet
const create_table_message = `CREATE TABLE IF NOT EXISTS message (
    username INTEGER NOT NULL,
    time             NOT NULL,
    msg_data TEXT    NOT NULL,
    sent     BOOLEAN DEFAULT FALSE
);`;

const create_table_user = `CREATE TABLE IF NOT EXISTS user (
    username TEXT    NOT NULL,
    password TEXT    NOT NULL,
    brain    TEXT,
    locked   BOOLEAN DEFAULT FALSE
);`;

//Executes the create statements.
//Serialize makes sure the statement is finished
//before the second one is started
db.serialize(function () {
    db.run(create_table_user);
    db.run(create_table_message);
});
```

Code-Snippet 5: SQLite3 database with node.js

At first a connection to the database file is established. If the file does not exist yet, it is automatically generated. To fill the database with tables the DDL of a table is saved as a string in a constant. Then these statements are executed, which leads to the creation of the tables if they have not existed yet.

```
//Define prepared statements

const get_messages_with_rowid = `SELECT * FROM message
                               WHERE rowid = ?;`;

const get_messages_for_user = `SELECT rowid,* FROM message
                               WHERE username = ? AND sent = 'FALSE'
                               ORDER BY time;`;

const new_message = `INSERT INTO message (username,time,msg_data)
                     VALUES(?,strftime('%s','now'),?);`;

const remove_sent_messages = `DELETE FROM message
                              WHERE sent = 'TRUE';`;

//Select the message with the rowid from the databse
db.get(get_message_with_rowid, [1], function (err, row) {
    //Do something with the data
});


//Select all messages of a user from the database
db.all(get_messages_for_user, [`default`], function (err, rows) {
    //Do something with the data
});

//Remove all sent messages
db.run(remove_sent_messages);

//Inserts a message into the database
db.run(new_message, [`default`, `You have won 100.000 $!!`]);
```

Code-Snippet 6: Defining and executing DDL on sqlite3 database with node.js

At first the needed statements are saved in constants.

Then the function “get()” is used to acquire the message with the “rowid” one from the database. If there is no message with the “rowid” one, the variable “row” would be “undefined”.

Next the function “all()” is executed to get all messages of the user “default”. If there are no messages for the user “default”, the variable “rows.length” would be zero.

Afterwards all messages that have been sent are deleted from the database with the function “run()”.

In the last line of code, a new message for the user “default” is inserted into the database with the message data “You have won 100.000$!!”.

3.4. Rest Service

The only way to communicate with the database is the REST service. For the implementation we used the libraries Http and Express.

3.4.1. Definition of a Rest Service

Representational State Transfer (REST) is a software architectural model for distributed networks, defined by Roy Fielding in his PhD dissertation “Architectural Styles and the Design of Network-based Architectures” at UC Irvine.

In an RESTful context, every object and/or datafile is called a resource and is identified using its “Uniform Resource Identifier” (URI). The URI describes the used protocols (such as: http, mailto, ftp), the used service and a valid web address. Using the HTTP-commands PUT, GET, POST and DELETE these resources can be described, altered, and deleted.

Cf. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [20.3.2018]

3.4.2. Example of a Rest Service in Node.js

```
var express = require('express');
var app = express();
var server = require('http').createServer(app);

//All the requests

server.listen(4400);
```

Code-Snippet 7: Setpu of a node.js server

Through these four lines of code a Node.js server is set-up which can be reached by the IP address and the port 4400. If the IP address of the server on which the rest service is running is 10.10.0.1 the rest service would be accessible under “10.10.0.1:4400”. In case the other applications are running on the same hardware the server can be reached through “localhost:4400”.

3.4.2.1. Post Request Server Side

```
app.post('/newmessage', function (req, res) {
    //Gets the request-header "Content-Length"
    var length = req.header('Content-Length');
    var reqdata = '';
    //Add a function which is called when data arrives
    req.on('data', (temp) => {
        reqdata += temp;
        //Checks if the whole data has arrived
        if (length == reqdata.length) {
            //Parses the received JSON-Object
            var obj = JSON.parse(reqdata);
            //Inserts it into the database
            db.run(new_message, [obj.username, obj.msg_data]);
            //Sends response to client
            res.status(200).send("Message saved");
        }
    });
});
```

Code-Snippet 8: Post request server side

In the code above a typical post-request is shown. The first argument of the app.post() method is the path for the request. The second parameter is a function, which is called asynchronously when a request for the before defined address is made. This function has two parameters:

- “req” for the request that has been sent.
- “res” for response of the request which will be sent back to the one who called the request.

3.4.2.2. Get Request Server Side

```
app.get('/gettypename/:type_id', function (req, res) {
    //Sends a request for the name of the type with the given id and adds a callback function
    db.get(get_type_name, [req.params.type_id], function (err, row) {
        //Sends response with type name to client
        res.status(200).send(JSON.stringify(row.label));
    });
});
```

Code-Snippet 9: Get request server side

The code shown atop displays a common get-request. The parameters of the method app.get() are the same as the ones from the method app.post(). The first parameter in this case is the string “/gettypename/:type_id”. When this request is called the “:type_id” is replaced with an actual username. This username can be accessed via req.params.type_id.

3.4.2.3. Get Request Client Side

```
http.get('http://' + dbserver + ':' + dbserverport + '/gettypename/' + id, (res) => {
    //Gets the request-header "Content-Length"
    var length = res.headers["content-length"];
    var resdata = '';
    //Sets a function which is called when data arrives
    res.on('data', (temp) => {
        resdata += temp;
        //Checks if the whole data has arrived
        if (length == resdata.length) {
            //Check if request was successful
            if (res.statusCode == 200) {
                //
                client.emit('type', JSON.stringify({ label: resdata, type_id: id }));
            }
        }
    });
});
```

Code-Snippet 10: Get request client side

To send get-requests we use the method http.get(). The first input is a string which represents the URI under which the request can be reached. The second parameter is a function which is called when the response of the server is received. In this method we check if the whole data has already arrived. That is done by comparing the length of the received data with the value of the request header “content-length”. If the sent data from the server is short there is no need to do that, because it arrives in one bit, but when there is a big amount of data sent from the server it arrives in small bits.

3.4.2.4. Post Request Client Side

```
//Defines the hostname and port under which the server is listening
const dbserver = 'localhost';
const dbserverport = 4400;

//Request definition
var req = http.request({
    hostname: dbserver,
    port: dbserverport,
    path: '/newmessage',
    method: 'POST',
    headers: {
        'content-type': 'application/json',
        'content-length': message.length
    }
//Function which is called on response of the server
}, (res) => {
    console.log('Message has been sent to server');
    });
//Sends the request
req.write(message);
req.end();
```

Code-Snippet 11: Post request client side

To send a post request we define a variable “req” with the “http.request()” method. This method gets a JSON-Object which has the settings for the request, such as hostname, port, path, method and so on and a method which is called when the response from the server is received.

The request is sent with the method “write()” which receives the data that should be sent to the server.

3.5. Sockets

The communication with the clients is happening via sockets. These sockets are implemented with help of the API socket.io.

3.5.1. Definition of Sockets

There are three types of sockets:

- stream sockets
- datagram sockets
- raw sockets

In our applications we use stream sockets which are supported by the API socket.io. For those sockets the two communication parties must establish a connection first. This type of sockets guarantees that any data which is sent over the network arrives at its destination.

The difference between a socket connection and a rest service is that a rest service is only able to answer to a request of a client. With the implementation of sockets, the server is able to send data independently to the client, ones the client is connected.

Cf. https://www.lifewire.com/socket-programming-for-computer-networking-4056385 [20.3.2018]

3.5.2. Example of a Socket Connection in Node.js

The concept of sockets will be explained on an easy example.

3.5.2.1. Server-Side Socket Connection

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

//when the url of the server is entered the index.html page is sent back
app.get('/', function (req, res) {
    res.sendFile(__dirname + '/index.html');
});

//sets a function that is called when a client connects
io.sockets.on('connection', function (client) {

    //sets a function that is executed when a client emits a login call
    client.on('login', function (data) {

        console.log(data);

        //Sends an answer back to the client
        client.emit('loggedin', 'Login successful');
    });

});

//defines the port on which the server is listening
http.listen(3000, function () {
    console.log('listening on *:3000');
});
```

Code-Snippet 12: Server-Side Socket Connection

The code above shows a node.js server which listens on the port 3000.

At first a server is initialized and with that server a socket is created. Then a get-request is defined, which is called when the server address is entered in a browser. The request responds with a html page (shown in Code-Snippet 13: Client-Side Socket Connection) which is then displayed in the browser. When a client connects to the server through a socket, the set callback function for the connection event is executed. In this function one can define custom events and add callback functions to them as we did with the event "login". It is also possible to send data to the connected client, through the "emit()" function. The first parameter of the emit function, is the name of the event under which the client expects the answer, the second one is the data that should be sent to the client.

3.5.2.2. Client-Side Socket Connection

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>Socket Example</title>
    <script type="text/javascript" src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
    <script type="text/javascript" src="https://code.jquery.com/jquery-1.11.1.js"></script>
    <script>
        $(function () {

            //creates a socket connectoin to the server example
            var socket = io.connect('http://127.0.0.1:3000');

            //sets a function that is called when the server sends a 'loggedin' call
            socket.on('loggedin', (data) => {

                console.log(data);

                window.alert('You have logged in!! :)');
            });

            //Sets a funtion that is executed when the login button is clicked
            $('#login').on('click', () => {
                //sends a login call to the server
                socket.emit('login', 'Smith');
            });
        });
    </script>
</head>
<body>
    <!-- Login Button -->
    <button id="login">Login</button>
</body>
</html>
```

Code-Snippet 13: Client-Side Socket Connection

The code shown above displays the file which is sent to the browser, in our example of a server-side socket connection (3.5.2.1Server-Side Socket Connection). In this example we use the JQuery and socket.io APIs. At first in the JavaScript section a socket is initialized which connects to the given internet address and port. Through this, the connection callback function on the server is executed. The next part shows how to define an event on which the client listens to the server. In our case the event is named “loggedin”. The callback function for this event writes the received data to the console and opens an alert window in the browser which shows the message: “You have logged in!! :)”. In the last JavaScript part displays a function which is called when the Login Button is clicked. In this function the socket emits the “login” event with the username as data to the server.

3.6. Testing and Training Application

The training and testing application checks every 5000ms if a task is available. It then downloads the task from the database through the rest service and sends a request to lock acquired task and user, to ensure that no other training and testing application works on the same task or user. It then executes the task and when it is finished it sends a request to remove the task from the database and unlock the user of the task. It also sends a message for the user of the task to the rest service which states, that his or her task has been processed. If it was a testing task a testing protocol is generated and saved locally.

3.6.1. Testing Protocoll

Every time a testing task has been finished, the application generates a testing protocol. The filename is generated after following schema:

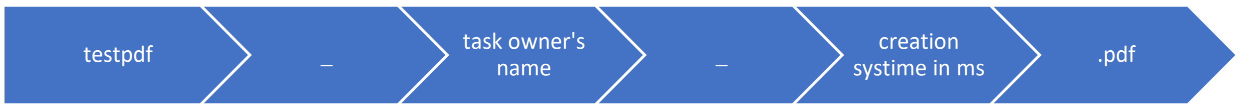


Figure 8: Filename of testing protocoll

An exemplary filename would be: “testpdf_myusername_1521297305320.pdf”.

A testing protocol contains information about:

- the network owner’s username
- the date the test has been done
- the time the test has been started and finished
- the number of datasets tested per type
- the prediction error per task in percent

3.6.1.1. Chartjs-node

Chartjs-node is a JavaScript library for the creation of charts. It supports different types of charts such as line graphs, pie and bar charts. With its help the training and testing application creates a chart, which shows the prediction accuracy (shown in the Figure 5) and a chart visualizing the number of test datasets per type.

3.6.1.2. PDFKit

The JavaScript API pdfkit makes it possible to create pdf files in an uncomplicated way. It supports the use of different text-styling options as well as including pictures into a pdf file. In our diploma thesis it is used for the creation of test protoclolls.

4. Frontend

4.1. Used Technologies and Concepts

4.1.1. jQuery

> "jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. […]" *https://jquery.com/* [22.03.2018]

According to w3techs.com jQuery is used on at least 73.3% of all webpages and has a market share of 96,2%, making it the most used JavaScript library. It focuses on three core aspects:

4.1.1.1. DOM Traversal and Manipulation

Get the first input element with the class 'draggable' and change its value to 'foo bar'.

```
$('input.editable')[0].value = 'foo bar';
```

Code-Snippet 14: DOM Traversalt using jQuery

4.1.1.2. Event Handling

Show an alert reading 'button clicked' whenever any DOM-Element possessing the id 'btn' is clicked.

```
$('#btn').on('click', function () {
   alert('button clicked');
});
```

Code-Snippet 15: Assigning a click handler using jQuery

4.1.1.3. Ajax

Call the local file ‘demo_test.txt’ and on success set the result as html of ‘#div1’.

```
$.ajax({
    url: "demo_test.txt", success: function (result) {
        $("#div1").html(result);
    }
});
```

Code-Snippet 16: A jQuery ajax call

This feature is only included for completion. in this project, a stream sockets are used. For further information on Sockets see subheading 3.5 Sockets

4.1.2. MVVM

Model–View–ViewModel (**MVVM**) is a software architectural pattern

> *"MVVM facilitates a separation of development of the graphical user interface – be it via a mark-up language or GUI code – from development of the business logic or back-end logic (the data model). The view model of MVVM is a value converter, meaning the view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented. In this respect, the view model is more model than view, and handles most if not all of the view's display logic. The view model may implement a mediator pattern, organizing access to the back-end logic around the set of use cases supported by the view. "* https://en.wikipedia.org/wiki/Model–view–view-model [07.11.2017]

4.1.2.1. Components of MVVM

The MVVM pattern features three key components, which are as follows:

- **Model**
  - Data representing objects and procedures of the business domain (e.g. data on a server).
- **ViewModel**
  - Representation of the data and operations on a User Interface.
  - Abstract model to simplify access and avoid confusion.
  - Performs event handling
- **View**
  - User Interface representing the state of the ViewModel. Displays the ViewModels data.
  - Sends commands (e.g. click events) to the ViewModel.

Thanks to the separation between the 'view' and the 'model' anyone of both can be changed without the other needing to.
The linking of ViewModel and view is handled by a binder, a special use of the observer pattern, which allows communication in both directions.

4.1.2.2. Advantages and disadvantages of MVVM

- Advantages
    - The view can be altered or replaced without having to change the model
    - Reduced Glue Code between model and view
    - It enables the abstraction of the model's data, thusly making it easier to present it to the user
    - Different views can be bound to one ViewModel without the need of adaption.
- Disadvantages
    - In large files, the binders tend to create a lot of data. It can be sufficient to replace bindings with static methods

Cf. https://blogs.msdn.microsoft.com/johngossman/2006/03/04/advantages-and-disadvantages-of-m-v-vm/ [07.11.2017]

4.1.3. KnockoutJS

> “Knockout is a JavaScript library that helps you to create rich, responsive display and editor user interfaces with a clean underlying data model. Any time you have sections of UI that update dynamically (e.g., changing depending on the user’s actions or when an external data source changes), KO can help you implement it more simply and maintainably. “ http://knockoutjs.com/documentation/introduction.html [22.03.2018]

Knockout implements the aforementioned MVVM pattern. The model is often a database server from which data is gathered through Ajax calls. The view is a HTML-Page and the viewmodel can be any JavaScript object. For example:

```
var ExampleViewModel = {
    ExampleText='foo bar';
    ExampleNumber=100;
}
```

Code-Snippet 17: A simple ViewModel

A View of this model accessing the ExampleText value could be:

```
<p>This is: <span data-bind="text:ExampleText"></span></p>
```

Code-Snippet 18: Binding a model value to a DOM elements attribute

Knockout then needs to be activated by executing the following line in any DOM-ready handler such as jQuery’s $ function.

```
ko.applyBindings(ExampleViewModel);
```

Code-Snippet 19: Activating KnockoutJS

Now our view will appear as if we had written:

```
<p>This is: foo bar</p>
```

Code-Snippet 20: Result of the above code

4.1.3.1. Observables

Observables enable Knockout to update a view automatically when the underlying model changes. Such observables are essentially functions acting as JavaScript objects which notify subscribers on changes. Altering the ViewModel above to contain observable would result in this:

```
var ExampleViewModel = {
    ExampleText=ko.observable('foo bar');
    ExampleNumber=ko.observable(100);
}
```

Code-Snippet 21: A simple ViewModel containing observables

Now all changes made to ExampleText would immediately show in the view.

4.1.3.2. Observable Arrays

An observable array is used to respond to changes of a collection of things. An observable array only notifies subscribers when an element is added or removed from the collection, but to react to changes of a property of an object inside the collection it has to be made an observable.

Observable arrays of objects with observable properties can be used to easily add, remove or update DOM-Elements during runtime.

4.1.3.3. Computed Observables

Computed Observables are functions dependent on at least one observable and will update whenever these dependencies change. An example for a computed observable is shown below:

```
function PatientViewModel() {
    var self = this;
    self.firstName = ko.observable('John');
    self.lastName = ko.observable('Doe');
    self.ssn = ko.observable(1234567890);
    self.fullName = ko.computed(function () {
        return "Patient no.  " + self.ssn + ": " + self.firstName() + " " +
self.lastName();
    });
}
```

Code-Snippet 22: Example of a computed observable

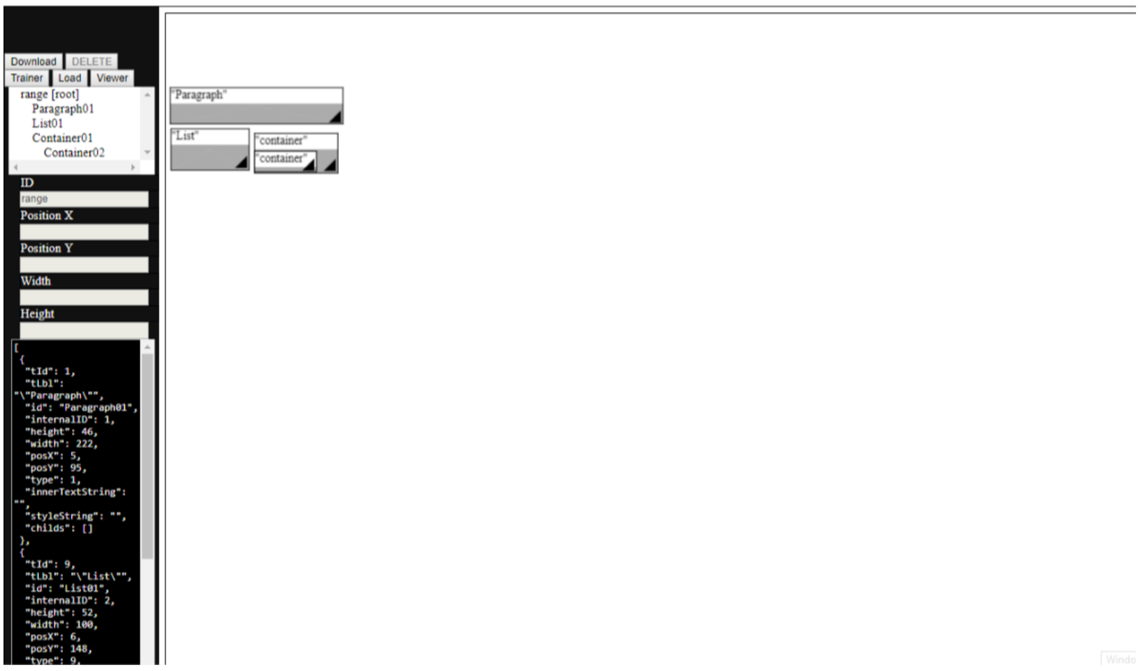
4.1.3.4. Bindings

Bindings are used to connect the View to the ViewModel's data and functions. Bindings can be used to generate a specific DOM pattern for each element in a collection. While it is possible to hardcode such patterns, formulating and using a template is more readable. Following text is responsible for rendering the tree display on the left-hand sidebar of the editor webpage:

```
<div id="treeModel">
    <ul id="rootList" class="childList">
        <li>
            <a data-bind="event:{click:setRange}">range [root]</a>
            <ul class="childList"
              data-bind="template:{name:'treeTemplate',foreach:elementArray()}"></ul>
        </li>
    </ul>
</div>

<!--Template-->

<script type="text/html" id="treeTemplate">
    <li>
        <a data-bind="class:id(),text:id(),event:{click:updateCO}"></a>
        <ul class="childList"
              data-bind="template:{name:'treeTemplate',foreach:childNodes()}"></ul>
    </li>
</script>
```

Code-Snippet 23: HTML and template used for rendering the editors tree representation

This code sample utilizes multiple bindings. The template binding is specified with the name parameter, the foreach parameter specifies the render method. In this case, the template is rendered for every element contained inside the observable elementArray. The template itself is contained by a script tag, whose id tag is synonym for its name. Rendering the template for multiple elements will result in something similar to this:



```
range [root]
  Paragraph01
  List01
  Container01
    Container02
```

ID: range
Position X
Position Y
Width
Height

```
[
 {
  "tId": 1,
  "tLbl":
"\"Paragraph\"",
  "id": "Paragraph01",
  "internalID": 1,
  "height": 46,
  "width": 222,
  "posX": 5,
  "posY": 95,
  "type": 1,
  "innerTextString":
"",
  "styleString": "",
  "childs": []
 },
 {
  "tId": 9,
  "tLbl": "\"List\"",
  "id": "List01",
  "internalID": 2,
  "height": 52,
  "width": 100,
  "posX": 6,
  "posY": 148,
  "type": 9,
```

Figure 9: Demonstrating recursive template rendering using the editor's treeTemplate

The event binding is used to add event handler for a specified event. When such an event is triggered, the specified JavaScript function (in above template the updateCO function) will receive the current model value and the DOM event object. Clicking on either representation of the ViewModels data (either the list element titled “Paragraph01”or the draggable titled “Paragraph”) will give the same current model value but different DOM event objects.

4.1.3.5. Creating a custom binding

KnockoutJS offers the possibility to define custom bindings. A binding is created by adding it as a subpropertiy of ko.bindingHandlers:

```
ko.bindingHandlers.yourBindingName = {
   init: function (element, valueAccessor, allBindings, viewModel, bindingContext){},
   update: function (element, valueAccessor, allBindings, viewModel, bindingContext){}
};
```

Code-Snippet 24: Registering a binding

Every binding features can feature either one or both of the following callback functions:

- init: This will be called when the binding is first applied to an element. Used to initialize variables and set up event handlers.
- update: Will be called once after every init to track dependencies (ko observables or computed observables) and after that every time any of those dependencies change

The passed parameters are:

- element: the involved DOM element
- valueAccessor: used to get the involved model property.

To implement the dragging and resizing abilities of the editors draggable elements a custom binding was created:

```
ko.bindingHandlers.dragResize = {
    init: function (element, valueAccessor, allBindings, viewModel, bindingContext) {
        var val = ko.unwrap(valueAccessor());
        //checking if inserted at root or as child of an element
        checkParent(element, val);
        //making element a draggable
        createDragElement(element, val);
        //making the handle draggable
        createHandle(element, val);
    },
    update: function (element, valueAccessor, allBindings, viewModel, bindingContext){
        var updateCoords = function () {
            //constraints on x,y,h,w of child.
            checkConstraints(val);
            //setting values after constraint checked
            TweenLite.set(element, { x: val.x(), y: val.y(), width: val.w(), height:
            val.h() });
            //aligning handle after constraints enforced
            TweenLite.set($(element).children(".resize-handle"), { top: val.h(), left:
            val.w() });
            //updating JSON in the sidebar's textarea
            updateJSON();
            //putting JSON into localStorage for the Viewer.js
            localStorage.setItem('data', JSONField.value);
        };
        var val = ko.unwrap(valueAccessor());
        // make the updateCoors function call whenever one of its observables
        // changes
        updateCoords();
    }
};
```

Code-Snippet 25: The custom dragResize binding

The init callback establishes on which level of the layout tree the element was inserted and creates DOM elements and event handlers for the drag and resize capabilities.

The update callback is used to confirm the validity of positioning and dimensioning, so that no child can be bigger than its parent or located outside of its parent. It also ensures that the handle remains at its bottom right position and calls functions to update the JSON representation of this layout as well as putting the JSON-string into the localStorage for the Viewer application to get and display. The TweenLite functions are explained in subheading 4.1.4 GreenSock GSAP

4.1.4. GSAP

> “Simply put, GSAP is the most robust high-performance animation library on the planet […]. Unlike monolithic frameworks that dictate how you structure your apps, GSAP is completely flexible; sprinkle it wherever you want.[…]"GSAP" describes all of the animation-related tools which include TweenLite, TweenMax, TimelineLite, TimelineMax, various plugins, extra easing functions, etc. ” https://github.com/greensock/GreenSock-JS/blob/master/README.md [23.03.2018]

The use of the GreenSock Animation Platform was restricted to the TweenLite animation tool for tweening some css properties and mainly as a dependency of the Draggable plugin. Therefore this description will not be very extensive

4.1.4.1. TweenLite and the CSSPlugin

TweenLite offers an easy way to animate various properties. With TweenLite only DOM properties can be animated, the CSSPlugin allows for additionally tweening CSS properties. Both are requirements for the GSAP Draggable plugin.

TweenLite offers many types of tweens, the most common is a TweenLite.to(), the following code piece is responsible for fading in a box-shadow property marking the currently selected element.

```
idString = self.chosenObject().internalID();

TweenLite.to($("#" + idString),0.5,{boxShadow:"rgb(35, 173, 255) 1px 0px 21px 0px"});
```

Code-Snippet 26: Example of the TweenLite.to() function

The first parameter is a DOM element or a collection of such elements who are to be animated. The second parameter specifies the duration in seconds and the third parameter is an object of tweakable attributes

In the case of simply setting the properties, a TweenLite.set() tween can be used. This acts the same way as setting the duration of a TweenLite.to() to zero. The following code piece realigns the resize handle with the bottom left corner of its parent DOM element after resizing due to size or position constraint violation

```
//aligning handle after constraints enforced
TweenLite.set($(element).children(".resize-handle"), { top: val.h(), left: val.w() });
```

Code-Snippet 27: Example of the TweenLite.set() function

4.1.4.2. GSAP Draggable Plugin

The GSAP Draggable plugin was chosen because it offers an easy way to define boundaries which the draggables are confined to and because of its rich callback system needed for counteracting adverse effects resulting from the need of nesting Draggables.

An element is made draggable by invoking Draggable.create() and supplying either a selector string or a DOM element reference. Following code sample contains the creation of a resize handle element, whose callback functions cause the parent element to resize when the handle is dragged. This function is called during the init callback of an dragResize binding

```
function createHandle(element, val) {
    //creating and positioning DOM element for the resize handle
    var handle = $("<div class='resize-handle'
style='bottom:0;right:0;'></div>").appendTo(element);
    TweenLite.set(handle, { top: val.h(), left: val.w() });
    drag = Draggable.create(handle, {
        type: "top,left", bounds: val.parent(),
        onPress: function (e) {
            e.stopPropagation(); // preventing event bubbling
        },
        onDrag: function (e) {
            val.w(Math.round(this.x));
            val.h(Math.round(this.y));
            TweenLite.set(element, { width: val.w(), height: val.h() });
        }
    });
}
```

Code-Snippet 28: function used to create a resize handle using GSAP Draggable

Draggable.create() is given the handle DOM reference and an object of options. The type option specifies the allowed drag direction. Either ‘left’ or ‘top’ by itself would the element allow to only be dragged horizontally or vertically. The callback onPress stops the event from bubbling upwards the DOM tree, triggering onPress events on every occasion. The onDrag callback sets the handles top and left position as its parent’s height and width.

The bounds option constrains the draggable to the given container. However this plugin was not intended to be used with nested draggables, so for every drag action, all parent draggables have to have their draggable status revoked and reassigned after the drag operation is completed.

4.1.5. Logical-Layout-Tree

This projects frontend data structure is DOM oriented, with every element object representing a node in a tree structure.

The object trees uppermost level is the EditorViewModels observable elementArray which contains all element objects, who do not have a real parent. Instead these elements self.parent attribute references the EditorViewModels empty rangePlaceholder element. This placeholder element functions as a pseudo root element.

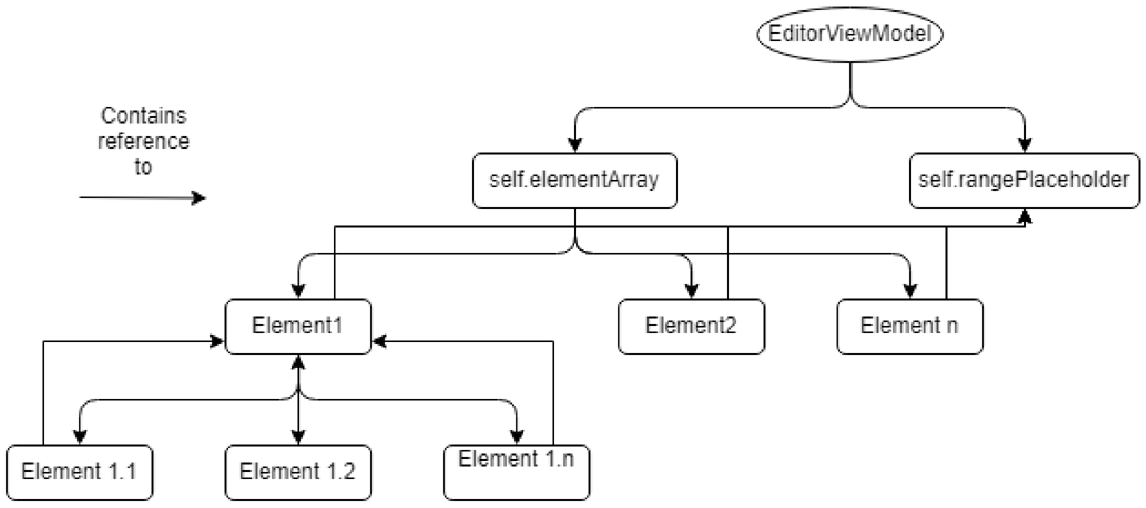


Figure 10: Tree diagram of the viewmodel's data structure

4.2. Pages

4.2.1. Authorization

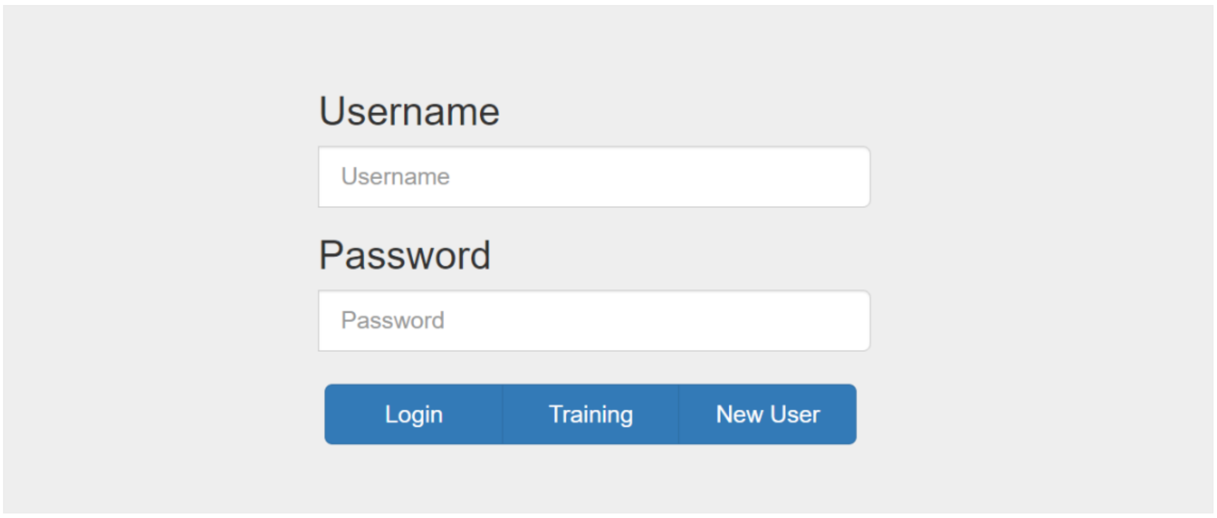
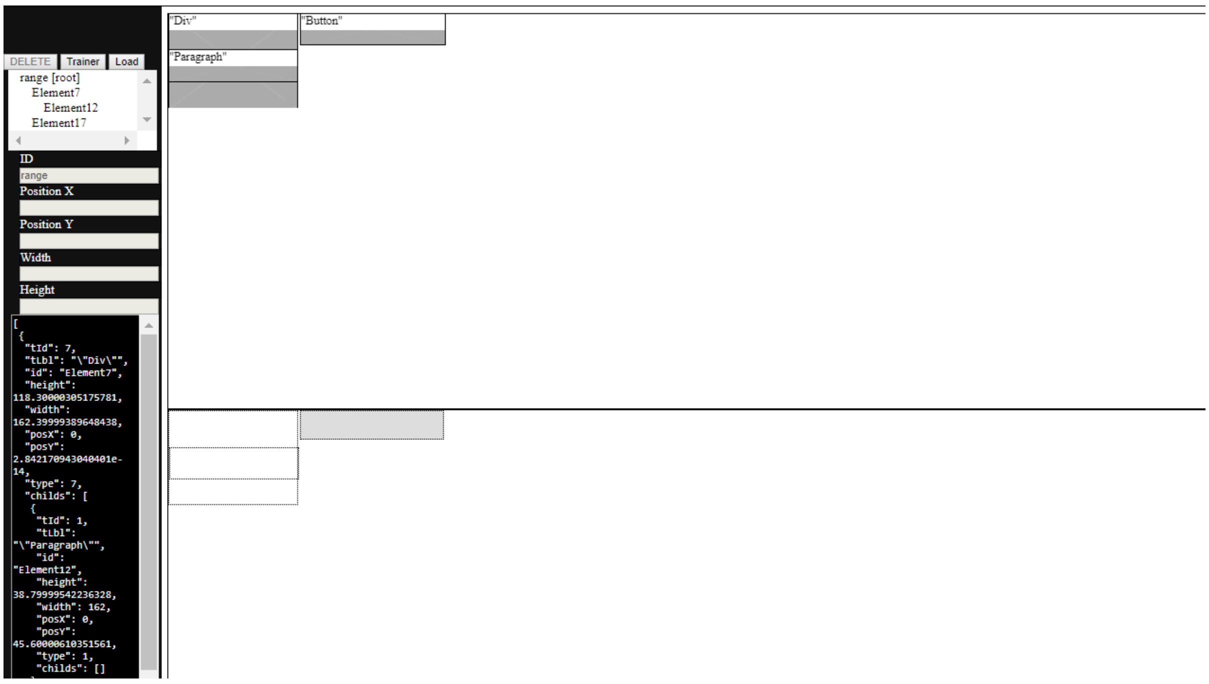


Figure 11: login.html

This is the first Screen the user encounters when connecting. It features a text input for entering the username and a password input for the password. Clicking on the buttons titled “login” or “training” triggers a socket emit, leading to the node server validating the login data. On successful validation the user is redirected to the pages “Editor.html” and “Training.html” respectively. Clicking the button labelled “New User” triggers a socket emit which causes the server to create a new user if the chosen username is not yet used.

4.2.2. Using the Editor



```
[
 {
  "tId": 7,
  "tLbl": "\"Div\"",
  "id": "Element7",
  "height":
118.30000305175781,
  "width":
162.39999389648438,
  "posX": 0,
  "posY":
2.842170943040401e-
14,
  "type": 7,
  "childs": [
   {
    "tId": 1,
    "tLbl":
"\"Paragraph\"",
    "id":
"Element12",
    "height":
38.79999542236328,
    "width": 162,
    "posX": 0,
    "posY":
45.60000610351561,
    "type": 1,
    "childs": []
```

Figure 12: Editor.html

The editor is the main program the user interacts with. On the left-hand sidebar it features various buttons explained later in the text, input fields for the main style properties (top and left for position, width and height for dimensioning) and a text area where the layout data is represented as a JSON string, as well as a tree display of all layout elements. The right-hand area is used to display the generated knockout elements. In this document this area is referred to as 'range-div'.

These elements are not implementations of their actual types but draggable divs containing a label and a placeholder image. A element object is definded using the following constructor:

```
function element(id, type) {
    var self = this;
    self.internalID = ko.observable(id)
    self.id = ko.observable(isNaN(id) ? id : 'Element' + id);
    self.parent = ko.observable();
    self.childNodes = ko.observableArray([]);
    self.x = ko.observable(0);
    self.y = ko.observable(0);
    self.w = ko.observable(100);
    self.h = ko.observable(100);
    self.styleString = ko.observable('');
    self.innerTextString = ko.observable('');
    self.type = ko.observable(type);
}
```

Code-Snippet 29: element object constructor

Such an Element contains various variables for positioning and dimensioning, as well as a css style string and a string containing its inner Text. Further it references its parent and contains an array of it's child elements. These elements are managed using the EditorViewModel function. This function is also the program's MVVM ViewModel.

Inside this ViewModel, the elements are contained in an simple observableArray, which is bound to the range container using a custom template.

```
<div id="range" data-bind="event:{dblclick:rangeClick,click:setRange}, template:{name:
'elementTemplate',foreach:elementArray()}">
</div>

<script type="text/html" id="elementTemplate">
   <div class="draggable" data-
bind="attr:{id:internalID},event:{dblclick:objDblClick,contextmenu:rightClicked,
click:objClick},
dragResize:{parent:parent,x:x,y:y,h:h,w:w,id:id,last:self.chosenObject}">
      <div data-bind="template:{name: 'elementTemplate',foreach:childNodes()}">
      </div>
      <img src="res/Download.png" class="mapClass" alt="Element" />
      <label class="labelType" data-bind="text:type().label"></label>
   </div>
</script>
```

Code-Snippet 30: The markup and template for rendering element objects as draggables

This template binding populates the container with the results of recursively rendering the elementTemplate for each of the elements contained in the elementArray observable or an element's childNodes observable.

The dragResize binding is bound to every DOM-Element generated from this template. This custom binding handles the main part of this websites functionality. It enables the drag and resize abilities, performs event coordinating and compensating for various problems occurring when dealing with nested draggables. This binding consists of two callbacks, `init` and `update`, with the first ensuring the position, dimension and visibility of a newly added element, as well as attaching the resize handle and the second updating and constraining the property variables (so that for example a child cannot be positioned outside its parent)

4.2.2.1. Adding an element

When the range-div or any of its subcomponents are double clicked, a canvas pops up, enabling the user to draw a symbol. Upon clicking the finish link, the canvas' context is retrieved and its ImageData property, an array of RGB values representing the image the canvas is currently displaying. This array of RGB values is then converted into an array of binary values. The canvas has a size of 400 times 400. To reduce the amount of data we compress the binary picture to a size of 100 times 100 and perform a length encoding.

```
function getPicData() {
    //get the pixel of the canvas
    var canvas = $('#can')[0];
    var ctx = canvas.getContext("2d");
    var pix = ctx.getImageData(0, 0, 400, 400);
    //Compress data
    var content = binaryCompression(pix);
    //Resizing the data
    content = resizeData(content);
    //Length Encoding
    var compcontent = lenEncode(content);
    //calls socket.on(type,...) when probability is returned
    socket.emit('gettype', user.username, compcontent);
}
```

Code-Snippet 31: fetching canvas ImageData and sending it to the neural network

The returned type is used to add an element object with this type. It is added as a child of the object doubleClicked. If a doubleClick is perfomed on the range-div, the element is added to the EditorViewModels observable self.elementArray.

4.2.2.2. Altering an element

Altering an elements main attributes (id, top, left, width and height) is possible in two ways:

- The main attributes can be set by dragging and resizing the DOM elements contained inside the 'range-div'.
- Those main attributes can also be set using the input fields inside the sidebar.

All other style properties and the DOM elements text can be set using the contentMenuReplacer.

4.2.2.3. The contextMenuReplacer

Rightclicking on a draggable opens a custom context menu, the contextMenuReplacer. It allows for manipulating an element objects self.innerText and self.styleString observables.

```
<div id='contextmenuReplacer' data-bind="with:chosenObject">
    <p>innerhtml:</p>
    <textarea id='innertextTArea' data-bind="value:innerTextString,
           valueUpdate:'keyup'"></textarea>
    <p>additional style tags</p>
    <p>format: property:value;</p>
    <textarea id='styleTArea' data-bind="value:styleString,
           valueUpdate:'keyup'"></textarea>
</div>
```

Code-Snippet 32: contextMenuReplacer markup

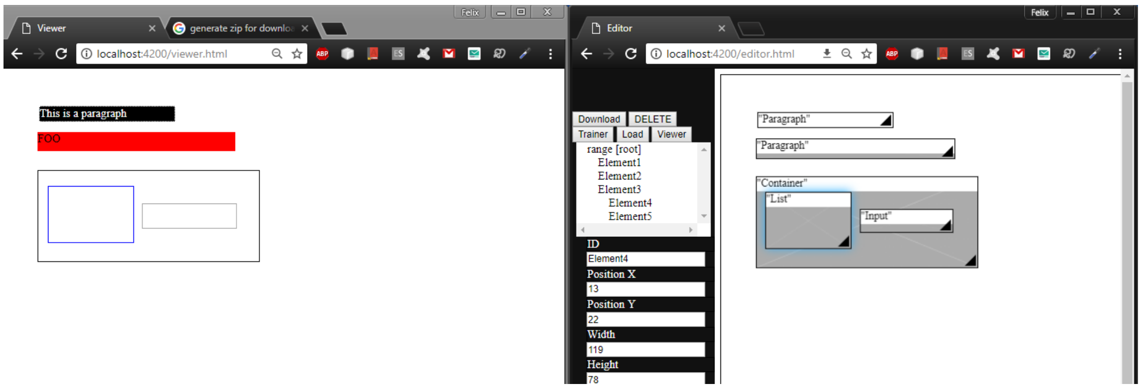
The contextMenuReplacer features two textareas where the observables can be changed. While the innerhtml area does not have a specific syntax to follow (you can even write HTML markup to prepopulate a list or table), the styleString follows the syntax of CSS, which is:

```
width: 189px;
height: 20px;
left: 48px;
top: 112px;
position: absolute;
border: 1px dotted white;
background: black;
color: white;
```

Code-Snippet 33: CSS properties

4.2.2.4. The Viewer

Clicking the button labelled Viewer will open a new tab displaying the viewer.html file. This application allows for a preview of the generated HTML page.



range [root]
Element1
Element2
Element3
Element4
Element5
ID
Element4
Position X
13
Position Y
22
Width
119
Height
78

Figure 13: Viewer (l) and Editor (r) displaying the same data

4.2.2.5. Downloading the files

When clicking the download button, two files will be created. The first is the HTML markup and the second is a JSON file that can be pasted into the editors textarea to reload it into the Editor.

5. Evaluation of used Technologies

5.1. Node.js

Node.js is a great concept for single paged, light-weight servers and applications. In our case to train and use a neuronal network it is not the best technology to use. Because of it being single threaded, we were not able to outsource the training of the network into an asynchronous thread, so we had to implement a separate application to handle the training and testing in order to ensure a stable flow of data between the web application and the Rest Service. My recommendation would be to use a more traditional backend programming language for the implementation of the servers like C# or Java.

5.2. Convnet.js

The API Convnet.js is a great tool to learn the concepts and basics of machine learning, because it makes it very easy to try different approaches and also provides a good tutorial and documentation on its webpage. There is one problem with the library. It is written in JavaScript which is not the best performing programming language nor the most stable one, which are booth crucial things for neuronal networks. If I had to implement a neuronal network again, I would recommend my contractor to use a Python or C++ library.

5.3. KnockoutJS

KnockoutJS is easy to learn, but hard to master. During this project I spent at least 180 hours working on the application and I am still certain that I do not yet have accessed its full potential. The MVVM pattern and its binding pattern were hard to grasp but during the year we worked with MVC a lot in class, made it easier for me to grasp the separation of MVVMs different areas. The only two problems I had was managing the dependencies and the events. As someone who is used to working without an observer pattern I often included unnecessary synchronizations. Event Handling with KnockoutJS is also a very tricky topic, especially with an event-heavy application as this projects frontend.

All this taken into consideration, I would recommend KnockoutJS to everyone as a method of organizing and synchronizing data on a website, and for the creation of single page applications, but in my opinion, it lacks as an event manager.

Sources

https://en.oxforddictionaries.com/definition/machine_learning [5.11.2017]

http://www.expertsystem.com/machine-learning-definition/ [5.11.2017]

http://whatis.techtarget.com/definition/machine-learning [5.11.2017]

https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/ [24.2.2018]

http://standoutpublishing.com/g/hidden-layer.html [12.3.2018]

http://whatis.techtarget.com/definition/supervised-learning [5.11.2017]

https://web.stanford.edu/class/cs345a/slides/12-clustering.pdf [8.3.2018]

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html [18.3.2018]

https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network#LayersofaCon-volutionalNeuralNetwork-PoolingLayer [18.3.2018]

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [18.3.2018]

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 [18.3.2018]

https://www.quora.com/What-are-the-advantages-of-using-Leaky-Rectified-Linear-Units-Leaky-ReLU-over-normal-ReLU-in-deep-learning?share=1 [2.4.2018]

https://cs.stanford.edu/people/karpathy/convnetjs/docs.html [18.3.2018]

https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm [21.3.2018]

https://nodejs.org/en/ [21.3.2018]

https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm [21.3.2018]

https://www.quora.com/What-are-the-disadvantages-of-using-Node-js [21.3.2018]

https://www.netguru.co/blog/pros-cons-use-node.js-backend [21.3.2018]

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [20.3.2018] https://www.lifewire.com/socket-programming-for-computer-networking-4056385

[20.3.2018]

https://jquery.com/ [22.03.2018]

https://en.wikipedia.org/wiki/Model–view–viewmodel [07.11.2017]

https://blogs.msdn.microsoft.com/johngossman/2006/03/04/advantages-and-disadvantages-of-m-v-vm/ [07.11.2017]

http://knockoutjs.com/documentation/introduction.html [22.03.2018]

https://github.com/greensock/GreenSock-JS/blob/master/README.md [23.03.2018]

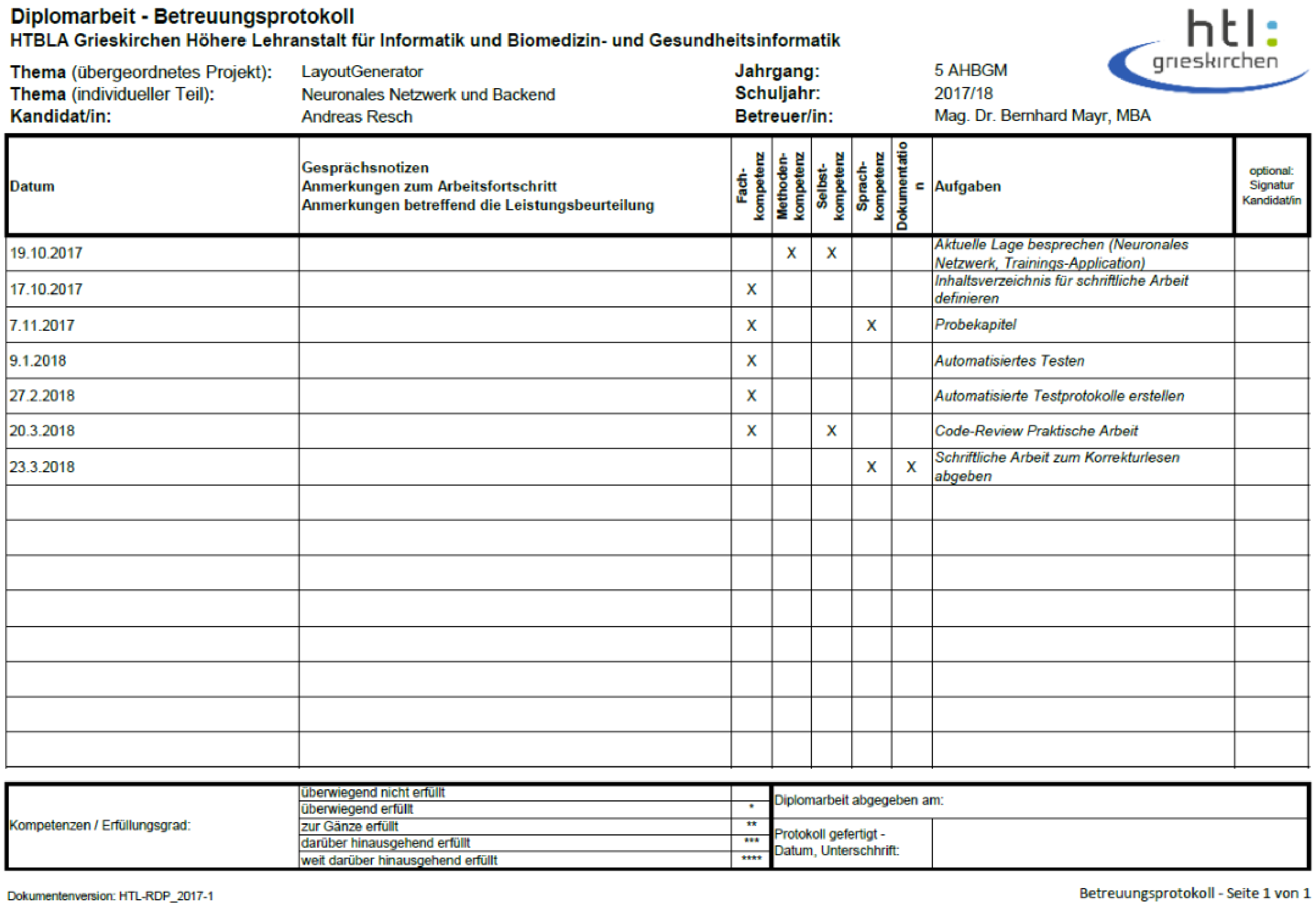
Picture Directory

Code-Snippet Directory

Term Definition

Term	Definition
Neuron	“A "neuron" in an artificial neural network is a mathematical approximation of a biological neuron. It takes a vector of inputs, performs a transformation on them, and outputs a single scalar value.” https://www.quora.com/What-are-neurons-in-machine-learning *[23.2.2018]*
Labelled training data	Every set of parameters has a given output value
OCR	“short for optical character recognition” https://en.oxforddictionaries.com/definition/ocr [6.11.2017]
MNIST training data set	“MNIST is like the "Hello World" of machine learning. Its a database of handwritten digits (0-9), with which you can try out a few machine learning algorithms.” https://www.quora.com/What-is-MNIST [2.4.2018]
MIT-license	“The MIT license provides one with the permission to reuse proprietary software, given the condition that the distributors include the copy of original MIT license with the distribution code (this is unlike GPL, LGPL) licenses.” https://www.quora.com/What-is-the-MIT-License [2.4.2018]
Event Loop	“The **Event Loop** is a queue of callback functions. When an async function executes, the callback function is pushed into the queue. The JavaScript engine doesn't start processing the **event loop** until the code after an async function has executed.” https://stackoverflow.com/questions/21607692/understanding-the-event-loop [04.04.2018]
Observer pattern	An object which notifies all its enlisted observers when its state changes Cf. https://en.wikipedia.org/wiki/Observer_pattern [07.11.2017]

Glue Code	“Source code only needed to adapt different parts of code that would otherwise be incompatible” https://en.wikipedia.org/wiki/Glue_code [07.11.2017]
NPM	“npm is the package manager for JavaScript and the world’s largest software registry.” https://www.npmjs.com/ [18.3.2018]



Diplomarbeit - Betreuungsprotokoll

HTBLA Grieskirchen Höhere Lehranstalt für Informatik und Biomedizin- und Gesundheitsinformatik

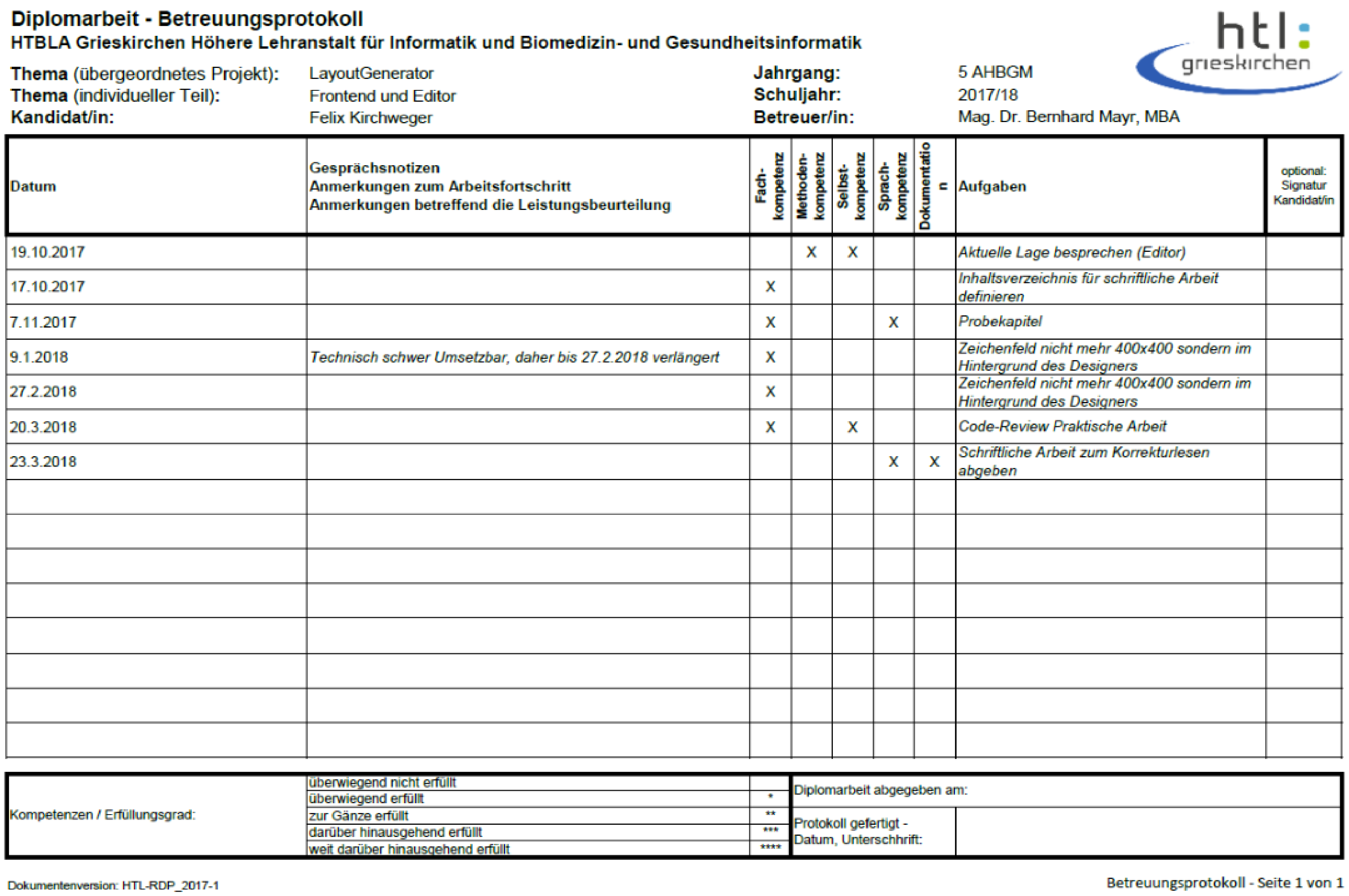
Thema (übergeordnetes Projekt):	LayoutGenerator	**Jahrgang:**	5 AHBGM
Thema (individueller Teil):	Neuronales Netzwerk und Backend	**Schuljahr:**	2017/18
Kandidat/in:	Andreas Resch	**Betreuer/in:**	Mag. Dr. Bernhard Mayr, MBA

Datum	**Gesprächsnotizen** **Anmerkungen zum Arbeitsfortschritt** **Anmerkungen betreffend die Leistungsbeurteilung**	**Fach-kompetenz**	**Methoden-kompetenz**	**Selbst-kompetenz**	**Sprach-kompetenz**	**Dokumentation**	**Aufgaben**	optional: Signatur Kandidat/in
19.10.2017			X	X			*Aktuelle Lage besprechen (Neuronales Netzwerk, Trainings-Application)*	
17.10.2017		X					*Inhaltsverzeichnis für schriftliche Arbeit definieren*	
7.11.2017		X			X		*Probekapitel*	
9.1.2018		X					*Automatisiertes Testen*	
27.2.2018		X					*Automatisierte Testprotokolle erstellen*	
20.3.2018		X		X			*Code-Review Praktische Arbeit*	
23.3.2018					X	X	*Schriftliche Arbeit zum Korrekturlesen abgeben*	

Kompetenzen / Erfüllungsgrad:	überwiegend nicht erfüllt		Diplomarbeit abgegeben am:	
	überwiegend erfüllt	*		
	zur Gänze erfüllt	**	Protokoll gefertigt - Datum, Unterschrift:	
	darüber hinausgehend erfüllt	***		
	weit darüber hinausgehend erfüllt	****		

Dokumentenversion: HTL-RDP_2017-1

Betreuungsprotokoll - Seite 1 von 1



Diplomarbeit - Betreuungsprotokoll

HTBLA Grieskirchen Höhere Lehranstalt für Informatik und Biomedizin- und Gesundheitsinformatik

Thema (übergeordnetes Projekt):	LayoutGenerator	**Jahrgang:**	5 AHBGM
Thema (individueller Teil):	Frontend und Editor	**Schuljahr:**	2017/18
Kandidat/in:	Felix Kirchweger	**Betreuer/in:**	Mag. Dr. Bernhard Mayr, MBA

Datum	**Gesprächsnotizen** **Anmerkungen zum Arbeitsfortschritt** **Anmerkungen betreffend die Leistungsbeurteilung**	**Fach-kompetenz**	**Methoden-kompetenz**	**Selbst-kompetenz**	**Sprach-kompetenz**	**Dokumentation**	**Aufgaben**	optional: Signatur Kandidat/in
19.10.2017			X	X			*Aktuelle Lage besprechen (Editor)*	
17.10.2017		X					*Inhaltsverzeichnis für schriftliche Arbeit definieren*	
7.11.2017		X			X		*Probekapitel*	
9.1.2018	*Technisch schwer Umsetzbar, daher bis 27.2.2018 verlängert*	X					*Zeichenfeld nicht mehr 400x400 sondern im Hintergrund des Designers*	
27.2.2018		X					*Zeichenfeld nicht mehr 400x400 sondern im Hintergrund des Designers*	
20.3.2018		X		X			*Code-Review Praktische Arbeit*	
23.3.2018					X	X	*Schriftliche Arbeit zum Korrekturlesen abgeben*	

Kompetenzen / Erfüllungsgrad:	überwiegend nicht erfüllt		Diplomarbeit abgegeben am:	
	überwiegend erfüllt	*		
	zur Gänze erfüllt	**	Protokoll gefertigt - Datum, Unterschrift:	
	darüber hinausgehend erfüllt	***		
	weit darüber hinausgehend erfüllt	****		

Dokumentenversion: HTL-RDP_2017-1

Betreuungsprotokoll - Seite 1 von 1