Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

BACHELOR'S THESIS IN INFORMATION SYSTEMS

Development of a Browser-Based Image Editor for TUMexam

Kempec Halk

Technical University of Munich

Department of Informatics

Bachelor's Thesis in Information Systems

Development of a Browser-Based Image Editor for TUMexam

Entwicklung eines Browser-Basierten Bildbearbeitungsprogramms für TUMexam

Author:	Kempec Halk
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Dipl.-Ing. Stephan M. Günther, M. Sc.
	Jonas Andre, M. Sc.
	Benedikt Jaeger, M. Sc.
Date:	15.02.2021

I confirm that this Bachelor's Thesis is my own work and I have documented all sources and material used.

Garching, 15.02.2021

Location, Date

Signature

Abstract

TUMexam is an exam management system that aims to digitally assist in the organization, correction, grade evaluation, and student review of exams. Initially, the application was designed to generate and process on-site examinations. Due to the COVID-19 pandemic and increasing demand for remote work, the platform received an update to support online submissions for exams and homework.

A key feature of TUMexam is the automated procedure of assigning every scan of an exam the correct page number and student. With remote exams, this introduces new difficulties as the uploads may be misaligned or distorted and include undesired background or a part of a different page.

In this thesis, we design and develop a browser-based solution to crop student submissions in case the TUMexam scan process does not recognize them. We analyze the emerged problem and derive the arising requirements for our application. We integrate our feature in the existing web frontend with a matching component in the backend. The browser interface offers user-friendly access to the necessary tools while the system backend carries out the instructions.

The result is a working implementation of a browser-based image editor in the platform to assist in the job of adjusting error pages. Our TUMexam extension provides high usability and allows for immediate familiarization with the functionalities.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

TUMexam [1] is a system assisting with various jobs during multiple steps of an exam. The application is developed at the Chair of Network Architectures and Services at the Technical University Munich. It is being led by Prof. Dr.-Ing. Georg Carle and has seen many improvements over the years since the beginning of development in 2015.

The platform aims to reduce workload and optimize exam organization procedures, correction, grading, and review. Repetitive tasks such as adding up problem scores and grading are being automated, and TUMexam is handling the effortful jobs of room assignments as well as reviews. This allows for a more fluid flow in every stage of exam-management and execution. Due to the review being held online, students similarly profit from this system. The time slots and limited period to inspect their exam results can be loosened, allowing a better understanding of the correction.

The system provides a web frontend for user-friendly access to the provided tools for instructors and correctors. A separate interface is offered for students allowing them to review their exams and upload online submissions. Developed with HTML [2], JavaScript [3], and CSS [4], those present a simple and powerful environment. The backend uses the micro web framework Flask [5] and is written in Python [6]. PostgreSQL [7] is used to provide the database.

The number of courses using TUMexam, together with the number of registrations for exams using the platform, has significantly increased since its first introduction. The amount of exams created and evaluated in TUMexam almost doubled from 2018 to 2019 alone. The uses jumped from a little over 15.000 to just below 30.000 [1]. Due to the COVID-19 pandemic and consequently the high demand for digital work, 200 courses used the service concurrently in the 2020 summer semester alone, with the Department

of Informatics already accounting for around 20.000 registrations using TUMexam by itself [8]. This shows the growing demand for the service but also presents new challenges with the current shift to remote exams.

Before the correction starts, all sheets need to be scanned and processed. TUMexam automatically carries out the tasks of linking an exam to the correct student, assigning each sheet the correct page-number, evaluating multiple-choice questions, and preparing the image to be ready for the correction. This scan process is designed with high-quality scans of the exam in mind.

With the introduction of remote exams and homework in TUMexam, the scan task started to run into some issues. Since the process was intended to work with professional scans, it is not equipped to handle low-quality student submissions. As a result of this, human intervention is necessary to assist with the assignment and revision of said pages.

In this thesis, we aim to develop an extension for TUMexam to adjust submissions within the platform itself. To do so, we implement a browser-based image editor providing the required tools. The feature is intended to support users in the job of revising unrecognized pages to be identifiable by the automated procedure. We design our integration to be seamlessly included in the workflow of fixing occurred scan errors.

In the following chapter, we explore similar exam platforms and existing solutions for related image editing. In order to comprehend the whole topic, we then describe some background information about the TUMexam system (Chapter 3). In the following chapter, we analyze the problem at hand by looking at the current situation, the cause, and the resulting consequences. On top of that, we derive the arising requirements for our intended implementation. In Chapter 5, we specify our limiting constraints given by various factors. In addition, we identify the precise goals for our feature. Subsequently, we determine our design decisions and present the choices based on the previous findings and set criteria. In Chapter 6, we elaborate on our implementation. To do so, we first explain the mathematical methods we use to calculate the input for various stages within our application. Afterward, we detail the approach used during the individual tasks. We evaluate the resulting functionality in Chapter 7 by measuring and analyzing the improvement in terms of reduction of workload and invested time as well as performance and impact on the system. Finally, we conclude our topic (Chapter 8) and present possible subjects for future work.

Chapter 2

Related Work

In this chapter, we explore selected systems to manage exams digitally similar to TUMexam. We survey their approach to handling difficult online submissions of paper-based tests. Furthermore, we investigate tools to edit scanned images in a web-based application. Subsequently, we compare TUMexam to the related platforms and review their procedure to guarantee an error-free transition into the correction phase. Moreover, we assess the existing tools in regards to their suitability for TUMexam.

2.1 Remote Examination Software

To gain an insight into different systems and their method to adjust unrecognizable submissions, we inspect two platforms built to assist in tests and allow students to upload handwritten solutions.

2.1.1 Gradescope

Gradescope [9][10] is a platform to administer, conduct, and grade on-site and remote exams. The project was initiated and originally developed at UC Berkeley and acquired by Turnitin in 2018 [11]. Similar to TUMexam, the system tries to assign exam questions and pages automatically. To do so, the system requires high-quality PDF scans of the individual submissions. In case the participant has no office scanner available, Gradescope recommends using smartphone applications able to produce digital copies of the sheets. In case the question matching failed, the student is asked to assign the correct page and declare areas for each unrecognized problem statement.

2.1.2 Crowdmark

Crowdmark [12] is another project to support exam organization, execution, and grading. It was initially created for the Canadian Open Mathematics Challenge (COMC) at the University of Toronto and since proceeds as an independent company. With Crowdmark, the cooperation of students is essential. The system heavily relies on uploads being adequately aligned and readable. The participant is asked to assign each file to the corresponding assignment. An image containing solutions to multiple problem statements is not supported. Therefore, a single submission containing all answers, like in TUMexam, is not possible. Although a section to fix scan errors does exist, it only allows instructors to manually correct the UUID (Universally Unique Identifier). The equivalent to this identifier in TUMexam is the ERID and is used on the cover page of an exam to map students to exam booklets as described in Chapter 3. However, the platform does offer a simple rotation of pages by 180 degrees during the correction.

Integration in Moodle

Crowdmark does allow integration with the Learning Management System Moodle [13]. The project is open source and designed to provide a personalizable education environment and is widely spread. On top of the primary purpose of offering course materials, it allows for digital tests. With Crowdmark as a plugin, paper-based remote exams and a corresponding assessment are made possible.

2.2 Web-Based Image-Editors for Scanners

In this section, we survey two projects offering a browser-based UI to edit scans. Both interfaces use the API Scanner Access Now Easy (SANE) [14] to obtain files and configure scanner settings.

2.2.1 phpSANE

The extension phpSANE [15] provides a web-based frontend with access to a limited number of tools to modify an image. The UI allows to select the scan area and specify the file format before the scans are saved. Additionally, the user can specify the resolution and adjust the coloring. Moreover, it supports the integration of OCR (Optical Character Recognition) [16], allowing for automatic recognition of printed text. The browser interface is written in HTML and PHP. It is an open-source project and licensed under GPLv2 (GNU General Public License, Version 2). The last commit to phpSANE was in 2013. The code is since abandoned and no longer maintained.

2.2.2 scanservjs

The project scanservjs [17] is a browser-based user interface for scanners. The integration is written in JavaScript and runs in the Node.js [18] environment. Scanservjs is the successor of scanserv [19] and started as an adaption of phpSANE (Section 2.2.1). The UI supports multiple scanners simultaneously with basic options to edit scans. The user can crop the image and modify brightness as well as contrast. Furthermore, it is possible to specify the compression level and output format before finalizing the scan process. The project is open-source and available under the GPLv2 license.

2.3 Comparison

Feature	**Gradescope**	**Crowdmark**	**TUMexam**	**phpSANE**	**scanservjs**
Student Mapping	✖	✔	✔	✖	✖
Page Recognition	✔	✖	✔	✖	✖
Page Split	✔	✖	✔	✖	✖
Crop Scan	✖	✖	✖	✔	✔
Flip Scan	✔	✔	✔	✖	✖
Rotate Scan	✖	✖	✖	✖	✖
Active Development	✔	✔	✔	✖	✔

Table 2.1: Features of related Platforms and Projects

Table 2.1 shows an overview of the surveyed projects and their features relevant to this thesis. The exam management systems Gradescope and Crowdmark do not provide instructors with the necessary tools to adjust an online submission to be suitable for the correction. Consequentially, the platforms heavily depend on the student to upload a proper copy of their solution. Furthermore, the participants are asked to handle errors and assign pages and solutions to the corresponding exam component if necessary.

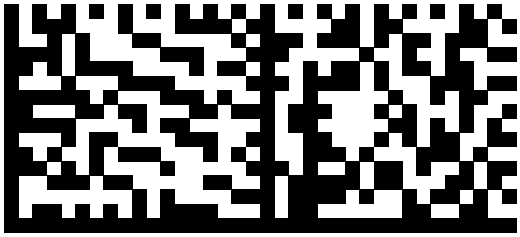
The browser-based user interfaces to edit scans we investigated only provide a very limited and basic set of tools. While the frontends offer a method to crop images, both lack the functionality to rotate or flip the input in any form. Additionally, phpSANE, as well as scanservjs, are integrated within the scan procedure and require interaction before the file is saved. Thus both extensions are not suited for large-scale exam digitization and remote submissions.

Chapter 3

Background

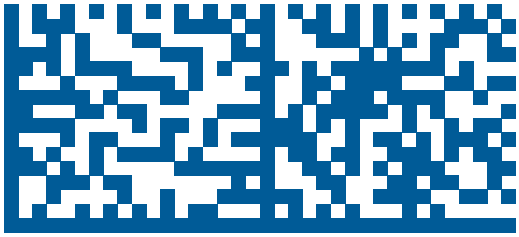
First, we describe how TUMexam proceeds to denote every exam for a distinctive identification. To do so, we investigate the structure and function of the individual codes embedded on each page. Furthermore, we explain the scan process within the system and elaborate on the different layers TUMexam uses for various stages.

3.1 Exam Creation



IN-GRNVS-1-20201214-E5002-03

(a) An example of a TUMexam page-code



S0173

(b) An example of a SRID-code

Figure 3.1: TUMexam Matrix-Codes

Typically an exam is generated using the TUMexam LaTeX-template. Alternatively, a PDF can be uploaded. The system generates a distinct page-code for a priorly specified number of exams individually. The resulting code contains the following information [20]:

- Department: Shortening of the corresponding department

- Instance and exam-id: Name of the TUMexam instance and id representing the exam
- Date: Year, month, and day of the exam in the format `yymmdd`
- ERID: Unique identifier of each individual exam copy
- Page number: The page number within the exam

Every single page contains this code in human-readable as well as a data-matrix [21] format at a specified location. An example of this is shown in Figure 3.1a. In this example, `IN` stands for the Department of Informatics. The component `GRNVS-1` stands for the instance and the exam-id. After this, the ERID (Exam-Relative Identifier) is defined by the sequence `E5002`. It is used to distinguish between individual exams and unique to each copy generated. Lastly, the page number in this example is `03`, representing the third sheet within this particular ERID. Those components together are used to precisely allocate every single sheet of an exam.

In addition to that, an SRID, Student-Relative Identifier, is assigned to every participant similarly (Figure 3.1b). Other than the page-code, here only the SRID itself is displayed in human-readable form, while the data-matrix contains data specifying the department, instance, exam-id, and SRID. This way, students are unrecognizable by revisers during the correction, eliminating possible bias. During the examination, this code is placed on the cover page in the form of a sticker. Due to this, the student can now be linked to an exam. The combination of page-code and SRID now allows for a precise determination and association of each sheet.

As an alternative, TUMexam supports registration boxes. Here the system generates several checkboxes on the cover page used for students to specify their matriculation number.

In the case of online-submissions, we have the advantage of every participant having a unique link to upload their solution. Therefore, every file is already linked to a student, and the sticker containing the SRID is embedded on the cover page in advance.

3.2 Scan Process

The scan process is responsible for the correct detection of said identifiers along with processing the submitted pages. Running through multiple stages [22], it checks each file for a page-code and, in the case of a cover page, for an SRID sticker. With a recognized code, the page-number and ERID of the exam are now specified. Simultaneously, the

sticker assigns the corresponding student to the identified ERID, allowing for all other exam pages to be attributed to the participant.

In case three codes are detected, the orientation can be determined by comparing the code locations to the ones on the generated exam-page. With one or two codes identified, the process is trying to estimate the position of the missing page-codes on the given image to the original using Scale-invariant Feature Transform [22]. Finally, the pages are being aligned using the obtained information in an affine transform. If this step fails or no page-code is recognized, the submitted file is labeled as error-page, and human intervention is required. This involves simple tasks like appending or ignoring the page as it is merely an additional sheet but also cropping the image manually if necessary.

Layers

During the different stages in the TUMexam workflow, modifications to files are stored in layers. As of now, the following layers are being used:

- Layer -1: Student submission
- Layer 0: Processed pages by the Scan process (Section 3.2)
- Layer 1: Annotations and Scores of first correction pass
- Layer 2: Annotations and Scores of second correction pass (if existent)

The method provides the ability to trace the last changes and allows access to any previous version if required. This is especially beneficial in the event of an incident causing a file to be damaged or faulty. Thereby, the risk of losing a page entirely is minimized.

Chapter 4

Problem Analysis

In this chapter, we analyze the current situation by comparing and reviewing the relative number of pages the scan process flagged as an error-page. Subsequently, we dissect the factors and causes for the evolved matter. Moreover, we illustrate the workflow currently practiced to handle the issues, estimating the severity of the problem. Based on that, the requirements for our implementation are assessed.

4.1 Failed Recognition

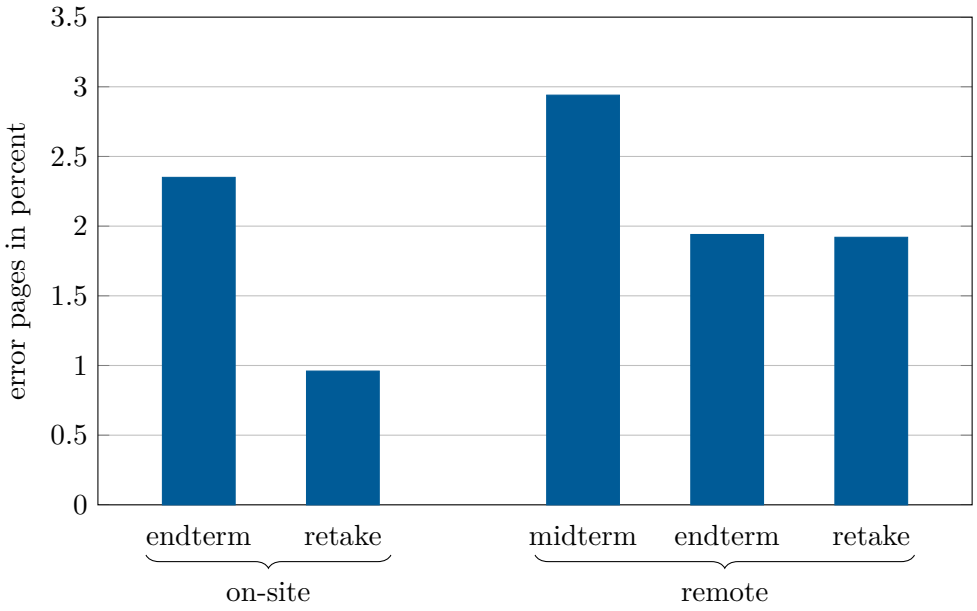


Figure 4.1: Error Pages: GRNVS 2020

	Exam	Error-Pages	Scanned Pages	Participants
on-site	Endterm	24	1020	85
	Retake	4	416	26
remote	Midterm	157	5338	743
	Endterm	143	7381	710
	Retake	87	4536	305

Table 4.1: Error Pages: GRNVS 2020

With the introduction of online submissions, the number of pages TUMexam failed to recognize, saw a substantial increase. As shown in Figure 4.1, the amount of error pages is considerably higher in remote exams compared to their on-site equivalent. A significant factor for this is the method students choose to submit their solutions.

The cleanest way to work on a remote exam is by using a PDF editor. Sheets can be populated, and problems solved digitally and uploaded afterward. To do, so a keyboard and, alternatively, a tablet or similar device supporting a stylus pen as input can be used. This also represents the advised method. Sometimes students do not follow this recommendation and use different approaches. They might not have the option to solve their exam in a digital form or do not want to and decide to print it instead and re-digitize it later on.

One way to do so is by scanning the pages using the same machine previously used for printing or, alternatively, an office document scanner. Because the platform was designed with professional scan stations in mind, those are typically not equally fit to attain the intended quality for the scan process. Submissions of this kind are generally still of decent quality and can be processed by TUMexam, only causing complications in rare circumstances.

The most common problem emerges from pictures taken with smartphones. Many mobile devices today can produce a high-quality photo, but this method can often result in other issues of various types the scan process is not equipped to handle.

- In many cases, misaligned or distorted images restrict the system in the ability to recognize or align the sheet reliably.
- Undesired objects in the background, such as a desk or carpet, can obstruct the operation.

- The picture may contain a part of a different page, adding the problem of having multiple different page-codes. Therefore, TUMexam is unable to identify and assign the submitted file accurately.
- Inconsistent lighting and shadows negatively affect the procedure [23].

Another popular option to bring the printout back into a digital format is the usage of document scanner applications for smartphones. Using the camera of the device, these try to recognize a sheet of paper, returning that section of the photo only as output. This technique resolves many issues induced by distortion and rotation. However, it may still impact the scan due to conflicting illumination on the image, often by the smartphone or the student himself.

Consequently, the scan process may be unable to derive essential information required to identify or align a page.

4.2 Current Workflow

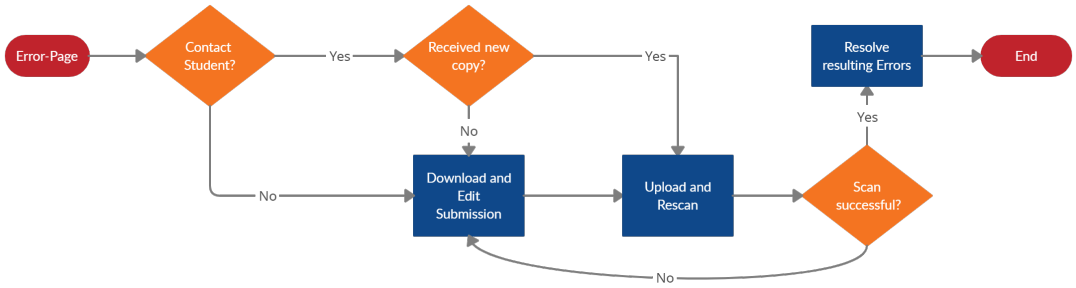


Figure 4.2: Workflow to gain a Working Copy of an Error Page

Presently, there is no way to select the desired area only from a submission in TUMexam. Thus this has to be done manually and locally. The responsible instructor has to run through multiple steps to correct said page, as demonstrated in Figure 4.2.

1. Since the exams only contain the SRID as relation to a student and are thereby anonymized, the instructor firstly needs to find out who the lousy submission belongs to.
2. Usually, the responsible participant is then contacted and asked to send a new copy of the page in question in an acceptable quality. This approach allows for a very promising outcome, with the student usually having an interest in his submission being evaluable. However, this also has the disadvantage of having to

rely on a third human factor. It might take a while for every person to respond as individuals might not be reading their mail frequently, for example.

3. When a fresh copy was finally received, it needs to be checked for changes by the instructor. The content on the page sent must match the initial submission, as students are not allowed to change their solution after the submission period has ended. This is essential to not provide anyone with the option of gaining an unfair advantage.
4. Suppose an individual fails to send an acceptable copy. In that case, the user is forced to modify the faulty image himself, trying to extract only the desired area so TUMexam can work with the result. This can currently not be done in TUMexam itself, however.
   (a) The instructor has to download the submission in question and decompress the archive to access the particular file.
   (b) Now the user needs to edit the page with a local image editor providing the required tools and export the outcome.
5. Furthermore, the reworked submission has to be uploaded to TUMexam once again in both cases.
6. Finally, this process can result in other errors like duplicate pages, which now have to be resolved in addition to that.

Not only does each step consume valuable time, but it also needs to be done for every single submission, not recognizable on its own, individually. Apart from this tedious process, there is no relationship between the original and modified file, making it a very sub-optimal and time-consuming approach. Consequently, the start of the correction may be postponed until the errors are taken care of and require more time and effort.

4.3 Requirements

Considering the factors and examples stated in Section 4.1 and the steps described in Section 4.2, we can determine several fundamental requirements.

(i) A functionality to choose and crop the desired area of an unidentified image has to be supported.

(ii) To allow for a more precise selection, a feature rotating the submission by a desired amount of degrees must be given.

(iii) In case a page is upside down on top of that, our implementation has to provide an option to account for that and flip the page. This also needs to respect the current utility granting this function to avoid any collisions or conflicts.

(iv) To prevent the necessity of exporting and importing any files, our application needs to run and be accessible within the TUMexam environment itself.

(v) A user-friendly interface in the frontend must allow for easy access with fast and uncomplicated tools to archive an optimal result. Ideally, this should be feasible in less than ten clicks.

(vi) The relationship between the initial submission and the edited image must be kept at all times. This way, it is always possible to review the original in a simple manner if an unclear circumstance requires doing so or any other situation demanding a revision emerges.

(vii) Finally, any potentially arising further errors have to be avoided. Our implementation must not result in a duplicate page or an additional new file not being recognized.

Chapter 5

Design

In this chapter, we identify the constraints we have to respect during the development of our application. We specify our goal and make design decisions based on all previously acquired factors.

5.1 Constraints

There are multiple restrictions to be respected. To always ensure accessibility and have the submitted pages of every student available, the original files must be preserved under any circumstances. A mistake caused by an error or human interaction in the cropping process should never influence the original, so this can be reversed, and the submission is never lost. Another matter for the need to keep this file is for legal reasons with the requirement to store exams for a certain amount of time. To provide constant relation of the originals to cropped and processed pages, those files need to be linked to their corresponding parent instead of being independent ones.

5.2 Design Goal

We want to integrate a tool to crop the needed section of a page within TUMexam itself.

(i) A crucial aspect is the efficiency of our solution with the time and interactions required to achieve the sought result. Our goal here is to realize this in at most ten clicks. This includes accessing and exiting our integration. Our set limit does cover extreme cases and familiarization with the tool as well. We want to provide an easily accessible feature with a straightforward layout and functionality.

(ii) The point of implementation should be in the error page section of TUMexam. This way, the image can be edited during the regular process of fixing scan errors without further interference required.

(iii) Our tool should provide an option to view and select the desired area on the page itself. On top of that, we want to offer the possibility of rotating the image to the correct orientation.

(iv) The final processing and crop of the page should be implemented and carried out on the TUMexam servers.

5.3 Design Decisions

To have a basis for the development, we decided on using the JavaScript library Cropper.js [24]. This JS provides the base functionalities we require to build the client-side implementation of the Image-Editor. It has ongoing support and is frequently updated, unlike many of its competitors. Available as an open-source project and licensed under the MIT license, the library embodies a good foundation for our TUMexam extension.

In order to have the required tools as accessible as possible, we take advantage of so-called modals. By opening a child window within the present working environment or browser-tab, we do not need to launch additional windows or applications, allowing for comfortable usability.

This component contains all the necessary tools we require to edit the chosen page. A canvas with the given image, as well as a flexible crop-box, are displayed to allow for direct interaction. The user can resize and reposition the box to select the desired output area. Alternatively, the mouse cursor can be used to mark a section on the canvas directly, creating a new crop-box. The user can choose between a fixed A4 aspect ratio or an unrestricted mode.

Additionally, a slider providing a rotation feature is presented. To serve a direct input turning the content inside the given canvas, an option to declare the exact number of degrees is provided. Moreover, the current amount can be increased or decreased using the arrow keys. Furthermore, the page can be flipped by 180° applied by a switch if the image is upside down. This function can be coupled with the previously mentioned rotation, allowing for a full 360 degrees. Finally, a preview of the selected area of the canvas is presented inside the modal and updated on any action or change made.

Submitting the modifications made also alters the image shown in the error page section of TUMexam. Rather than the original, the cropped version of the picture is now being displayed. Once all desired adjustments to the tied submission are made, the correlated data is sent to the server to finalize the result.

To keep the integrity of the submission, we decided to introduce a new layer (Section 3.2). In case of an image being assigned a new page-number and cropped, the newly introduced layer -2 is being populated with the raw page uploaded by the student. Simultaneously, the result is stored in layer -1 for further processing and easy access during the correction phase.

A page that is being appended to an exam is still stored in layer 0 after being cropped with the original remaining in layer -1. Whereas exam-pages are being rerun through the scan-process, appended images are not. Thereby they are not modified any further until the correction, so this is not required.

Chapter 6

Implementation

In this chapter, we realize the established design and develop our feature. To do so, we firstly define the mathematical calculations used. Followed by that, we go into detail on the separate components within the frontend and the backend. Here the different procedures and processing of each component are addressed and the individual implementations illustrated.

6.1 Two-dimensional object Transformation

To transform an image as intended for our purposes, an understanding of transformation matrices is necessary. Therefore we first review the basic principles and concepts of this transformation. In order to understand the formulas, a basic knowledge of euclidean space and matrix calculation is assumed.

6.1.1 Rotation Matrix

The rotation matrix [25] represents the basis for applying the corresponding action to an object. It is required to identify the new coordinates for a given point in an n-dimensional coordinate system. The standard rotation matrix rotates counterclockwise along the x-axis and about the point of origin. For a two-dimensional entity, we need to generate a 2×2 matrix. Let M_0 be our rotation matrix for a two-dimensional object, and $\theta \in [0, 359]$ be the angle of rotation performed. Then

$$M_0(\theta) = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \tag{6.1}$$

where $\alpha = \cos\theta$ and $\beta = \sin\theta$.

We denote the location of a given point p in our image by (x, y) and $p' = (x', y')$ as result respectively. We multiply matrix M_0 with the provided coordinates p so that we receive

$$p' = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cdot \alpha - y \cdot \beta \\ x \cdot \beta + y \cdot \alpha \end{bmatrix} \tag{6.2}$$

where p' is defined as the sought location.
For images, we can now use this to get the new location of every point and thereby the converted graphic.

6.1.2 Transformation Matrix

Because we want to rotate about the center of an image rather than its origin, we need to modify our rotation matrix.

To do so, we first need to create a translation matrix. Let

$$T(x_r, y_r) = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

be our matrix with x_r and y_r as p_r defining the point of rotation.

We need to add a row and column with a dummy coordinate to M_0, turning it into a homogeneous matrix. By multiplying the obtained matrix with $T(-x_r, -y_r)$ followed by $T(x_r, y_r)$ being multiplied with the result, we receive

$$\begin{aligned} M_T(\theta, x_r, y_r) &= \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \left(\begin{bmatrix} \alpha & -\beta & 0 \\ \beta & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} \alpha & -\beta & (1-\alpha) \cdot x_r + \beta \cdot y_r \\ \beta & \alpha & -\beta \cdot x_r + (1-\alpha) \cdot y_r \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{6.4}$$

as transformation matrix [26].

Due to the translations beforehand, p_r is now moved to the root. The input is shifted accordingly, rotated about the origin, and then translated back to the correct location. Applying

$$\begin{aligned} x'_i &= x_i \cdot \alpha - y_i \cdot \beta + (1-\alpha) \cdot x_r + \beta \cdot y_r, \text{ and} \\ y'_i &= x_i \cdot \beta + y_i \cdot \alpha - \beta \cdot x_r + (1-\alpha) \cdot y_r \end{aligned} \tag{6.5}$$

to every point i in a given canvas lets us compute the final transformed image.

In the following sections, we use the procedure explained and apply it to the different stages in our implementation.

6.2 Frontend

The frontend is written in HTML (HyperText Markup Language) and uses JavaScript for interactions. It communicates with the backend for an accurate and valid presentation on the client together with subsequent reliable execution on the server. TUMexam uses Bootstrap [27] to add the capability of providing the user with dialogs in various segments within the platform.

6.2.1 Cropper.js

Image Editor

We now realize the design described in Section 5.3. We use the already mentioned Bootstrap to grant user-friendly access, opening a modal with our application as demonstrated in Figure 6.1. Here all previously defined tools, as well as the live preview, can now be found and applied to the image in real-time. Whenever the image editing feature is launched, the client immediately checks whether there are values to crop the chosen page already. If so, the individual components in our window are adjusted to represent the correct options. On top of that, the canvas is resized to fit the image in case it was rotated. Finally, the preview is updated as well. After the user submits his changes, the feature stores all modifications made in the form of variables and closes the modal. Those are loaded in case of further adjustments and ultimately sent to the server backend.

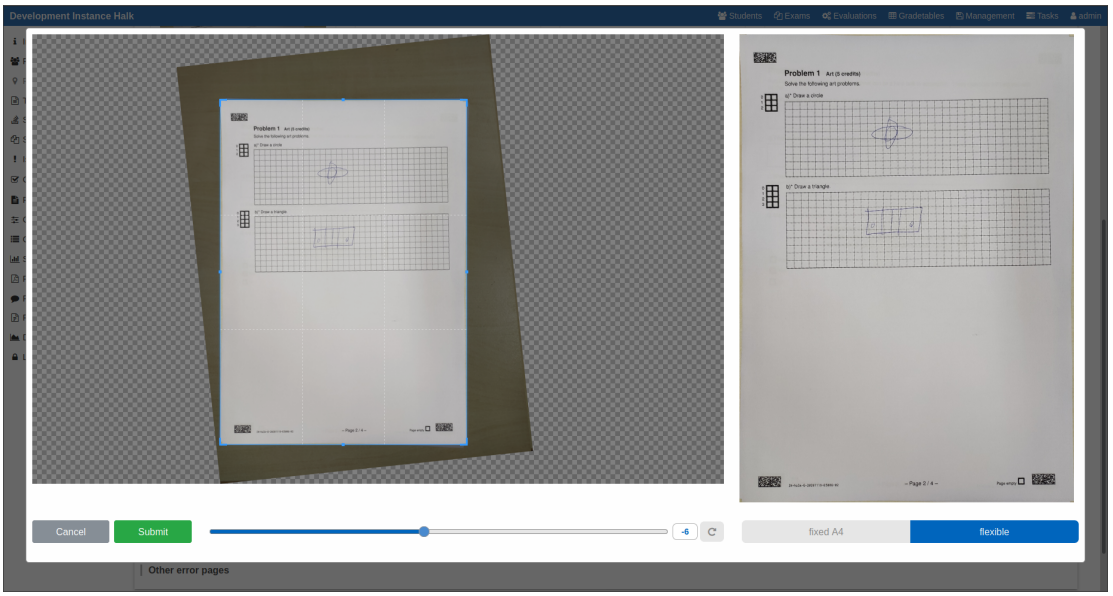


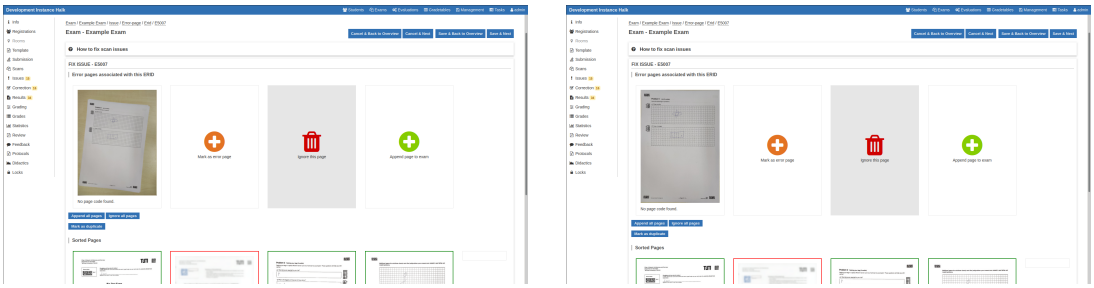
Figure 6.1: Image Editor in Modal

Calculations

We use the previously mentioned variables in the further steps within our implementation to correctly display and modify pages in TUMexam. Those values give us information about the coordinates, dimensions, and rotation set in the Cropper.js window. To be able to use this data, we need to understand how this library computes those variables first. As can be seen in Appendix A.1, Cropper.js uses a clockwise rotation around the center. To account for the coordinates becoming negative when rotating more than 180°, the modulo is being taken. This library uses radians to calculate α (Appendix A.1 line 927) and β (Appendix A.1 line 926). For better demonstration, we do the same in our code. Due to width and height changing places when rotated 90°, those variables are swapped. Finally, Cropper.js returns width and height, rotation in degrees, along with the coordinates of the point to crop from in pixel from the upper left corner of the resulting canvas as illustrated in Appendix A.1, but does not include the new dimensions computed.

6.2.2 CSS

Correct Presentation



(a) Error Page (b) Cropped Page

Figure 6.2: Error Page Preview

In TUMexam, error pages are shown as they are, as shown in Figure 6.2a. After modifying a page, this preview still demonstrates the original image. We rotate, resize and reposition the corresponding object on the website to display the selected area instead. To do this, we adjust the values in CSS for transformation and width accordingly. An example of the result can be seen in Figure 6.2b.

Calculations

The dimensions the browser works with do not adjust when an image is rotated. Therefore we need to account for that. Because the image is not actually being cropped here, we do not need to use the rotation matrix directly. Let

$$\begin{aligned} \alpha &= \left|\cos\left(\theta \cdot \pi/180\right)\right|, \\ \beta &= \left|\sin\left(\theta \cdot \pi/180\right)\right|, \text{ and} \qquad (6.6) \\ w' &= \alpha \cdot w + \beta \cdot h, \\ h' &= \beta \cdot w + \alpha \cdot h \qquad (6.7) \end{aligned}$$

where θ represents the angle of rotation and w together with h as well as w' with h', respectively denote the given and adjusted size of the image. Unlike Cropper.js (Section 6.2.1), we do not use the modulo of θ in any step nor swap width and height due to the fact that

$$\left|\cos\left((\theta \bmod 90) \cdot \pi/180\right)\right| = \left|\sin\left(\theta \cdot \pi/180\right)\right| \qquad (6.8)$$

for any angle. Thereby α and β are switched, causing the width to be calculated with the formula of height and vice versa. The Equations (6.7) are the sum of absolute

values from points $(w, 0)$ and $(0, h)$ with the multipliers determined in Equations (6.6) as described in Section 6.1.1.

Let x and y be the corresponding coordinates provided by Cropper.js with (w_0, h_0) defining the natural size of the image, then we receive

$$\begin{aligned} x' &= \left(\frac{1}{2}(w' - w_0) - x\right) \div w_0, \text{ and} \\ y' &= \left(\frac{1}{2}(h' - h_0) - y\right) \div h_0 \end{aligned} \tag{6.9}$$

as fractions, based on the original, on how far the object needs to be shifted in either direction.

Next, we need to scale the page. The parent container has a static width we want to fill. To do so, the cropped area is being resized to inherit its natural dimensions. With w_0/w_c, we gain the multiplier for width and height.

Since the parent container height is dynamic, we need to remove the resulting overflow from the bottom of the page. Let (h_c, w_c) be the size of the selected area and w_t be the width of the parent container, then

$$h_{neg} = \frac{h_c}{w_c} - \frac{h_0}{w_t} \tag{6.10}$$

gives us the fraction of the container we need to subtract from the bottom.

Now that we have everything required to display the correct part of the page on TUMexam, we apply those changes and set the point of rotation to the center of the given image.

Listing 6.1: Code to display correct area in TUMexam Error Page (Appendix A.2)

```
1 image.css({
2     "transform-origin": "50% 50%",
3     "transform": "translateX(" + (x' * 100) + "%) translateY(" + (y' * 100) + "%)
          rotate(" + θ + "deg)",
4     "width": ((w0 / wc) * 100) + "%",
5 })
6 image.css("margin-bottom", (((hc / wc) - (h0 / wt)) * 100 + "%"))
```

6.3 Backend

To implement our required server-sided functionality, we make use of the existing TUMexam backend. The platform already uses the Open Source Computer Vision Library (OpenCV) [28] for image processing of scans. The library offers a range of

features, including tools to modify two-dimensional objects. The suitable functionalities and already adopted dependencies render it the ideal candidate for our backend implementation.

6.3.1 Integration

Once a page is edited and submitted in the TUMexam GUI, the system instructs our integration to adjust the file accordingly. To do so, we provide the script with the values representing the rotation and cropping as specified by the user. To receive a precise result, we need to account for the different procedure once again, as described in Section 6.3.2. Subsequently, the server loads the image, applies the modifications, and saves the output as a new copy of the submission in layer -1. At the same time, the original student-upload is moved to layer -2.

6.3.2 Calculations

Unlike previous operations, OpenCV provides a method for creating a Rotation Matrix, as explained in Section 6.1.1, for us. Furthermore, we can specify the coordinates we want to be set as origin. However, this library does not account for changes in size after rotating an image. As a consequence, a part of our page is outside the canvas and thereby cut off. Similar to Equation (6.7), we need to compute the new width and height. Since the dimensions of the image and thus its center changed, we need to account for that as well and adjust matrix M_T. We obtain

$$M_T' = M_T + \begin{bmatrix} 0 & 0 & \frac{1}{2}(w' - w_0) \\ 0 & 0 & \frac{1}{2}(h' - h_0) \end{bmatrix} \tag{6.11}$$

as our new transformation matrix.

Since this matrix uses counterclockwise rotation, we use the negative value of θ as input along with the center as the point to rotate about. After applying the above calculations and adjustments, we create our rotated and resized image. Finally, we crop the page and return the result.

Chapter 7

Evaluation

In this chapter, we evaluate our design decisions and implementation by conducting multiple experiments. We perform tests to receive information about the optimization of the previous workflow as well as the usability and time consumption introducing our feature. Along with this, we conduct a trial to gain data on the performance impact our implementation has on the system. For each scenario, the devised setup is described. Subsequently, we present and assess our results.

7.1 Usability

The most important aspect of this thesis is to implement a user-friendly and time-efficient solution. Therefore, we conduct a test by asking volunteers to make use of our feature compared to the original approach as described in Section 4.2. Because we cannot estimate or evaluate the strategy of contacting a student, we ignore this method and focus on the concept of editing an image. The inconsistent and unpredictable human behavior makes it impossible to acquire an accurate measurement.

7.1.1 Setup

We provide five users with an identical set of drafted student submissions containing three error pages. Each of which requires rotation and cropping. The participants are given a short introduction to TUMexam and the working procedure. This way, we ensure prerequisite for everyone involved in the trial. We ask the subjects to perform two runs of the experiment. Once they are using our image editor in the TUMexam web interface and again using a local application.

We gather information on the time it took to finish a task. On top of that, we observe the number of user interactions measured in clicks and input. However, actions to accomplish one operation are joined. Thus navigating to a folder or selecting an input field to change the value count as one interaction.

The starting point of the experiment is defined as the error page section in TUMexam. Whereas the end is represented by receiving a working copy of the image in the system. Therefore the browser-based editing involves every step from opening the modal to submitting the final adjustments. Equally, using a local application includes all operations, from finding and downloading the correct submission to obtaining an assigned page.

7.1.2 Result

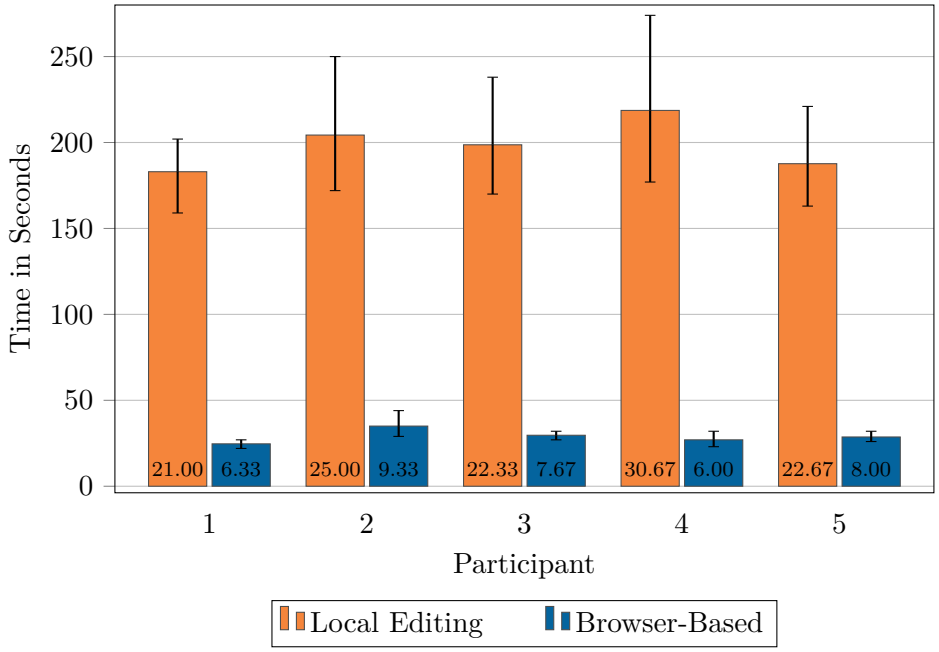


Figure 7.1: Time and Interactions required to edit a Student Submission

Figure 7.1 illustrates the duration each participant spent editing an image on average in seconds. The figure at the bottom of each bar specifies the number of interactions required. The lines included demonstrate the minimum and maximum duration each individual needed to finish a task. From the result, we can observe a significant improvement over the previous procedure. The shortest time a user spent on a single page using our implementation was 22 seconds, only requiring six actions with the highest amounting to ten in 44 seconds.

The local approach ranged from 157 seconds in 18 steps to 274 seconds in 38 operations for one image. The average here comes to 198.47 seconds. This value is not to be underestimated considering the number of error pages emerging from remote-exams, as described in Chapter 4.

Subjects had no issue familiarizing themselves with the components our browser-based image editor provides. Even during the first usage of our tool, no participant finished the job in more than 32 seconds. Moreover, the total average equals only 29.20 s. These results and the steadily low number of interactions in this approach indicate a user-friendly and self-explanatory user interface. Consequently, related issues can be taken care of in a less stressful manner than previous to our implementation. As the participants were unfamiliar with the system beforehand, further improvements in duration can be expected.

7.2 Performance

The server resource usage of our implementation is an essential aspect of the procedure. Additionally, the extra time until a subsequent scan process is finished describes an essential factor. Therefore, we run tests by passing trial data to our feature and collecting information on performance and execution period.

7.2.1 Setup

To assess the performance given by the server-side processing of images, as described in Chapter 6, we create a simulated scenario and feed it with differing data. The location to crop from, width and height, as well as the rotation, are randomly generated within a given range to produce realistic scenarios. The sample files we use are error page results of the TUMexam scan process and differ in size from 657 kB to 950 kB. Furthermore, we assume a system usage of a single core with two threads and a frequency of 1.80 GHz to 4.90 GHz. The provided Random Access Memory we use in our environment runs at 2667 MHz. The test itself generates a total of 5000 different cases. Subsequently, we evaluate the outcome of the experiment with the help of a resulting diagram providing the total runtime, memory consumption and, CPU utilization. To gain accurate values, we measure mentioned data in an interval of one second.

7.2.2 Result

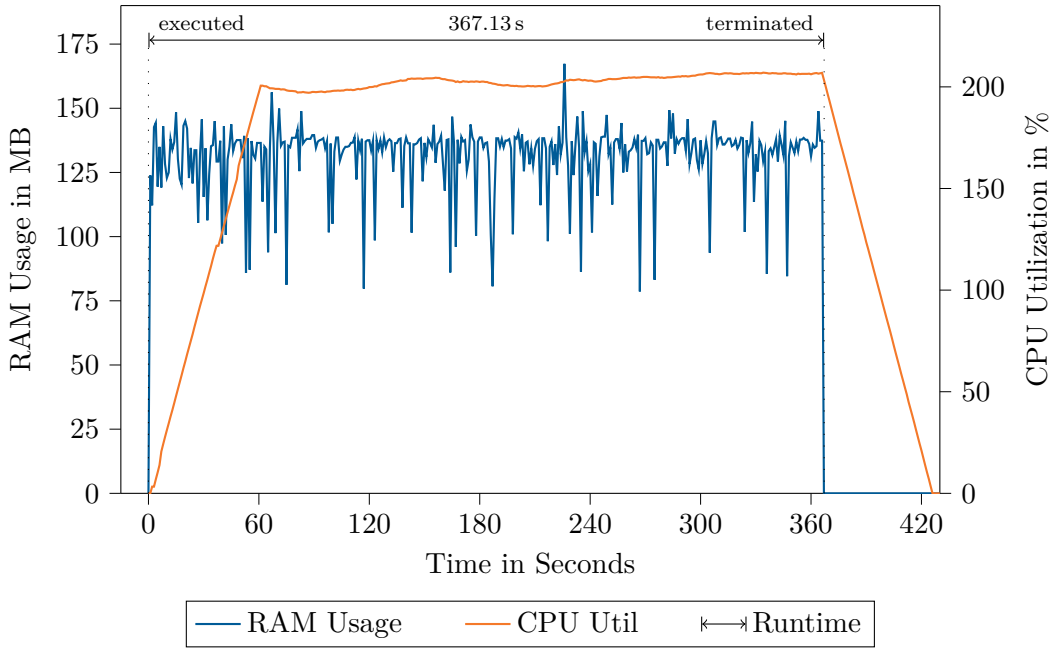


Figure 7.2: Server Load of Server-Sided Cropping Procedure

Figure 7.2 shows the results of the conducted scenario, including execution, termination, and runtime. The y-axis describe the RAM usage displayed in MB on the left along with the CPU utilization in percent on the right.

The memory usage varies depending on the input file, defined rotation, and area to crop. The average consumption amounts to 131.84 MB with a minimum of 78.65 MB and spiking to a maximum of 167.33 MB. The Central Processing Unit quickly reaches a peak of around 200% and remains at this level with slight fluctuation until every image modification is completed. Thus the CPU fully utilizes the provided core throughout the entire runtime. Finally, the process terminates in the 368th second with a total duration of 367.13 seconds for our sample input of 5000 files. This results in an average of 73.43 ms per page in our testing environment.

The server requires very little resources to handle vast amounts of requests to edit pages and does so in a short time period. With an average of 73.43 ms per image, while only benefiting from a single core and a fairly low memory consumption, an increase in duration the scan process requires is negligible. Furthermore, the implementation is only applied when a faulty submission requires human intervention. Hence the method is only being called once a user requests adjustments for a specific page rather than for every file during the scan.

Chapter 8

Conclusion

In this thesis, we successfully designed and developed a feature for TUMexam, which provides the necessary tools to modify unrecognized pages in the platform. With the increasing number of remote exams due to the COVID-19 pandemic, low-quality submissions became more frequent and introduced complications in the system. To offer a solution, we implemented a browser-based image editor in the existing web frontend of TUMexam to improve the current procedure to handle submissions not recognized by the scan process.

We derived the requirements in a deep analysis of the problem and identified the constraints for our extension of TUMexam.

Our implementation is divided into two sections. We integrated a user-friendly browser application in the TUMexam frontend but execute the final modification in the server backend. To do so, we first had to familiarize ourselves with two-dimensional transformation and rotation calculations. We understood how each component handles the data. We developed methods to convert the values for each unit to produce the correct output.

Our feature within the frontend is designed to be self-explanatory and requires very few interactions to achieve the desired output. We introduced an easily accessible modal in the web interface containing tools to edit a chosen error page. Here we used Cropper.js [24] to provide the basic functionalities. The opened window presents the user with a canvas showing the image to select the desired content and rotate the page freely. In addition to that, the current result is always shown in a live preview. To avoid confusion, we adjusted the demonstration of the error pages in the browser in case they were modified to display the cropped version instead.

The backend communicates with the frontend to provide the necessary information and, in return, receives data on how to edit a page. Our integration creates a modified copy of the image as instructed by the client and runs it through the scan process.

Our implementation reduces the workload and time invested in obtaining a working copy of an unrecognized page. The graphical interface allows for easy and quick access to the image requiring adjustments. The tools provided in our designed browser-based editor are straightforward and display a well-organized setting. Users can familiarize themselves with the feature within a few moments. We never exceeded ten interactions adjusting a submission and average on 7.47 necessary actions, suggesting high usability. The impact our TUMexam feature has on the server is kept to a minimum. With 73.43 ms to edit an image, the increment in time until the scan is completed is negligible. Therefore, we can assert that our application is a successful and beneficial extension to TUMexam.

Chapter 9

Future Work

In this work, we decreased the workload resulting from online-submissions by implementing a feature focused on cropping and rotating unrecognized pages.

9.1 Perspective Transformation

On some occasions, submitted documents can be distorted to a degree where TUMexam is unable to deal with the problem at hand anymore using only an affine transformation. Neither the scan process nor the current image editor is equipped with the functionalities required to repair these kinds of submissions reliably. At the moment, the image worker within TUMexam, as well as our implementation, actively works with a three-point affine transformation (Chapter 6). This approach, however, is not suited for issues involving severely distorted pictures.

Support of perspective transformation to correct the page in both mentioned features would improve the situation substantially. To do so, the image editor requires the option to select a 4-point, non-rectangle area within the picture. Furthermore, the capability to apply perspective transformation needs to be added to the previews in the browser, as well as the server-side script modifying the file.

9.2 Image-Editor for Students

The implementation is only available to instructors at the moment. Implementing this feature for students as well might further reduce the workload and serve a useful function for all sides.

With a complete implementation now operational for instructors, the next step could be to offer this feature to exam participants as well. As it is unclear if students would use an image editor in the TUMexam submission frontend, firstly assessing the value and gain, together with the demand and user acceptance of such, may be of advantage.

Of course, this needs to be secure, so users cannot modify data past the submission deadline. The existing image editor does provide a good basis for this since the only information transferred regarding the crop are the variables of how to modify a page instead of a new image. Thus tempering or replacing the file is not possible due to the checksum already transferred preserving the integrity of each submission.

9.3 Rework Scanned Pages

Currently, scanned and processed pages cannot be modified anymore. Those can be assigned an incorrect page-number though, and, in some instances, be distorted due to bad adjustments made by TUMexam. Thus a feature to reverse the processing and assignment of a particular page could be beneficial to the system. Returning an image to the original state of layer -1 (Section 3.2) would give the user the possibility to modify and reassign the submission.

Chapter A

Appendix

Listing A.1: Cropper.js Image Dimension Calculation [29]

```
degree = Math.abs(degree) % 180;

if (degree === 90) {
    return {
        width: height,
        height: width
    };
}

var arc = degree % 90 * Math.PI / 180;
var sinArc = Math.sin(arc);
var cosArc = Math.cos(arc);
var newWidth = width * cosArc + height * sinArc;
var newHeight = width * sinArc + height * cosArc;
return degree > 90 ? {
    width: newHeight,
    height: newWidth
} : {
    width: newWidth,
    height: newHeight
};
```

Listing A.2: Code to display correct area in TUMexam Error Page

```
const alpha = Math.abs(Math.cos(theta * Math.PI / 180))
const beta = Math.abs(Math.sin(theta * Math.PI / 180))
const new_width = alpha * w_0 + beta * h_0
const new_height = beta * w_0 + alpha * h_0
const new_x = (-1) * ((w_0 - new_width) / 2 + x) / w_0
const new_y = (-1) * ((h_0 - new_height) / 2 + y) / h_0
image.css({
    "transform-origin": "50% 50%",
    "transform": "translateX(" + (new_x * 100) + "%) translateY(" + (new_y * 100)
        + "%) rotate(" + (rot) + "deg)",
    "width": ((w_0 / w_c) * 100) + "%",
})
image.css("margin-bottom", (((h_c / w_c) - (h_0 / w_t)) * 100 + "%"))
```

Listing A.3: Code for Server-Side Cropping

```python
if abs(deg) != 0:
    (h, w) = img.shape[:2]
    center = (w / 2, h / 2)
    rot_mat = cv2.getRotationMatrix2D(center, -deg, 1.0)

    # Get alpha and beta from RotationMatrix
    alpha = abs(rot_mat[0, 0])
    beta = abs(rot_mat[0, 1])

    # Calculate new dimensions
    new_h = int((h * alpha) + (w * beta))
    new_w = int((h * beta) + (w * alpha))

    # Adjust the rotation matrix accordingly to new dimensions
    rot_mat[0, 2] += (new_w / 2) - center[0]
    rot_mat[1, 2] += (new_h / 2) - center[1]

    # Rotated Image with adjusted canvas to fit the whole image
    dst_img = cv2.warpAffine(img, rot_mat, (new_w, new_h))

return dst_img[int(dst_y):int(dst_y + dst_h), int(dst_x):int(dst_x + dst_w)]
```

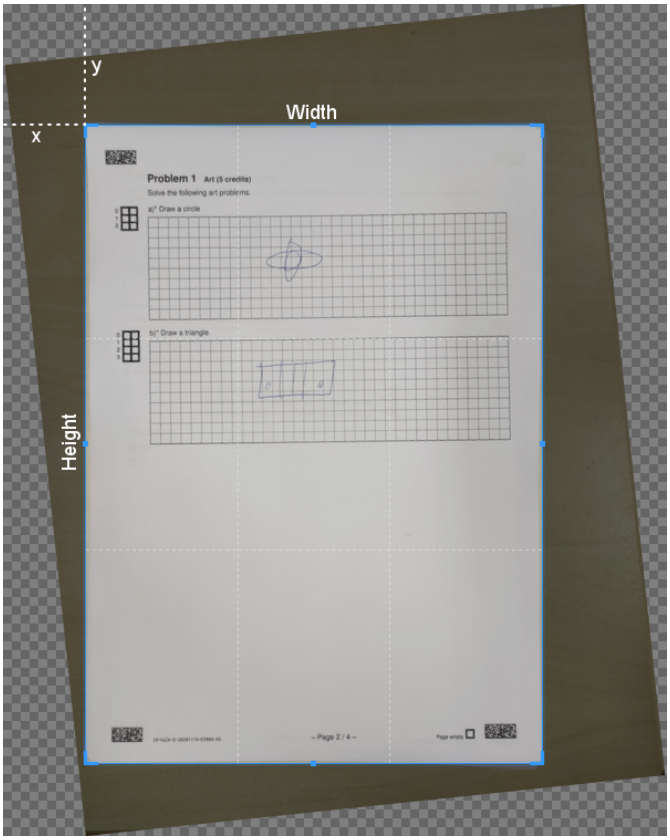


Figure A.1: Cropper.js Variables

Local Editing				
Participant	**Measurement**	**Page 1**	**Page 2**	**Page 3**
1	Seconds	178	159	212
	Interactions	22	18	23
2	Seconds	191	172	250
	Interactions	26	21	28
3	Seconds	188	170	238
	Interactions	25	20	28
4	Seconds	205	177	274
	Interactions	29	22	38
5	Seconds	179	163	221
	Interactions	24	19	25
Browser-Based				
1	Seconds	25	22	27
	Interactions	6	6	7
2	Seconds	29	32	44
	Interactions	9	9	10
3	Seconds	28	31	33
	Interactions	7	8	8
4	Seconds	26	23	32
	Interactions	6	6	6
5	Seconds	32	26	28
	Interactions	10	7	7

Table A.1: Result of each Page with both Methods of each Participant

Chapter B

List of acronyms

API	Application Programming Interface.
COMC	Canadian Open Mathematics Challenge.
CPU	Central Processing Unit.
CSS	Cascading Style Sheets.
ERID	Exam-Relative Identifier.
GPLv2	GNU General Public License, Version 2.
GUI	Graphical User Interface.
HTML	HyperText Markup Language.
JS	JavaScript.
MDN	Mozilla Developer Network.
MIT	Massachusetts Institute of Technology.
OCR	Optical Character Recognition.
OpenCV	Open Source Computer Vision Library.
PDF	Portable Document Format.
PHP	PHP: Hypertext Preprocessor.
RAM	Random Access Memory.
SANE	Scanner Access Now Easy.
SRID	Student-Relative Identifier.
UC	University of California.
UI	User Interface.
UUID	Universally Unique Identifier.

Bibliography

[1] TUMexam. (2021). "TUMexam Website", [Online]. Available: https://tumexam.de/ (visited on 01/20/2021).

[2] MDN, *HTML: HyperText Markup Language*. [Online]. Available: https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/HTML_basics (visited on 01/20/2021).

[3] ——, *JavaScript*. [Online]. Available: https://developer.mozilla.org/de/docs/Web/JavaScript (visited on 01/20/2021).

[4] ——, *CSS: Cascading Style Sheets*. [Online]. Available: https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/CSS_basics (visited on 01/20/2021).

[5] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[6] Van Rossum, Guido and Drake, Fred L., *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[7] P. G. D. Group, *PostgreSQL*, 2020. [Online]. Available: https://www.postgresql.org.

[8] Z. Sari, "Analysis of Modern Exam Management and Conduction Using Scan and Remote Exams and TUMexam in Particular", Bachelor's Thesis, Technische Universität München, Jan. 2021.

[9] Turnitin LLC. (2020). "Gradescope Website", [Online]. Available: https://www.gradescope.com/ (visited on 02/02/2021).

[10] A. Singh, S. Karayev, K. Gutowski, and P. Abbeel, "Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work", *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, 2017.

[11] Turnitin LLC. (Oct. 2018). "Press Release: Turnitin Acquires Gradescope", [Online]. Available: https://www.turnitin.com/press/turnitin-acquires-gradescope (visited on 02/02/2021).

[12] Crowdmark Inc. (2021). “Crowdmark Website”, [Online]. Available: https://crowdmark.com/ (visited on 02/02/2021).

[13] Moodle Pty Ltd. (2021). “Moodle project Website”, [Online]. Available: https://moodle.org/ (visited on 02/02/2021).

[14] *SANE Project Website*, 2020. [Online]. Available: http://www.sane-project.org/ (visited on 02/02/2021).

[15] David Fröhlich, John Walsh, Vincent Metalhead. (2013). “phpSANE SourceForge Project Website”, [Online]. Available: https://sourceforge.net/projects/phpsane/ (visited on 02/02/2021).

[16] Mithe, Ravina and Indalkar, Supriya and Divekar, Nilam, “Optical character recognition”, *International journal of recent technology and engineering (IJRTE)*, vol. 2, no. 1, pp. 72–75, 2013.

[17] S. Strachan. (2021). “scanservjs GitHub”, [Online]. Available: https://github.com/sbs20/scanservjs (visited on 02/02/2021).

[18] OpenJS Foundation. (2021). “Node.js Website”, [Online]. Available: https://nodejs.org/ (visited on 02/02/2021).

[19] S. Strachan. (2015). “scanserv GitHub”, [Online]. Available: https://github.com/sbs20/scanserv/ (visited on 02/02/2021).

[20] TUMexam, *Basics of TUMexam*, 2020. [Online]. Available: https://tumexam.de/static/exam-correction.pdf.

[21] “Automatic Identification and Data Capture Techniques – Data Matrix Bar Code Symbology Specification”, International Organization for Standardization, Geneva, CH, Standard, 2006.

[22] N. Bergbauer, “TUMexam – Joint Image Processing for Remote and On-site Exams”, Master’s Thesis, Technische Universität München, Dec. 2020.

[23] D. Esser, K. Muthmann, and D. Schuster, “Information extraction efficiency of business documents captured with smartphones and tablets”, *Proceedings of the 2013 ACM symposium on Document engineering*, 2013.

[24] C. Fengyuan, *JavaScript image cropper*. Version 1.5.9, Sep. 2020. [Online]. Available: https://chenfengyuan.com/en/projects/cropper.js.html.

[25] H. Goldstein, C. Poole and J. Safko, *Classical Mechanics*. San Francisco: Addison Wesley, 2002, vol. 3, pp. 141–144.

[26] D. Zwillinger, *CRC Standard Mathematical Tables and Formulae*. CRC Press, 1996, vol. 30, pp. 264–266.

[27] Bootstrap team, *Bootstrap Website*. [Online]. Available: https://getbootstrap.com/ (visited on 01/20/2021).

[28] G. Bradski, “The OpenCV Library”, *Dr. Dobb’s Journal of Software Tools*, 2000.

[29] C. Fengyuan, *Cropper.js Source Code*, version 1.5.9, Sep. 2020. [Online]. Available: `https://github.com/fengyuanchen/cropperjs/blob/master/dist/cropper.js`.