Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Maurice Hostettler & Lukas Boner

Digitalization of Chess Score Cards

Bachelor Thesis

Centre for Artificial Intelligence
Zurich University of Applied Sciences (ZHAW)

Supervision

Mark Cieliebak

June 2023

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Anleitung:
Kopieren Sie den untenstehenden Inhalt in Ihre Bachelorarbeiten. Dieser ist zu Beginn der Dokumentation nach dem Titelblatt mit **Namen und Datum** einzufügen.

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Name Studierende:**
Zürich, 20.06.23	Maurice Hostettler [illegible]
Winterthur, 20.06.23	Lukas Boner [illegible]

First and foremost we'd like to thank Prof. Dr. Mark Cieliebak and Pius von Däniken of the Centre of Artificial Intelligence at ZHAW, who have continuously supported us throughout the time we've written this thesis. Their dedicated feedback and constructive criticism were invaluable to us and made this paper possible.

Furthermore, we would like to thank Gundula Heinatz for providing us with many of the chess scoresheets needed to build our corpus and test our tool.

Special thanks also go to our friends Aurel Feer, Niklas van der Heide, Felix Saaro, and Pascal Schellenberg for proofreading and technical support throughout our time working on this thesis.

We are also especially grateful for the last-minute proofreading of Julia Boner, who showed us many mistakes and errors we had overlooked. She helped us tremendously to improve our thesis.

Finally, we want to express our profound gratitude to our friends, families, and partners for their unfailing support and ongoing encouragement throughout our studies. We would not have come this far without them.

Zusammenfassung

Die vorliegende Arbeit behandelt die Entwicklung einer Sammlung von annotierten Schachformularen, um Applikationen zu testen. Diese Applikationen digitalisieren, die an Schachturnieren von den Spielerinnen und Spielern mit ihren Spielzügen, ausgefüllten Formulare mittels Handschrifterkennung, der sogenannten Optical Character Recognition (OCR). Die Digitalisierung ist wichtig für die vereinfachte Veröffentlichung und Archivierung der Turnierresultate.

In früheren studentischen Arbeiten an der ZHAW wurde dafür bereits ein solches Applikations-Tool namens ChessReader entwickelt. Die ChessReader-Applikation wurde im Rahmen dieser Arbeit in eine Representational State Transfer- Application Programming Interface (REST-API) - Applikation umgebaut, weshalb auch Änderungen in der Handhabung und Speicherung der Daten vorgenommen wurden, um neu sehr aufwändige Annotationen automatisiert generieren zu können. Ein besonderes Augenmerk wurde darauf gelegt, möglichst unterschiedliche Formulare zu verwenden, um eine grosse Vielfalt in Bezug auf Lesbarkeit und Korrektheit der aufgeschriebenen Spielzüge zu gewährleisten. Zudem wurde darauf geachtet, dass weitere Sprachen und Notationsformulare hinzugefügt werden können.

Der in der Arbeit entwickelte Korpus beinhaltet aktuell 62 Spiele und etwa 4000 Spielzüge, davon 50 Spiele in englischer, 11 in deutscher und eines in französischer Schachnotation.

Abstract

This thesis covers the development of a chess scoresheet corpus, that can be used to test automated digitization applications. These scoresheets are created by chess players during tournament play and can then be digitized using Optical Character Recognition (OCR) to detect the players handwriting. The digitization of these scoresheets is needed for archival purposes and to publish the tournament games online.
In previous works at ZHAW, one of these applications, named ChessReader was already developed. The ChessReader application was repurposed to help with the labor-intensive task of annotating the corpus, which required a change to a Representational State Transfer- Application Programming Interface (REST-API) application. These changes also included a re-imagining of how data is used and stored during the process. Special attention was given to using as many different scoresheets as possible to ensure a large variety in terms of readability and correctness of the written moves. Additionally, sheets written in multiple different languages were included.
The corpus that was developed in the course of this thesis contains 62 games and around 4000 moves. It contains 50 games in english, 11 in german and one in french notation.

Contents

Nomenclature

Acronyms and Abbreviations

API	Application Programming Interface
CC	Creative Commons
CNN	Convolutional Neural Networks
CSV	Comma Separated Values
FAIR	Findable, Accessible, Interoperable, Reusable
FAN	Figurine Algebraic Notation
FIDE	Fédération Internationale des Échecs
HCS	Handwritten Chess Scoresheet
HTTP	Hypertext Transfer Protocol
ID	Identifier
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JWT	JSON Web Token
MNIST	Modified National Institute of Standards and Technology
OCR	Optical Character Recognition
PGN	Portable Game Notation
PNG	Portable Network Graphics
REST	Representational State Transfer
RSS	Really Simple Syndication
SAN	Standard Algebraic Notation
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
ZHAW	Zürer Hochschuhle für Angewandte Wissenschaften

Chapter 1

Introduction

Chess is an extremely old and very popular game. Currently, researchers trace its root back to a closely related game called "Chaturanga" in 7th century India[1]. Quickly it became part of courtly education as a lesson in war strategy. It has evolved from a game for kings and queens into both, a game enjoyed by thousands, as well as an internationally competitive sport.

Whether one plays chess on a piece of paper, in a park, on a tablet, or on a mahogany chess set, to improve one must play many games and analyze every move, to find the best strategy for a given situation. To do this, ambitious players record all played moves of a game, to be able to replay any moment again at a later date.

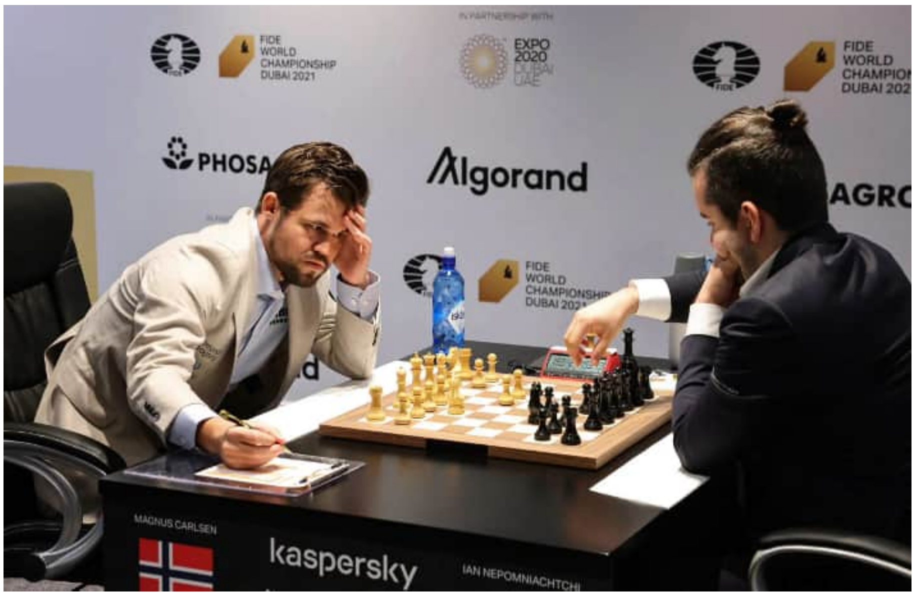


Figure 1.1: Magnus Carlson writing on his scorecard during the 2021 World Chess Championship.[2]

There are two main ways to accomplish this goal. One option is to use a digital board, that automatically keeps track of all pieces. Tools like Lichess can do this already. For in-person play on a physical board, it gets a bit more complicated. Either an expensive board with sensors, which recognizes where the different pieces are at any given moment, is needed or the players need to keep track of the moves by hand, using pen and paper. For that, usually, special forms are used, so-called "scorecards". On these scorecards, every player notes down their own moves and those of their opponent using Standard Algebraic Notation (SAN).

But since the world is becoming more and more digital, so is chess and the analysis of games. Many players and tournaments thus want to keep the games in a digital format, saving time and space. If the game was already played using a digital board or one that keeps track of the moves it is very easy to do that. On the other hand, if only paper scorecards were used, then each card needs to be input into the computer by hand.

This is where ChessReader[1] comes into play. ChessReader is a website where users can upload high-resolution images or scans of their handwritten chess scorecards. The images are then analyzed and searched for text. The recognized moves will then be presented to the user for final checking before being exported as a Portable Game Notation, (PGN)[3] file. This drastically reduces the repetitive workload of manually typing every move into a computer.

In order to accomplish its task, ChessReader uses Optical Character Recognition (OCR) to find and recognize text in images. Depending on the image quality, lighting conditions, handwriting style, etc. the results of OCR can vary vastly in terms of their accuracy. To be able to compare ChessReader to other tools and also to track performance improvements after updates, a reference point needs to be established. This requires a dataset on which the tools can be evaluated, encompassing images, varying in quality, lighting, etc. To verify the accuracy, the relevant data, eg. the moves played, needs to be checked by hand. In this thesis, we are going to define and start to build a corpus satisfying these requirements.

1.1 Motivation

1.2 Initial situation

At the start of this thesis, the aim was to improve the ChessReader tool, which was originally designed and implemented by other teams at ZHAW.[4] In doing so the need for a dataset to do testing that had not already been used to evaluate the ChessReader tool arose. The Handwritten Chess Scoresheet (HCS)[5] dataset was the most prominent set found[5] that was used for training a Neural Network from scratch. Since it was designed for training, it contained a large number of datasheets, however, they were annotated with the correct moves only. The images in the HCS were also already cropped, aligned, and contained only one kind of scoresheet, with neat lines. The included images also mostly featured neat and only English handwriting, instead of being representative of all the problems that arise during chess scoresheet digitization.

All in all, the HCS dataset was well suited for the training of a Machine Learning Algorithm, ChessReader however used external handwriting recognition. Therefore, testing on this dataset would not accurately represent what users of our tool might input, as there are many different kinds of scoresheets used, as well as many different ways of taking pictures for processing. The previous thesis already showed that scoresheets with nice lines, decent lighting, and rather neat handwriting were not a problem for ChessReader. Problems usually emerged when these conditions were not met.

Other datasets that were suitable for testing were not available, which is why the main topic of this thesis is trying to construct such a dataset. The goal is not only to build a great dataset to test the ChessReader tool on but also to have a comparative baseline for other tools. It started to become apparent this was less of a dataset and more of a corpus trying to accurately mirror the vastly different issues found in real-world examples.

In order to construct this corpus the ChessReader tool was selected to help annotate it. The current design of ChessReader however made it difficult to use it as more than it was designed to do, meaning the closed-off pipeline that was in place was not suitable to extract the needed data. Therefore changes to the backend of ChessReader were going to be needed. A more detailed description of these changes can be found in Chapter 4.2

[1]`http://chessreader.org/`

Creating a corpus of chess scorecards

In researching this project, a common theme was that there have been several attempts to solve this problem of Chess Scorecard Digitization. However, comparisons between different approaches and tools were anecdotal at best. In an effort to compare the work that had been done on ChessReader to that of others, but also to add a meaningful piece to the larger effort in this field, it is important to have a standardized reference point to compare advancements. Therefore it was decided to create a corpus containing pictures of scorecards with their respective digitization. Ideally, the corpus will contain some easy-to-solve problems and some harder-to-solve problems so that a perfect score will be very hard to achieve, allowing the tested programs to improve further.

Another important aspect will be that the different score sheets cover a wide variety of challenges in this problem. From players using notation abbreviations like "QxQ" for "queen takes queen", to improper lighting when taking the picture. An additional challenge might be to include chess notation from different languages. While English is the de facto standard for many of these score-sheets, a lot of places around the world use algebraic notation in their own language. Not only will this need adjustment in how recognized characters are evaluated, but it is also important to understand that different challenges might arise from local quirks in writing styles. As an example, Americans often do not cross the number seven while Europeans do etc.

1.3 Goals

The goal of this thesis is to design and build a corpus of chess scoresheets that can be used to evaluate and test applications that recognize the text on such handwritten sheets. The corpus will support multiple languages but will contain annotated scoresheets only in English, German and French notation for now. The moves on the sheets in the corpus are checked by hand to guarantee the correctness of the data and to add metadata such as readability or lighting conditions. The position of the tables containing the moves and the positions of all the boxes is also included in the annotation. Due to the amount of work required, the positions of tables and each box are not checked by hand and are instead generated automatically.

Overall we will follow our interpretation of Sinclair's principles of Corpus building[6] and respect the FAIR principles laid out by Wilkerson et al.[7].

In addition, the goal to modify the existing ChessReader application was also set to help create the corpus by being able to extract the positions of tables automatically. It requires first extracting the core functionality from the existing application and wrapping it inside of a clean REST-API interface and some small modifications to its code to get access to intermediate results of the processing so they can be included in the corpus.

Chapter 2

Background

This section contains a short introduction to the topics required to understand this work, the corpus, and the application itself. Basic knowledge of machine learning and artificial intelligence is considered prerequisite.

2.1 Chess scorecards

During a game of chess, players often record the played moves, either to be able to review the match and analyze it later or because they are required to do so when competing in a tournament. Either way, if the match is not recorded automatically, each player notes down the moves on a sheet of paper. To standardize the layout of these sheets, many chess organisations have introduced special "scorecards", usually consisting of some space to note down date, player names, as well as other metadata, and one or more tables where the actual moves for each color are recorded.

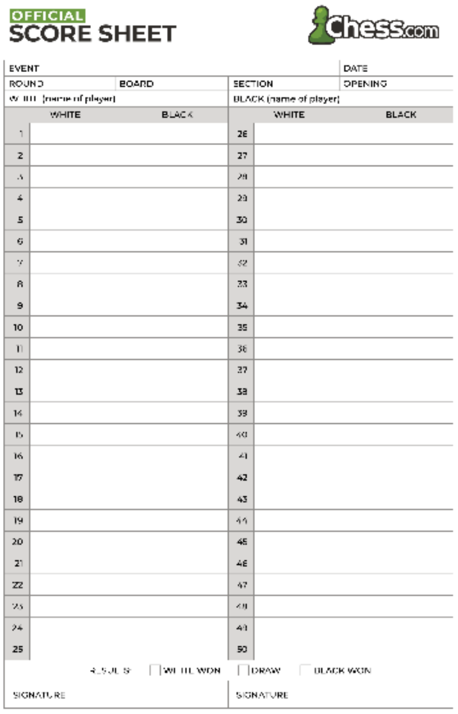
Figure 2.1 contains three examples of different designs of scorecards. The first sheet 2.1a consists of just a table, akin to how it would be displayed in say an Excel document. Each cell is contained within four borders and the columns are all of the same height.

In the next image 2.1b, the table consists now only of a few lines organized in columns. This might not seem like a big deal at first, but for computer vision it is quite a bit more difficult. The easiest way to detect tables is to recognize vertical and horizontal lines, but of course the vertical lines are missing in this example. One example of an issue that could arise because of this is, that the turn number is recognized as part of whites move, which would obviously result in a wrong OCR outcome.

The scoresheet in Figure 2.1c is even more different than the others. Not only does it contain a chessboard, which does resemble a table in its structure, but also includes additional columns for things like the time spent while making a move. The table is also split into two parts that are not the same height, which could further influence detection.

Overall it is obvious from these examples, that scoresheets come in many different shapes and forms, varying in design, layout, column count, etc. This means that the application needs to be flexible enough to be able to adapt to most of the designs that are used. While 2.1c is an extreme example, ChessReader and similar tools need to be able to deal with different amounts of columns of varying heights.

The only other solution to this problem would be to provide a specialized scoresheet, for which the application could be optimized. This would most likely fail to be adopted by a sufficient amount of users to justify the tool at all. In addition to that, many tournaments require players to use the provided sheets, which then still presents the same issues again.



OFFICIAL
SCORE SHEET
Chess.com

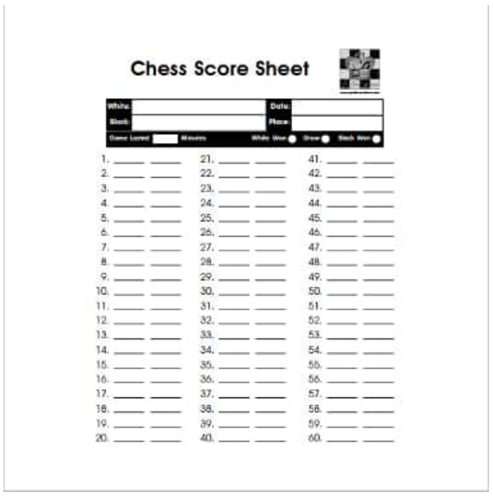
EVENT			DATE
ROUND	BOARD	SECTION	OPENING
WHITE (name of player)		BLACK (name of player)	

	WHITE	BLACK		WHITE	BLACK
1			26		
2			27		
3			28		
4			29		
5			30		
6			31		
7			32		
8			33		
9			34		
10			35		
11			36		
12			37		
13			38		
14			39		
15			40		
16			41		
17			42		
18			43		
19			44		
20			45		
21			46		
22			47		
23			48		
24			49		
25			50		

RESULTS ☐ WHITE WON ☐ DRAW ☐ BLACK WON

SIGNATURE SIGNATURE

(a) Clean lines, table easily recognizable, functional design



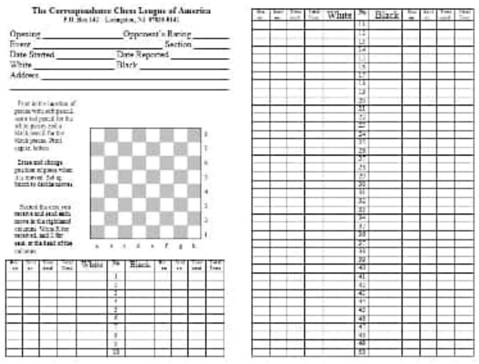
Chess Score Sheet

White:		Date:	
Black:		Place:	

Game Lasted Minutes White Won ○ Draw ○ Black Won ○

1.			21.			41.		
2.			22.			42.		
3.			23.			43.		
4.			24.			44.		
5.			25.			45.		
6.			26.			46.		
7.			27.			47.		
8.			28.			48.		
9.			29.			49.		
10.			30.			50.		
11.			31.			51.		
12.			32.			52.		
13.			33.			53.		
14.			34.			54.		
15.			35.			55.		
16.			36.			56.		
17.			37.			57.		
18.			38.			58.		
19.			39.			59.		
20.			40.			60.		

(b) More stylized, table is only implied with two columns of lines



The Correspondence Chess League of America
P.O. Box [illegible] Lexington, [illegible]

Opening Opponent's Rating
Event Section
Date Started Date Reported
White Black
Address

[illegible]

White No. Black

(c) Fields are clearly divided, but not all columns have the same height

Figure 2.1: Examples of different styles of score sheets

2.2 Standard Algebraic Notation

While playing in such tournaments, but also when playing just for fun, players use the so-called SAN notation. Standard Algebraic notation, also Algebraic Notation or Standard Notation is the only officially recognized way to record chess matches according to the "Laws of Chess"[8] by the International Chess Federation (FIDE).

In order to describe a move accurately, one needs to know three things:

- Which piece was moved?
- Where did it come from?
- Where was it moved to?

To describe the source location and the target location, each field is assigned a coordinate consisting of a letter of the Latin alphabet, denoting the column of the field and a number for the row. The field in the first row, first column would thus be the field A1 as can be seen in Figure 2.2 below.

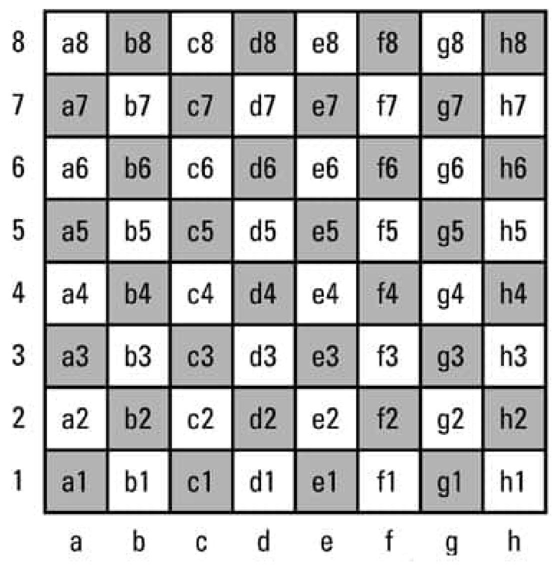
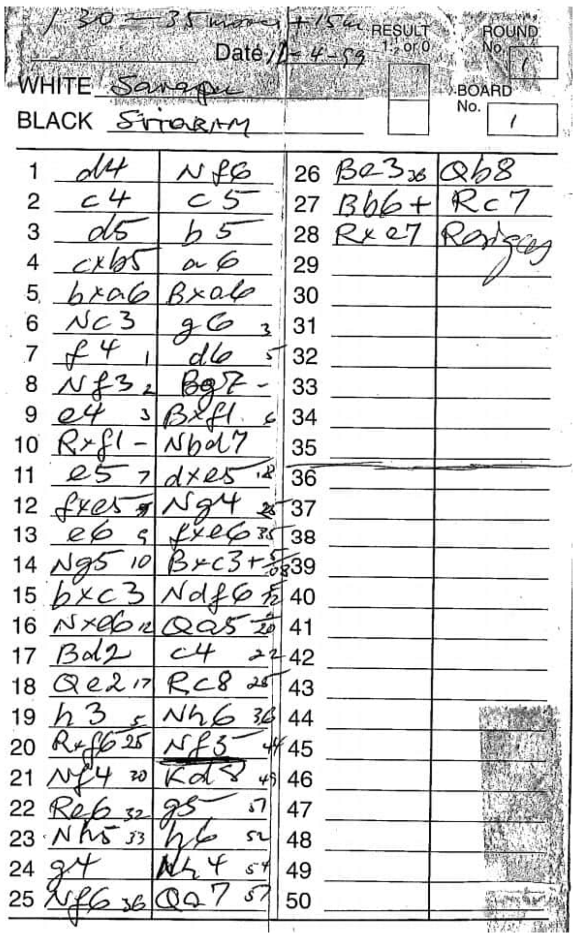

8 a8 b8 c8 d8 e8 f8 g8 h8
7 a7 b7 c7 d7 e7 f7 g7 h7
6 a6 b6 c6 d6 e6 f6 g6 h6
5 a5 b5 c5 d5 e5 f5 g5 h5
4 a4 b4 c4 d4 e4 f4 g4 h4
3 a3 b3 c3 d3 e3 f3 g3 h3
2 a2 b2 c2 d2 e2 f2 g2 h2
1 a1 b1 c1 d1 e1 f1 g1 h1
a b c d e f g h

Figure 2.2: Chessboard with a label for each field. [9]

The chess pieces themselves are assigned an uppercase letter of the Latin alphabet, which is dependent on the language which is used. In the English notation the letter "K" is used for the king and "Q" for the queen. Since "K" is already used, it can not be used for knights as well, which is why either "N" is used (or "S" in German for "Springer"). This complicates reading scorecards written in different languages, which is why in internationally published books and other articles, images are used in place of the letters, resulting in Figurine Algebraic Notation (FAN). In both SAN and FAN pawns are usually not assigned a letter or image, but are identified by the lack of one instead.

If the source field can be inferred from the type of piece moved and the target field, it can be omitted. This is the case if the piece in question is unique, like the King, or if the target can only be reached by one piece of the denoted type.



1 30 — 35 moves + 15 m Date 12-4-99

RESULT 1/2 or 0

ROUND No. 1

WHITE Sarapu

BLACK Sitaram

BOARD No. 1

1	d4	Nf6	26	Ba3 36	Qb8
2	c4	c5	27	Bb6+	Rc7
3	d5	b5	28	Rxc7	Resigns
4	cxb5	a6	29		
5	bxa6	Bxa6	30		
6	Nc3	g6 3	31		
7	f4 1	d6 5	32		
8	Nf3 2	Bg7 -	33		
9	e4 3	Bxf1 6	34		
10	Rxf1 -	Nbd7	35		
11	e5 7	dxe5 18	36		
12	fxe5 8	Ng4 28	37		
13	e6 9	fxe6 35	38		
14	Ng5 10	Bxc3+ 5/08	39		
15	bxc3	Ndf6 5/12	40		
16	Nxe6 12	Qa5 20	41		
17	Bd2	c4 22	42		
18	Qe2 17	Rc8 25	43		
19	h3 5	Nh6 36	44		
20	Rxf6 25	Nf5 44	45		
21	Nf4 30	Kd8 46	46		
22	Re6 32	g5 57	47		
23	Nh5 33	h6 52	48		
24	g4	Nh4 54	49		
25	Nf6 36	Qa7 57	50		

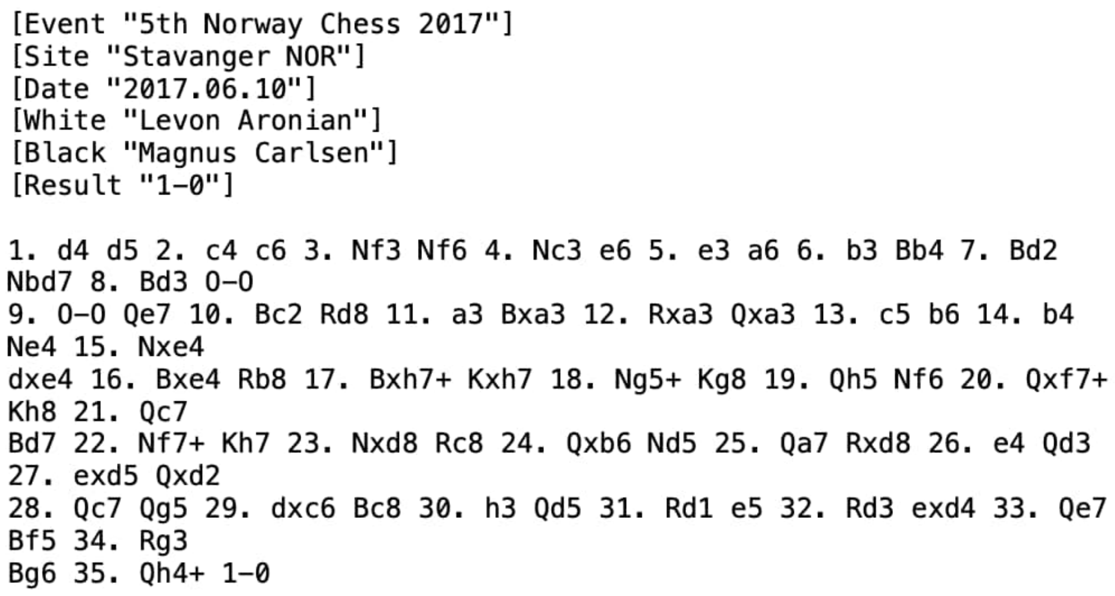
Figure 2.3: Filled out scorecard of the last game of renowned chess player Ortvin Sarapu

2.2.1 Portable Game Notation (PGN)

Nowadays, everything becomes more digital and so is chess. This requires PGN, a computer-friendly equivalent of SAN. Portable Game Notation is a human-readable, plain text format for recording chess games. It is supported by most chess-applications and serves as an easy way to transfer chess games between services.

The format consists of a first part describing metadata, like date, event, which round of the event this game was played in, site, players and who won, and a second part consisting of all moves that were played during the match.

The notation of the moves follows mostly the SAN notation explained above.



```
[Event "5th Norway Chess 2017"]
[Site "Stavanger NOR"]
[Date "2017.06.10"]
[White "Levon Aronian"]
[Black "Magnus Carlsen"]
[Result "1-0"]

1. d4 d5 2. c4 c6 3. Nf3 Nf6 4. Nc3 e6 5. e3 a6 6. b3 Bb4 7. Bd2
Nbd7 8. Bd3 O-O
9. O-O Qe7 10. Bc2 Rd8 11. a3 Bxa3 12. Rxa3 Qxa3 13. c5 b6 14. b4
Ne4 15. Nxe4
dxe4 16. Bxe4 Rb8 17. Bxh7+ Kxh7 18. Ng5+ Kg8 19. Qh5 Nf6 20. Qxf7+
Kh8 21. Qc7
Bd7 22. Nf7+ Kh7 23. Nxd8 Rc8 24. Qxb6 Nd5 25. Qa7 Rxd8 26. e4 Qd3
27. exd5 Qxd2
28. Qc7 Qg5 29. dxc6 Bc8 30. h3 Qd5 31. Rd1 e5 32. Rd3 exd4 33. Qe7
Bf5 34. Rg3
Bg6 35. Qh4+ 1-0
```

Figure 2.4: PGN file of a match between Levon Aronian and Magnus Carlsen

2.3 Optical Character Recognition

Optical Character Recognition (OCR) describes the process by which computers extract text from images. In the case of this thesis the focus lies on extracting handwritten text, which is a non-trivial challenge even humans can struggle with. Many solutions make use of machine learning to solve the problem by training on huge datasets to be able to extract rules. With this, the algorithm can make a guess which set of characters is likely to be shown in the image. As said before, this can be extremely difficult and an accurate result can not be guaranteed.

One such dataset algorithms are trained on is the MNIST (Modified National Institute of Standards and Technology) [10] dataset. It consists of approximately sixty-thousand images of handwritten digits to train a model and an additional ten-thousand of such images to be used as testing data.

Using this data it is possible to apply various Machine Learning methods to be able to recognize and label images of digits. Aside from linear classifiers, support vector machines, and deep neural networks, convolutional neural networks (CNNs) are also used, which seem to perform very well in this particular type of problem. In the paper "An ensemble of simple convolutional neural network models for MNIST digit recognition"[11] an error rate of only 0.09% was achieved using an ensemble of just three CNNs

Usually, OCR-applications consist of two stages:

First, the image is preprocessed, to align the text properly, convert the image to grey-scale since color information is not relevant, remove artifacts, and increase the contrast.

Second, the image is run through the "actual" OCR-algorithm, typically consisting of one or more CNNs of varying kernel size. During testing, the output of the algorithm is then also verified, in order to give feedback and improve the results further.

The challenges of handwriting recognition are fundamentally the same regardless of origin. This is why ChessReader relies on OCR-tools that have been trained on more general datasets of handwriting. This can unfortunately introduce a bit of additional error, which could potentially be avoided if an algorithm was specifically trained on a sufficiently big dataset of handwritten chess notation, like HCS.

2.4 Corpora

Sinclair's Principles

The design of this corpus will stick closely to a guide written by John McHardy Sinclair, titled "Developing Linguistic Corpora: a Guide to good practice"[6]. This ensures that it follows a certain baseline of the field. Since this guide was meant for corpora containing written texts instead of chess games, therefore the principles outlined will be adapted and modified where it is necessary to fit the purpose of chess scorecard digitization.

To start off, Sinclair's definition of a corpus: "*A corpus is a collection of pieces of language text in electronic form, selected according to external criteria to represent, as far as possible, a language or language variety as a source of data for linguistic research.*" [6] To rephrase for this corpus: A collection of pictures of chess Scorecards, in electronic form, selected to represent a large body of different chess games. Since the goal is to use these games as a dataset, only pictures in Portable Network Graphics (PNG) format will be considered, as opposed to other electronic formats, to ensure lossless storage.

Sinclair then defines a series of good principles that are to be considered when creating a corpus. Some of these criteria can be applied almost unchanged, while others can be ignored completely as they do not fit a corpus focused on images. The 10 principles are[6]:

1. *"The contents of a corpus should be selected without regard for the language they contain, but according to their communicative function in the community in which they arise.*
2. *Corpus builders should strive to make their corpus as representative as possible of the language from which it is chosen.*
3. *Only those components of corpora that have been designed to be independently contrastive should be contrasted.*
4. *Criteria for determining the structure of a corpus should be small in number, clearly separate from each other, and efficient as a group in delineating a corpus that is representative of the language or variety under examination.*
5. *Any information about a text other than the alphanumeric string of its words and punctuation should be stored separately from the plain text and merged when required in applications.*
6. *Samples of language for a corpus should wherever possible consist of entire documents or transcriptions of complete speech events, or should get as close to this target as possible. This means that samples will differ substantially in size.*
7. *The design and composition of a corpus should be documented fully with information about the contents and arguments in justification of the decisions taken.*
8. *The corpus builder should retain, as target notions, representativeness, and balance. While these are not precisely definable and attainable goals, they must be used to guide the design of a corpus and the selection of its components.*
9. *Any control of subject matter in a corpus should be imposed by the use of external, and not internal, criteria.*
10. *A corpus should aim for homogeneity in its components while maintaining adequate coverage, and rogue texts should be avoided."*

Principle 1 can be ignored right away since the communicative function of chess notation is the same across all languages, which simplifies this aspect. Principle 2 however is much more important. The language is in fact algebraic chess notation itself, regardless of the notation language the chess game was recorded in. Therefore, this can be adapted to mean that the corpus should reflect not just a variety of game types and lengths, but also a variety of notation languages. It should be as representative as it can be of the entire set of possible scoresheets.

Principle 3 can be understood as trying to include annotations that actually contrast different sheets. Meaning if an annotation for legibility and notation error is included there also need to be scoresheets that differ in those areas.

Principle 4 will be tackled by creating a clear outline and record of the annotation criteria and why they were chosen later in Chapter 3.

Principle 6 will be followed at face value. Since one of the main goals is to use the corpus for testing the quality of OCR tools for chess, it makes sense to include scorecard images the way they would be uploaded to such a tool. Meaning trying to mirror the way players and tournament organizers might photograph a score sheet and upload it. Most of the time this will be with a fairly low-resolution mobile phone camera, but could also include a better resolution where something like a scanner was used. Lighting conditions might also differ from picture to picture.

Principle 7 and 8 will be followed, by describing and justifying the decisions later in the thesis.

Finally, principle 9 and 10 will be ignored, as these apply mostly to text corpora since there is no such thing as a rogue chess game.

FAIR principles

The other foundation used for this corpus design is the FAIR-principles for scientific data management and stewardship.[7]

- *"To be Findable:*
    1. *(meta)data are assigned a globally unique and persistent identifier*
    2. *data are described with rich metadata (defined by R1 below)*
    3. *metadata clearly and explicitly include the identifier of the data it describes*
    4. *(meta)data are registered or indexed in a searchable resource*
- *To be Accessible:*
    1. *(meta)data are retrievable by their identifier using a standardized communications protocol*
    2. *the protocol is open, free, and universally implementable*
    3. *the protocol allows for an authentication and authorization procedure, where necessary*
    4. *metadata are accessible, even when the data are no longer available*
- *To be Interoperable:*
    1. *(meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation*
    2. *(meta)data use vocabularies that follow FAIR principles*
    3. *(meta)data include qualified references to other (meta)data*
- *To be Reusable:*
    1. *(meta)data are richly described with a plurality of accurate and relevant attributes*
    2. *(meta)data are released with a clear and accessible data usage license*
    3. *(meta)data are associated with detailed provenance*
    4. *(meta)data meet domain-relevant community standards"*

These principles were designed for much larger databases. However, following as many as possible will help to make using and expanding the corpus as seamless as possible.

To summarize, a compiled, shorter list of the most important criteria that were set based on Sinclair's and the FAIR principles that this corpus is built on:

1. Include games from different languages, game lengths, image quality, writing quality, etc. to arrive at a representative sample.
2. Create a clear outline as to what criteria will be recorded and how they will be annotated.
3. Make sure that the images and scorecards included mirror real-world examples and problems, not just artificially created "good" examples.
4. Include criteria and annotations that will support other future tasks and research as well.
5. Every image and its metadata should have a unique identifier that links them together.
6. The corpus should be published in a way to be accessible and contain a public usage license.

2.5 REST-API

Representational State Transfer (REST) is a software architecture that imposes certain conditions on how an API should work. RESTful APIs provide resources to a client, along with all the information needed for the client to identify, modify and/or delete said resource. Requests to REST-APIs are also stateless, meaning they are executed as a whole by the server, which allows for the request to be made in any order without the requests conflicting with each other. The advantage of such a system is that it allows for the complete separation of client and server. This layering allows for the different parts to evolve independently, improving flexibility and maintainability. For example, the application does not need to be changed at all, in order to make a change to how data is stored on the server. Additionally, since all requests are stateless, the server does not need to store any data related to requests, which preserves system resources and allows for better scalability if the need for that arises.

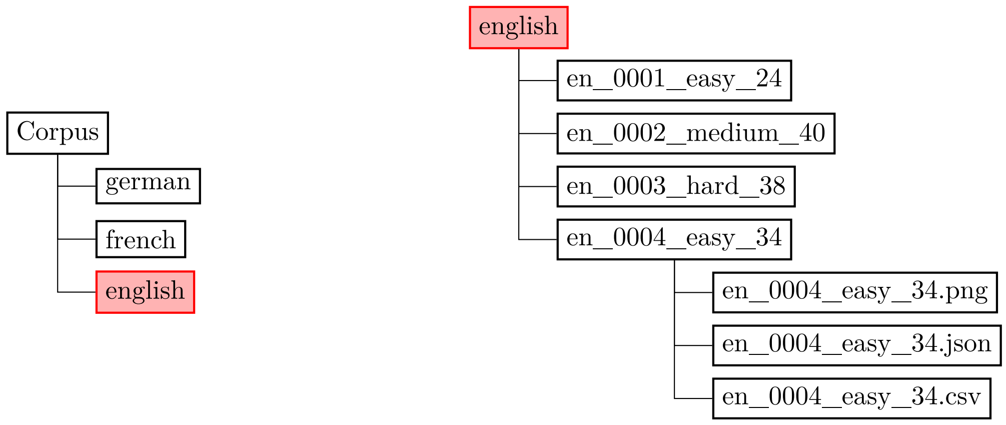
Chapter 3

Corpus Design

In order to fulfill the criteria outlined in 2.4, this chapter will now outline the way the corpus of scoresheets was designed. It describes in detail how the corpus was structured as well as how the included images were annotated. The annotations themselves are divided into two categories, the move annotations and the metadata annotation. The move annotations contain all the information that pertains to individual cells of the scoresheet, which contain the moves played, while the Metadata Annotations describe the information for the whole game, as well as the qualities and characteristics that are image specific.

3.1 Structure

In this section, the detail of how the corpus was structured and what the goals to be achieved were. Since the thesis was limited in scope, this aims to document clearly how the files were built and annotated, so that the collection can be easily expanded. It was also important to ensure that this corpus is not limited to just being used for digitization tasks only, which is why it was necessary to make it as transparent as possible so that other fields could use it as well.



This figure shows the file structure of the corpus with the English folder selected.

The naming of the files follows the structure of 2 symbol language code, 4 symbols for the identifier (ID) of the sheet, and then the difficulty and the game length.

<language>_<ID>_<difficulty>_<game length>

Sometimes there are multiple images that are tied to the same game, namely then when the game is longer than can fit on a single page. In those cases, the ID's are connected with an &. For example "0001&0002". The rest of the naming stays the same, as they are connected to the game, not the image. If the difficulty rating would change from one image to the other, the more difficult rating is applied.

Next, the choice of file formats is justified. PNG was as the format for the image of the scoresheet primarily because it is a lossless format. In some cases, especially pictures that were taken with phone cameras and automatically compressed, the obtained Joint Photographic Experts Group (JPEG) images were manually converted to PNG files.

For saving metadata JavaScript Object Notation (JSON) was selected. The primary reason for this is that it is both machine and human readable, so it allows for direct import into a variety of programs without needing to be parsed first, while still allowing for easy editing and readability for other uses. The JSON files will contain information about the game that is being annotated as a whole, as well as information about the image quality. The exact contents of this file will be discussed later in this chapter.

Finally the game data, i.e. the moves, and their annotations are saved as a Comma-Separated Values (CSV) file. Originally the idea was to also save these as JSON files, in the end however, CSV was selected, as it allows for easier processing using Excel or similar tools. Since chess moves are always recorded in the form of a table, it made sense to use a format very commonly used in the processing of tabular data. It also still allows for easy human readability.

3.2 Annotations

This section will cover how the images in the corpus were annotated. First, the move annotations will be discussed after which the metadata annotations pertaining to each analyzed sheet are described.

3.2.1 Move Annotations

In order to explain the annotation decisions, consider the table below as an example. One turn is always put on a row, so it mirrors the way it would usually be written on a scoresheet. The example was taken from an easy German scoresheet but works analogously for other languages and difficulties. Each turn is represented by a row in the CSV, so there is a white move and a black move per row.

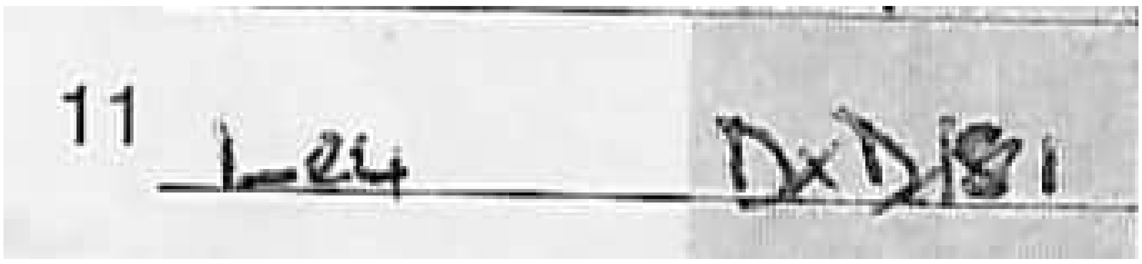

11 Le4 DxDd81

Figure 3.1: An example to illustrate how the .csv file was used to annotate.

corrected white	**uncorrected white**	**is hard to read**	**has notation error**
Le4		0	0

corrected black	**uncorrected black**	**is hard to read**	**has notation error**
Dxd1	DxDd81	1	1

- **corrected{white/black}:** Contains corrected move for each colour.

- **uncorrected**: Completely unedited and contains only the exact data that a handwriting recognition algorithm or a human might detect. In cases where the "uncorrected" move is the exact same as the corrected move, the "uncorrected" column was left blank entirely.

- **is hard to read:** If the cell containing the handwriting is hard to read, this was marked 1, 0 otherwise.

- **has notation error:** This cell will contain a 1 if the notation does not correspond to correct SAN. 0 otherwise.

In some instances, it is impossible to determine what move was played. This can either happen due to extreme illegibility or because the notation was for a wrong move entirely, meaning it did follow SAN, but the player wrote down a different move than the one that was played. In the case of complete illegibility, "illegible" is written in the corrected field and in the uncorrected field what the annotator managed to decipher. In the case where a player wrote down a completely different move, it gets tricky. Sometimes it is possible to notice this immediately because the move that was written down is not a legal move. In that case, it is marked as a notation error, and the "wrong" move is written into the uncorrected field. If it is possible to determine what move was actually played, based on future or past positions, the correct move is written into the color (black or white) field. If it is still ambiguous, the color field is marked with "unknown". In the following, some examples are discussed to illustrate the decision-making process.

Examples that are hard to read

The following examples show how the decision of what fields are hard to read or what fields contain a notation error was made.

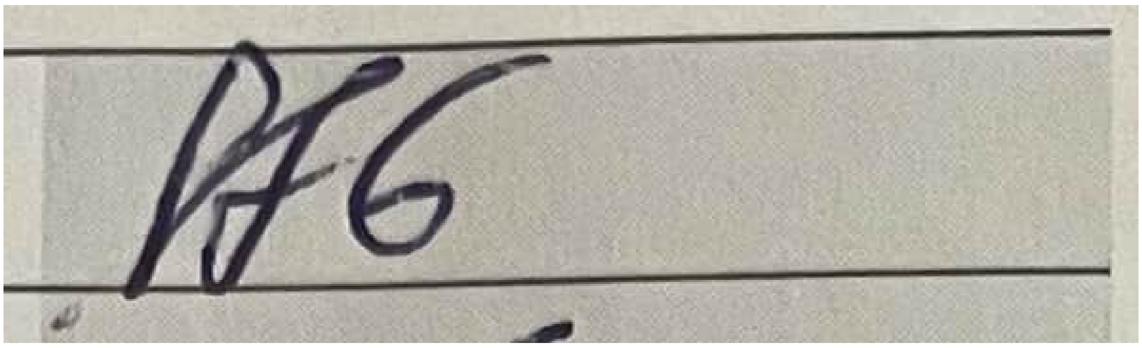


Figure 3.2: A hard-to-read notation, reads Sf6

In Figure 3.2 the simplest case can be seen, where the S in Sf6 is difficult to read due to the handwriting. In these cases, the annotator judged whether they found it hard to read. German notation is used for this and the next example.

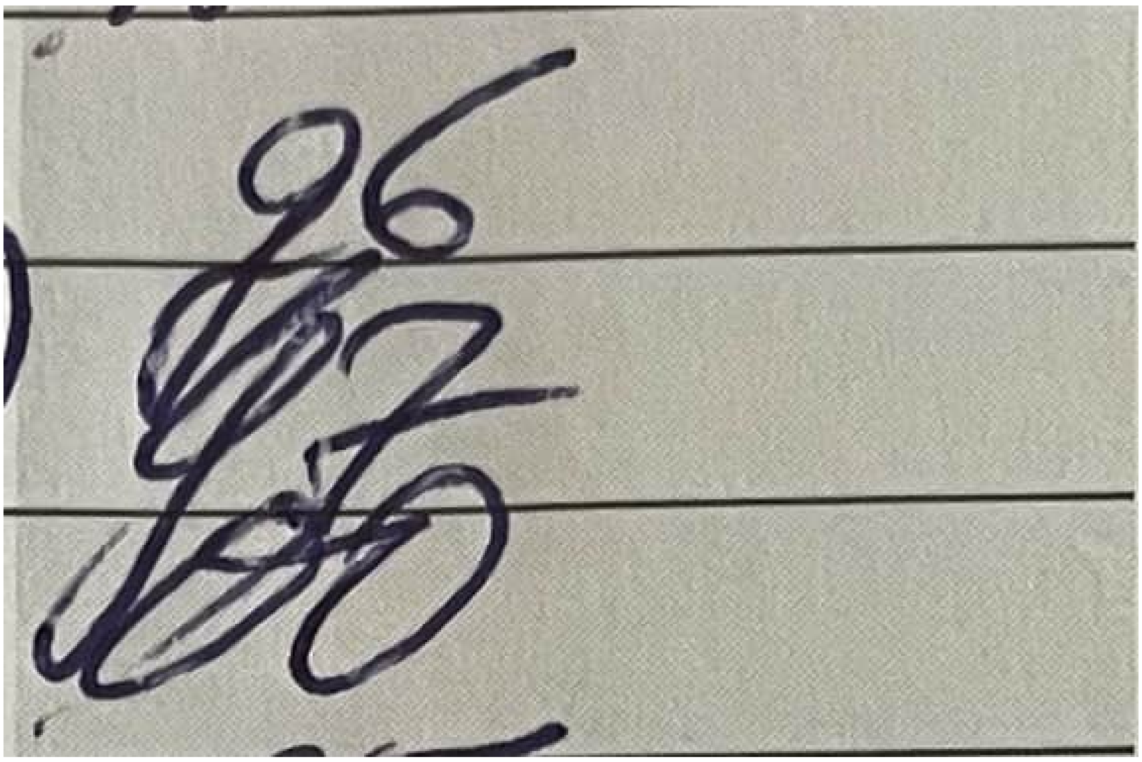
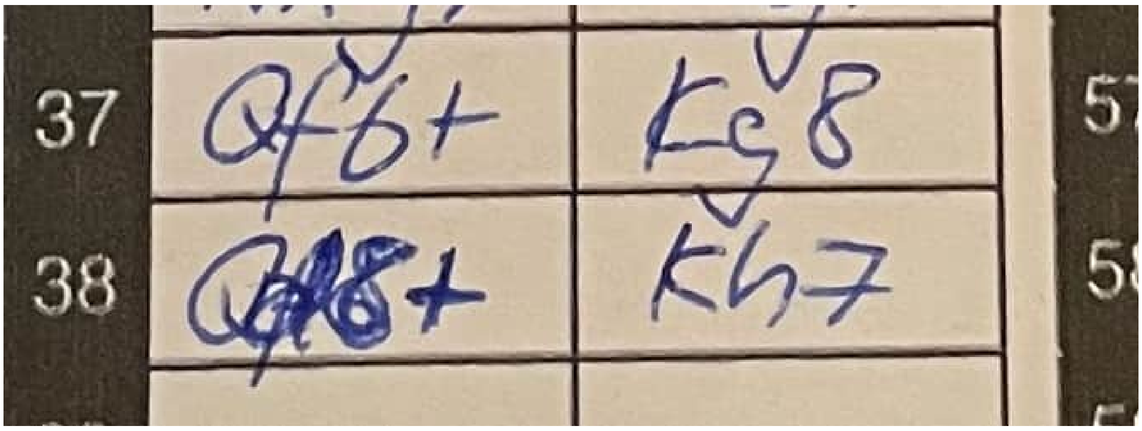


Figure 3.3: Hard to read moves bleeding into multiple cells. It reads: g6, Lg7, O-O. The L and the g merged, so the L is almost unrecognizable

Figure 3.3 is hard to read, not only because of the handwriting but also because the writing bleeds into multiple lines. It reads: g6, Lg7, O-O. It is the continuation of the same game as 3.2. A multitude of problems can be demonstrated with this example. Firstly the move g6 is fairly readable. However the g crosses the line into the next field significantly, therefore when considering only the part of the move that is readable within the field, the move will look more like a6. The second field Lg7, is almost unreadable by itself, not only does the g from the upper cell interfere with this cell, but the L and g of Lg7 also bleed into each other. This time the g looks very different than the previous one and also bleeds into the cell below. The last move of O-O is quite readable again.

At first glance, this move sequence looks like g6, g7, O-O. However, in situations like these, it helps to have a basic understanding of chess. The moves g6, g7 for black are impossible since pawns can only move forwards. Furthermore, the moves of Figure 3.2 and Figure 3.3 were the first 4 moves black played in this game, so castling would also not be allowed after this move order, which means O-O would be illegal. Therefore it is much more likely, that the black player in this game fianchettoed (a popular chess pattern) his king's side bishop as it would happen in a variation of the popular Kings Indian Defense. This would lead to the moves Sf6, g6, Lg7, O-O. In the annotations, all these moves were marked as hard to read, as they would be without any understanding of chess. However, it was still possible to determine what moves were played.

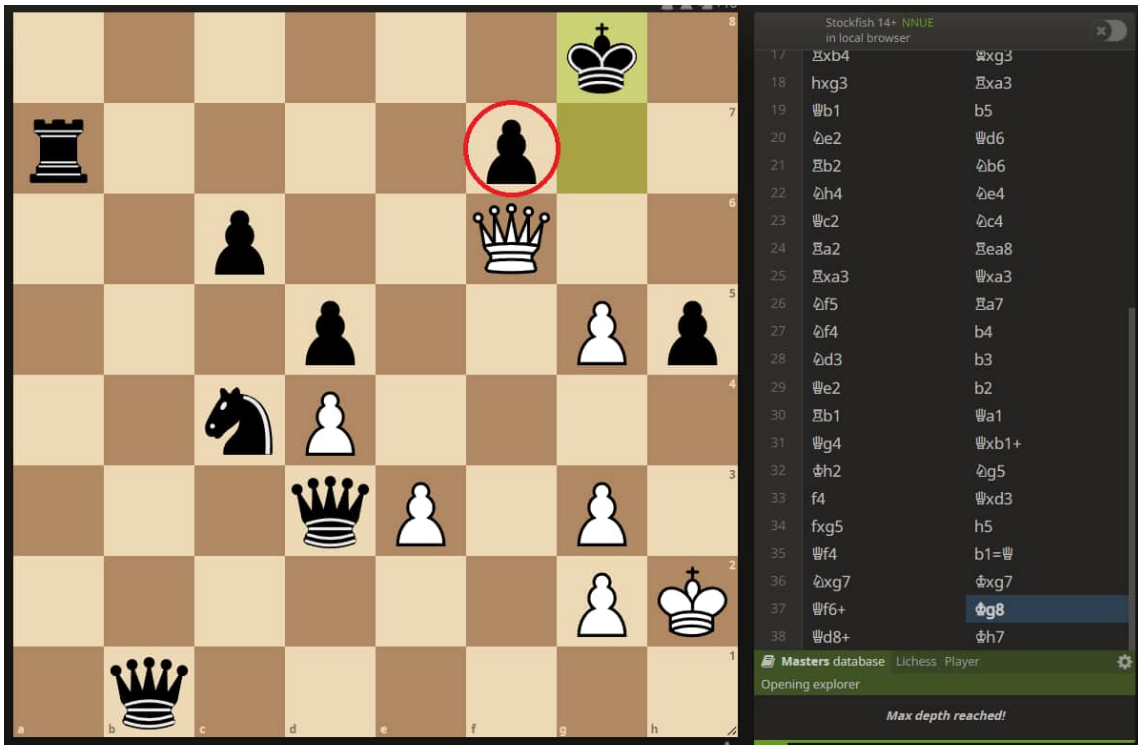
In the following example, a last-ditch effort is shown when it was not possible to determine the contents of a field. This will be done using the sheet "en_0007_medium_38" from the corpus.



37	Qf6+	Kg8
38	Qd8+	Kh7

Figure 3.4: An excerpt from the sheet "en_0007_medium_38" containing an ambiguous move.

In Figure 3.4 it can be seen that move 38 for white is either Qd8+ or Qf8+. This was the last turn of the game so it is not possible to use later information to determine the contents of this field. However, from the previous turn, it is known that the white queen was on f6, and the black king moved to g8. Unfortunately, this does not help determine the move either, as both Qd8+ and Qf8+ would be legal moves, resulting in the black king being checked again, and then Kh7 for black would also be legal for both options. So the last chance was to replay the entire game, to observe if the position on the board would help determine the position.



17	♖xb4	♗xg3
18	hxg3	♖xa3
19	♕b1	b5
20	♘e2	♕d6
21	♖b2	♘b6
22	♘h4	♘e4
23	♕c2	♘c4
24	♖a2	♖ea8
25	♖xa3	♕xa3
26	♘f5	♖a7
27	♘f4	b4
28	♘d3	b3
29	♕e2	b2
30	♖b1	♕a1
31	♕g4	♕xb1+
32	♔h2	♘g5
33	f4	♕xd3
34	fxg5	h5
35	♕f4	b1=♕
36	♘xg7	♔xg7
37	♕f6+	♔g8
38	♕d8+	♔h7

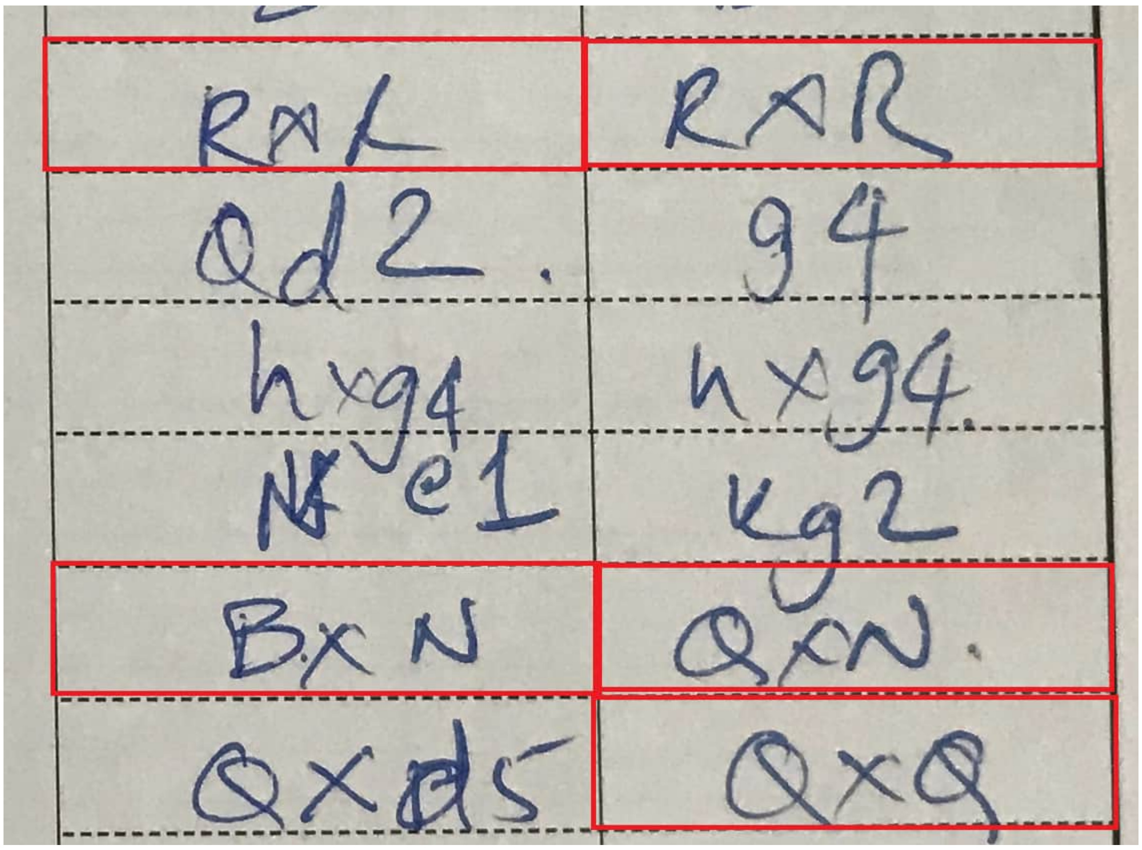
Figure 3.5: Replay of the game from the previous example on the board

After taking a look at the board in Figure 3.5, it can be seen that the black pawn on f7 actually blocks the move Qf8+ for white, which means the move in the game must have been Qd8+. This replay-approach helped in most situations where it was not clear if the correct notation was correct but this process was extremely time-consuming.

To conclude, there can be multiple reasons why a cell could get a "hard to read" annotation. They include the handwriting style of the player, letters bleeding across cell lines, and corrections of moves. The annotator decided when a cell is hard to read based on their own judgment. In any case, the surrounding boxes could provide a hint to which move was made. If that was not possible the game was replayed up to that point to help figure out what the played move was.

Examples that have notation errors

Next, the "has notation error" annotation will be looked at.



RxL	RxR
Qd2.	g4
hxg4	hxg4.
Nfe1	Kg2
BxN	QxN.
Qxd5	QxQ

Figure 3.6: An excerpt from a scoresheet containing multiple instances of a shorthand notation.

Something is considered an error if it does not respect the rules for SAN. Many chess players use shorthands or dialects as in Figure 3.6 above the popular "QxQ" notation. This is still useful to the players in most cases, as the piece that is being taken can be identified uniquely. Meaning for "QxQ" there is usually only one Queen to be taken. This is still recorded as an error in the annotation though since it would be hard to know every version of these shorthands. However, it would be very interesting for a future topic to analyze these errors and find out which of these were actual mistakes during writing and which are just shorthands, as well as how many different shorthand notations exist.

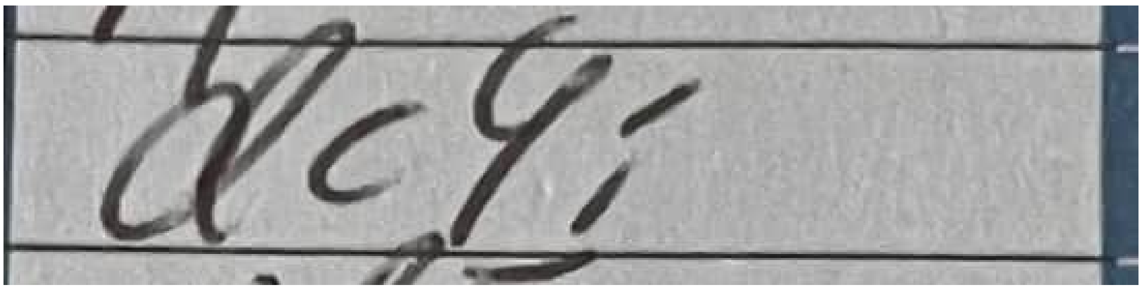
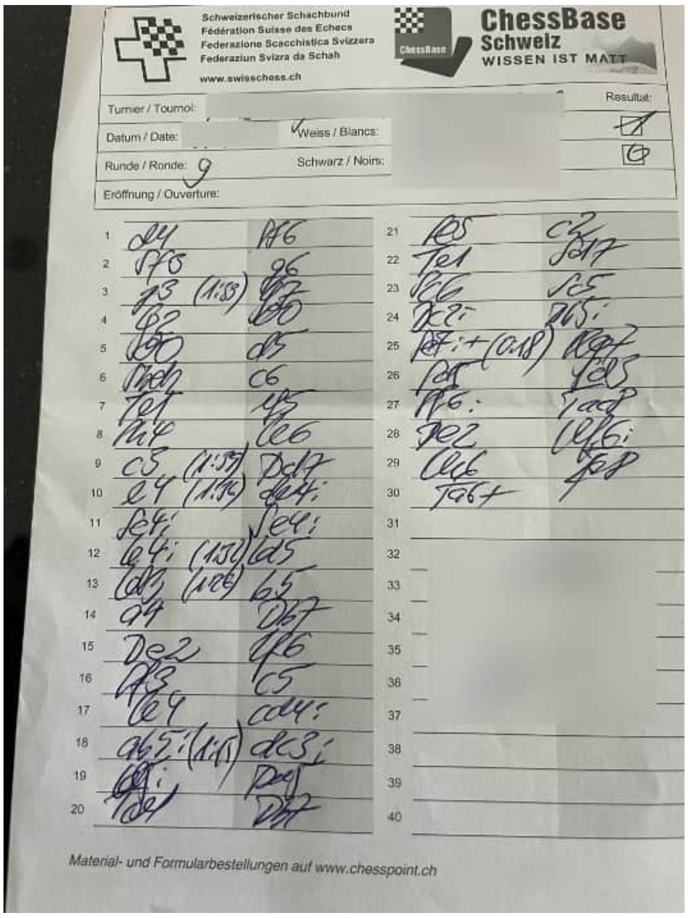


Figure 3.7: Another example of a common notation error is the ; takes notation

A lot of players also write "pawn takes" a little differently than in SAN. Since pawns can only move forwards, if a pawn moves diagonally it always has to have taken another piece. Therefore many players choose to omit the "x" when writing that a pawn has taken, such as in Figure 3.7 writing "dc4" instead of "dxc4". In this player's case they instead also wrote a ";" at the end of the cell, which has the same meaning as the x in SAN. These cases were both considered notation errors in the annotation.



Schweizerischer Schachbund
Fédération Suisse des Echecs
Federazione Scacchistica Svizzera
Federaziun Svizra da Schah
www.swisschess.ch

ChessBase Schweiz
WISSEN IST MATT

Turnier / Tournoi: [redacted]
Datum / Date: [redacted] Weiss / Blancs: [redacted]
Runde / Ronde: 9 Schwarz / Noirs: [redacted]
Eröffnung / Ouverture:

Resultat: 1 0

1	d4	Sf6	21	Le5	c2
2	Sf3	g6	22	Te1	Sd7
3	g3 (1:33)	Lg7	23	Sc6	Sc5
4	Lg2	0-0	24	Dc3;	Db5;
5	0-0	d5	25	Lf7;+ (0:18)	Kg7
6	Sbd2	c6	26	Dc5	Td5
7	Te1	a5	27	Tf6:	Tad8
8	h4	Le6	28	De2	Lf6;
9	c3 (1:37)	Dd7	29	Lc6	Tf8
10	e4 (1:36)	de4;	30	Ta6+	
11	Se4;	Se4;	31		
12	Te4; (1:30)	Ld5	32		
13	Te3 (1:28)	b5	33		
14	a4	Db7	34		
15	De2	Lf6	35		
16	Lf3	c5	36		
17	Le4	cd4;	37		
18	ab5; (1:15)	dc3;	38		
19	[illegible]	[illegible]	39		
20	Td1	Db7	40		

Material- und Formularbestellungen auf www.chesspoint.ch

Figure 3.8: An sheet containing examples of time markings. "de_0056_hard_30" sheet of the corpus.

Other writings in the cells of the scoresheet were ignored, as seen in Figure 3.8, such as markings to the position (ex. +3 or -3) as well as markings recording the time for a turn (ex. [1:25]). They will not be recorded as errors, however, these markings usually make the cell more difficult to read so they will sometimes lead to a "hard to read" annotation.

3.2.2 Metadata Annotation

Everything that is not part of the moves themselves is considered metadata. The following metadata annotations were included:

- **language:** The ISO 639-1 language code. (for example: de/en/it/fr)
- **game length:** Number of moves made in the game.
- **difficulty:** Rating of the processing difficulty of the scoresheet. How these ratings were generated will be discussed in the next chapter in detail.
- **keywords :** A list of the criteria why a game received a given difficulty rating.
- **winner :** Winner of the game, "draw" in case of a draw.
- **number of tables:** Number of tables present on the scoresheet. Some scoresheets include multiple tables on one page.
- **table position:** x-y position of these tables in pixels on the image
- **table size:** the pixel size of the table.
- **box positions:** x-y position in pixels of every box that contains a move, as well as the height and width in pixels.

```
{
  "language":"de",
  "game_length":29,
  "difficulty":"hard",
  "keywords": [
    "incompleteTable",
    "moreThan33Illegible",
    "noLines"],
  "winner":"white",
  "number_of_tables":1,
  "table_position": [{"x":435,"y":723}],
  "table_size":[{"width":2372, "height":2696}],
  "box_positions":[[
        {
            "box_index": 0,
            "height": 76,
            "pos_x": 358,
            "pos_y": 980,
            "width": 196
        }, ... <truncated to save space>
        ]
    ]
}
```

Example of such a JSON file, containing Metadata Annotations.

```
{
  "language":"de",
  "game_length":29,
  "difficulty":"hard",
  "keywords": [
    "moreThan33Illegible",
    "noLines"],
  "winner":"white",
  "number_of_tables":2,
  "table_position": [{"x":435,"y":723},{"x":2435,"y":723}],
  "table_size":[{"width":2001, "height":2696},{"width":372, "height":2696}],
  "box_positions":[[
        {
            "box_index": 0,
            "height": 76,
            "pos_x": 358,
            "pos_y": 980,
            "width": 196
        }, ... <truncated to save space>
        ], ... <truncated to save space>
    ]
}
```

Example of a JSON file with multiple tables on a single sheet.

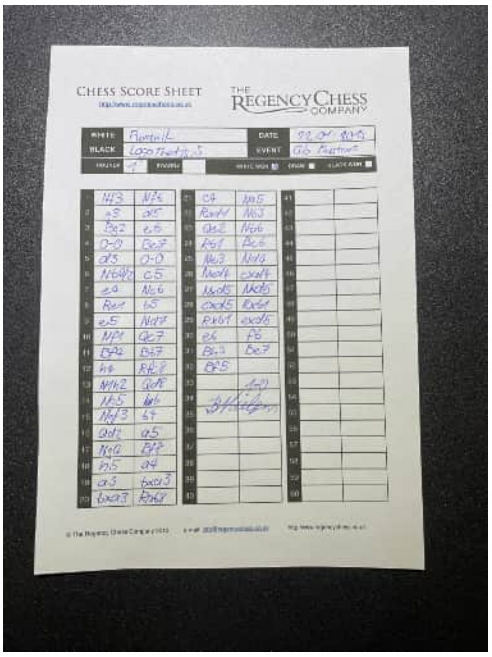
The table position, table size, and box position fields would be unreasonable to do by hand, therefore it was decided to use the outputs of ChessReader to help annotate these fields. The table position and size were however confirmed by hand afterwards. Box positions however will not be checked by hand at all.

Difficulty Annotation

Some sheets are much harder to process than others. In order to reflect this, a difficulty rating was included. A 3-tier system of easy, medium, and hard was decided on. During testing, it was determined the following conditions made sheets more difficult to process:

- Low brightness or contrast of the picture.
- Dark shadows obscuring part of the table.
- Skewed angle of the scoresheet. Small amounts of yaw are not a problem, but especially rotations in pitch or roll make it harder for the OCR to work.
- Incomplete table on the image. For certain table detection algorithms, the whole table needs to be visible in order for the detection to work. This was included to signal that a missing part of the table might make the detection of it more difficult.
- More than 33% of moves are illegible.
- No lines or only dotted or dashed lines used for the table.

If none of these conditions apply, a sheet was rated easy to process. If exactly one of these conditions applies, it was rated medium and if more than one of these applies, it was rated hard. No objective way to determine bad lighting or skewed angles could be defined, instead these aspects rely on the subjective opinion of the annotator. The exception to this is the "33% illegible" condition, which can be checked after having done the move annotations. Additionally, each condition that applies is included as a keyword in the JSON file, so that it is clear why a sheet got a certain difficulty rating.



CHESS SCORE SHEET

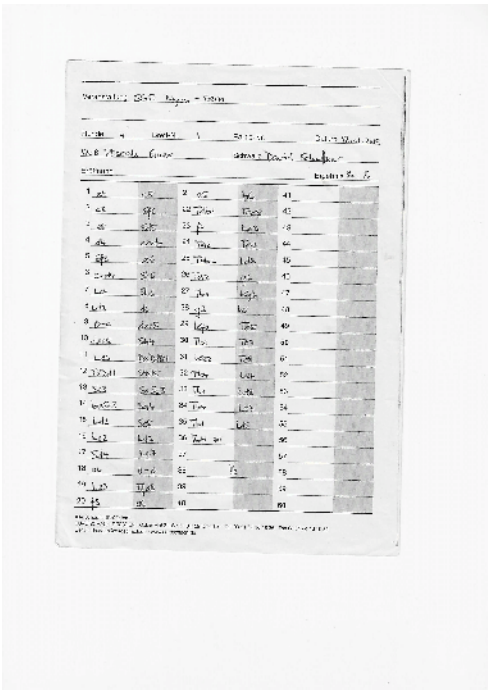
THE REGENCY CHESS COMPANY

WHITE	Pontnik	DATE	22.04.2018
BLACK	Logothetis S.	EVENT	GS Pastures

ROUND 4 BOARD [illegible] WHITE WON [✓] DRAW BLACK WON

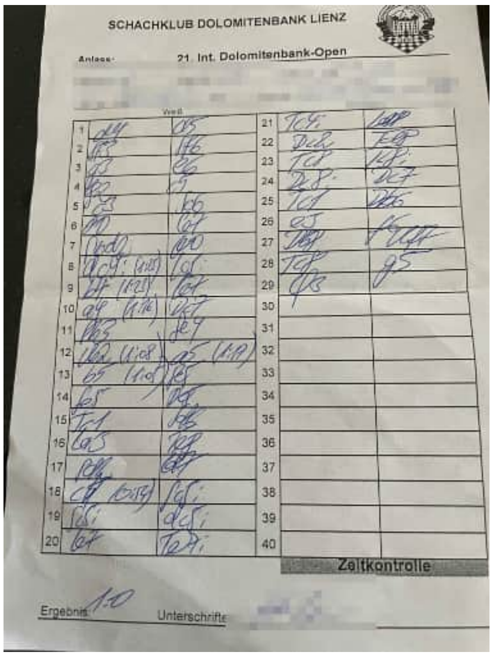
1	Nf3	Nf6	21	c4	Na5	41		
2	g3	d5	22	Rad1	Nc3	42		
3	Bg2	e6	23	Qe2	Nb6	43		
4	O-O	Be7	24	Rb1	Bc6	44		
5	d3	O-O	25	Nc3	Nd4	45		
6	Nbd2	c5	26	Nxd4	cxd4	46		
7	e4	Nc6	27	Nxd5	Nxd5	47		
8	Re1	b5	28	cxd5	Rxb1	48		
9	e5	Nd7	29	Rxb1	exd5	49		
10	Nf1	Qc7	30	e6	f6	50		
11	Bf4	Bb7	31	Bh3	Be7	51		
12	h4	Rfc8	32	Bf5		52		
13	N1h2	Qd8	33		1-0	53		
14	Ng5	Nb6	34	[illegible]		54		
15	Nhf3	b4	35			55		
16	Qd2	a5	36			56		
17	Nh2	Bf8	37			57		
18	h5	a4	38			58		
19	a3	bxa3	39			59		
20	bxa3	Rab8	40			60		

© The Regency Chess Company 2015 e-mail [illegible] http://www.regencychess.co.uk



[illegible]

(a) easy, no bad conditions (b) medium, No vertical lines for the table



SCHACHKLUB DOLOMITENBANK LIENZ

Anlass: 21. Int. Dolomitenbank-Open

[redacted]

Weiß

1	d4	d5	21	Tc4	[illegible]
2	Sf3	Sf6	22	Dc2	[illegible]
3	e3	e6	23	Tc1	[illegible]
4	[illegible]	c5	24	[illegible]	Dc7
5	c3	b6	25	Tc1	[illegible]
6	[illegible]	Ld6	26	e5	[illegible]
7	[illegible]	0-0	27	[illegible]	[illegible]
8	[illegible] (1:08)	Te8	28	Tc8	g5
9	[illegible] (1:25)	[illegible]	29	[illegible]	
10	a4 (1:16)	[illegible]	30		
11	[illegible]	Le7	31		
12	[illegible] (1:08)	a5 (1:19)	32		
13	b5 (1:05)	[illegible]	33		
14	[illegible]	[illegible]	34		
15	Tc1	[illegible]	35		
16	La3	[illegible]	36		
17	[illegible]	[illegible]	37		
18	[illegible] (0:54)	[illegible]	38		
19	[illegible]	dxc5	39		
20	[illegible]	[illegible]	40		

Zeitkontrolle

Ergebnis 1:0 Unterschrifte [redacted]

(c) hard, incomplete table, 33% illegible

Figure 3.9: Examples of all 3 tiers of difficulty and explanation for the rating

Chapter 4

Corpus Construction

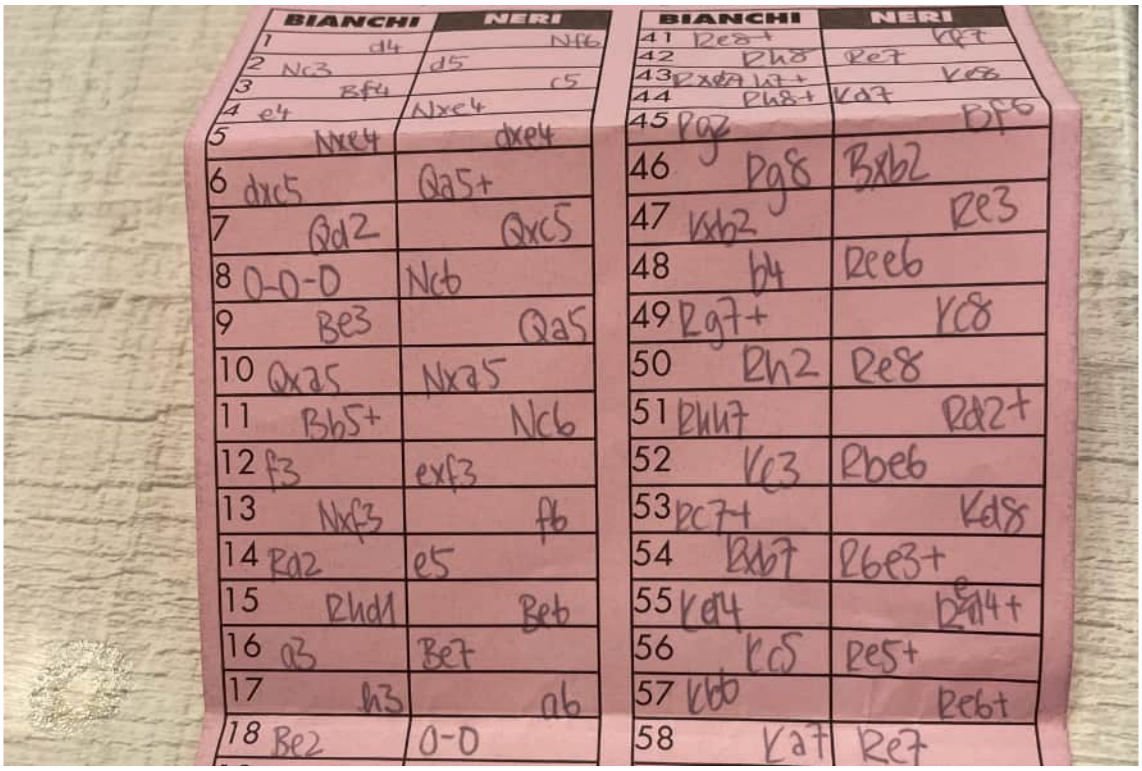
After describing how the annotations were made in the last chapter, in this chapter the construction of the corpus itself is documented. The methods used to collect and annotate the sheets are described and the changes made to the ChessReader application are addressed as well.

4.1 Corpus Construction Process

In the first step, the scoresheet used in previous theses on the topic[4] were collected. Those consisted of both, games that were transcribed specifically for this purpose, as well as scoresheets from live chess games. They make up the bulk of english scoresheets used in the corpus.

A second source was the scoresheets provided by Gundula Heinatz, who was able to use her connections to the Swiss chess community to provide us with a variety of sheets from chess tournaments, containing different styles of sheets as well as English, German, and French sheets. Some images were obtained directly through her, meaning she and her colleagues photographed them in the same way as they would to use the ChessReader tool. A collection of physical sheets was also obtained, which were then digitized manually, using both camera pictures and scans to represent different possible options. This gave us a broad foundation of languages and image types to be used for the corpus.

A group of images was excluded due to them not containing all the necessary information for proper processing. Meaning there were several images where part of the game was obscured.



	BIANCHI	NERI		BIANCHI	NERI
1	d4	Nf6	41	Re8+	Kf7
2	Nc3	d5	42	Rh8	Re7
3	Bf4	c5	43	Rxd7 h7+	Ke6
4	e4	Nxe4	44	Rh8+	Kd7
5	Nxe4	dxe4	45	Rg2	Bf6
6	dxc5	Qa5+	46	Rg8	Bxb2
7	Qd2	Qxc5	47	Kxb2	Re3
8	0-0-0	Nc6	48	b4	Ree6
9	Be3	Qa5	49	Rg7+	Kc8
10	Qxa5	Nxa5	50	Rh2	Re8
11	Bb5+	Nc6	51	Rhh7	Rd2+
12	f3	exf3	52	Kc3	Rbe6
13	Nxf3	f6	53	Rc7+	Kd8
14	Ra2	e5	54	Rxb7	R6e3+
15	Rhd1	Be6	55	Kd4	R$\overset{e}{d}$4+
16	a3	Be7	56	Kc5	Re5+
17	h3	a6	57	Kb6	Re6+
18	Be2	0-0	58	Ka7	Re7

Figure 4.1: Example of an image that was excluded

In the next step, any identifying information on the images was blurred, where there was such information. Part of the goal was to not preprocess the images contained in the corpus too much, as for example cropping the image to contain only the table, like it had been done in the HCS dataset, would remove part of the difficulty in processing these images.

Next, each move was annotated by hand as explained in chapter 3. Afterwards metadata was added also by hand where necessary, consisting of which side won the game, etc., and then used the reworked ChessReader tool to help annotate the table and box position.

Finally, a Creative Commons Attribution 4.0 International (CC BY 4.0) license was added to the main folder of the corpus. This will give others the right to use and modify all the contents created during this thesis as they please, for both commercial and non-commercial uses.

Before looking at the results of this process, the changes made to ChessReader, which helped with annotating the sheets, will now be discussed in further detail.

4.2 Server-side implementation of ChessReader

When trying to parse a scoresheet, as they are in the corpus, the table and its fields need to be extracted, before an OCR-algorithm can be applied. By doing that, the different moves can be separated and enumerated. Since this is non-trivial as well, it was decided to not only include the moves that were recognized but also their approximate position on the image. This would be extremely tedious to do by hand for this many scoresheets, so the coordinates are not verified by hand. Instead, the ChessReader application that was developed in multiple theses prior to this work was adapted to allow the extraction of the necessary data automatically. To achieve these changes to how data was handled, saved, and transmitted by the server-side part of the application had to be made. Up to now multiple teams had worked on its source code, and the quality of the code had clearly suffered from this. During the previous semester thesis [12], work started on untangling the different pieces and decoupling the client-side from the rest of the application. The old source code lived on and served as a temporary "API" to the frontend.

This was clearly not the way to go forward, since data had to be extracted out of strings using complicated and error-prone algorithms. The different paths of the old project returned vastly different values, with seemingly no pattern. It quickly became apparent, that in order to be able to extract the relevant data from the database, a structured interface needed to be designed, so this task can be automated easily and properly. The solution to this problem was a JSON-based REST-API as a wrapper around the core functionality of the previous application.

4.2.1 REST-API

To build this API a library called "FLASK-restful" was used. Its main building blocks are resources, which represent a single endpoint each, handling one type of data.

```python
class MatchResource(Resource):
    @jwt_required()
    def get(self, match_id):
        id = get_jwt_identity()
        match = ChessMatch.query.get_or_404(match_id)

        # Check if current user is allowed to perform this action
        if match.user_id != int(id):
            return None, 404
        return chess_match_schema.dump(match)

    @jwt_required()
    def delete(self, match_id):
        id = get_jwt_identity()
        match = ChessMatch.query.get_or_404(match_id)

        # Check if current user is allowed to perform this action
        if match.user_id != int(id):
            return None, 404
        db.session.delete(match)
        db.session.commit()
        return None, 204
```

Listing 4.1: The code for the MatchResource

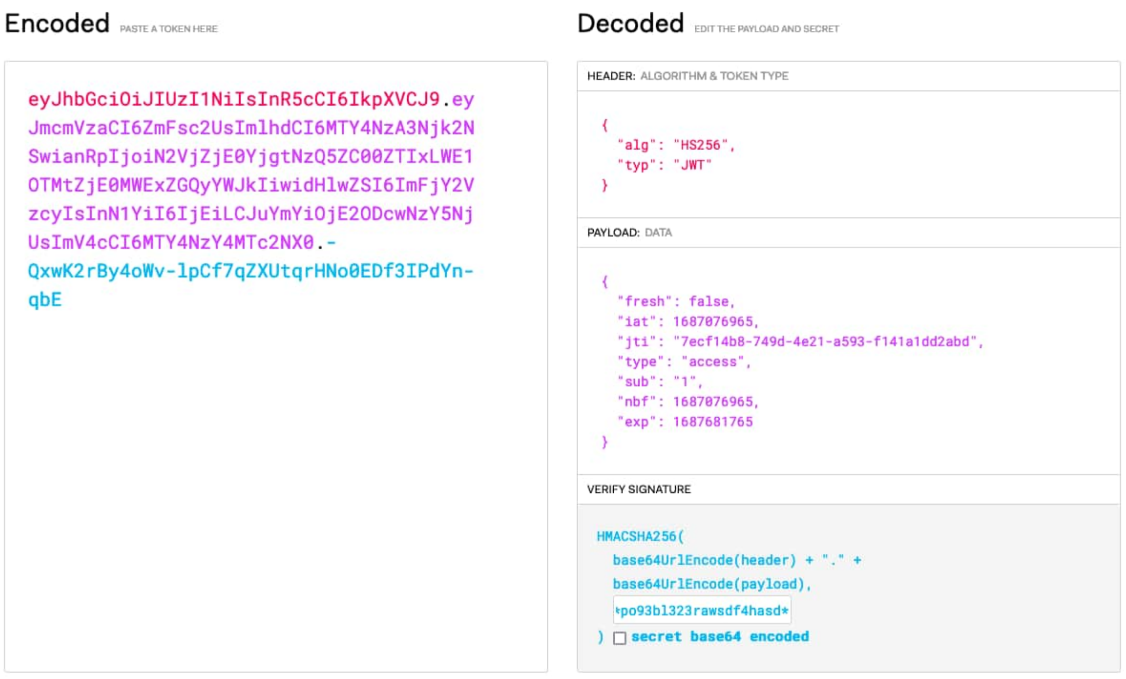
The code segment in Listing 4.1 shows the MatchResource as an example. It can be seen that the resource itself is represented by a Python class, which is in turn a child of the Resource class, as defined by FLASK-restful. The resource only deals with matches that already exist on the database, and allows clients to either retrieve a match from the database or delete it using the respective Hypertext Transfer Protocol (HTTP) methods.

If a feature needs to be introduced that updates the entire match data at once, another allowed HTTP method would be added to this resource, for example a PATCH-method, allowing clients to patch new data into the database. But if only part of the match data, for example the user it is associated with, would be updated, it would be better to create a dedicated resource and endpoint for that in order to not introduce inconsistency in regards to the data scope affected by a request.

The documentation of all the API endpoints created can be found in Appendix B.

Authentication

To handle most requests, the server needs to know which user currently accesses the API. This needs to be done to serve data associated with the account, but also to verify that they are in fact allowed to modify or view the data they are accessing. A naive implementation would just rely on the client to tell the server which user is logged in, which is a terrible idea from a security standpoint. Instead, the server should never trust the client and always assume malicious intent. So if the client can not be trusted, how is it possible to know which user is making a request?



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JmcmVzaCI6ZmFsc2UsImlhdCI6MTY4NzA3Njk2N
SwianRpIjoiN2VjZjE0YjgtNzQ5ZC00ZTIxLWE1
OTMtZjE0MWExZGQyYWJkIiwidHlwZSI6ImFjY2V
zcyIsInN1YiI6IjEiLCJuYmYiOjE2ODcwNzY5Nj
UsImV4cCI6MTY4NzY4MTc2NX0.-
QxwK2rBy4oWv-lpCf7qZXUtqrHNo0EDf3IPdYn-
qbE
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "fresh": false,
  "iat": 1687076965,
  "jti": "7ecf14b8-749d-4e21-a593-f141a1dd2abd",
  "type": "access",
  "sub": "1",
  "nbf": 1687076965,
  "exp": 1687681765
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  *po93bl323rawsdf4hasd*
) ☐ secret base64 encoded
```

Figure 4.2: Example of a JSON Web Token, encrypted and decrypted. The ID of the user is stored in "sub", which can be seen on the middle right

This is where so-called JSON Web Token (JWT) come into play. A JWT is also a JSON-object, in this case a list of key-value pairs, that is issued by the server to the client and can be used to identify a user. The server provides a login endpoint where, upon successful verification of the provided credentials, it generates a personal JWT with the users internal ID, along with other properties like a validity duration, and encrypts it using a secret. This token is stored by the client and attached to each request made to the server. When a request comes in, the server looks for this token, decrypts and verifies it before the request is allowed to continue.

If a request is not authenticated, the token is no longer valid, or the user is not allowed to access a certain resource, best practice guidelines are followed and as little information as possible about the data that was tried to be accessed is provided. In the first two cases, the server responds with the status that the authentication failed, while in the third case, it returns that the resource does not exist, even if it might.

4.2.2 Core functionality

The core functionality of the server-side application was largely transferred from the old ChessReader application.

This functionality includes alignment of the images, as well as table recognition and the actual text extraction using OCR. This part of the application was modified as little as possible since rewriting the code completely was not the focus of this bachelor thesis. The result of this approach is that the old "legacy" code is treated as a black box, with only a few contact points to the new code. This separation should also make it easier to replace the old code at a later date.

But in order to accomplish the goal of using ChessReader to help with the annotations, it was still necessary to make some changes to the code, dive a bit into the "legacy" code, and make small adjustments to it. One example is how box detection and confirmation worked before. In the old application, the detected boxes were not saved to the database. The results of the detection were directly used to apply OCR to the image and then discarded.

This was obviously a problem since extracting the boxes from the database was the main reason why ChessReader should be used for this thesis in the first place. Therefore the application had to be modified so that the data was actually saved for later.

4.2.3 Database

With the changes made to the responses and what data was being saved, it was clear that the old database scheme was no longer a good fit for the application. So a new database layout had to be designed. The new layout tried to lean on work that had been done already but also ensures that future teams working on the project will not have a hard time expanding on it.

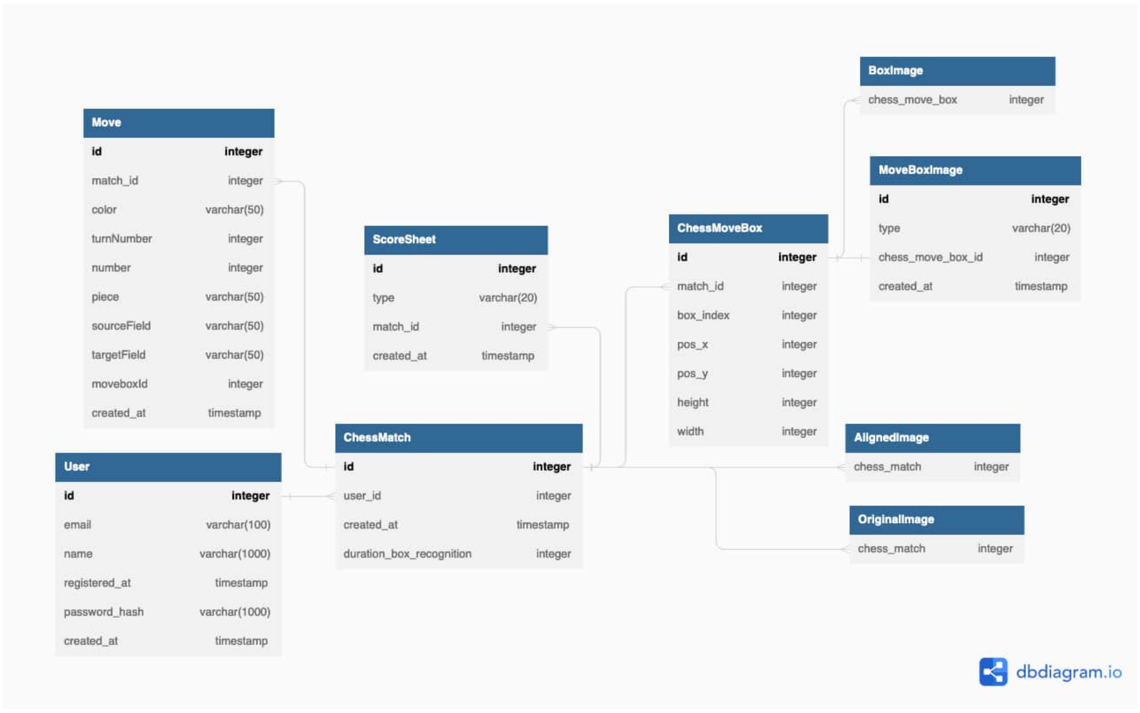


Figure 4.3: Diagram of the database structure

In Figure 4.3 the updated database is shown. The tables AlignedImage, BoxImage, and OriginalImage are shown for completeness' sake. They are entirely handled by the SQLAlchemy-ImageAttach library and are automatically generated.

The table that is most relevant for this thesis, and the one that did not exist before, is the ChessMoveBox table. Here, the position, height, width, and number for ordering of each box are stored, which can then be queried to be used in the corpus. The entries of this table have a relation to the ChessMatch table using the match_id which links almost all relevant data to a user.

Another important table is the Move table, which stores the recognized texts. Currently, the fields "sourceField" and "piece" are unused. Initially, an idea was to fill all fields with the appropriate values but this had to be abandoned since in order to store the source field of a move the entire match had to be replayed. This would clearly not have fit into the scope of this thesis, but the fields were left in, so future teams can implement this feature.

4.3 Client-side implementation of ChessReader

After rebuilding the server-side of ChessReader, including restructuring the database, minor changes to the frontend of the application had to be made.

Most changes were made to the file responsible for making requests to the server. The client-side application written during the previous semester thesis [12], still relied on the old ChessReader application as an "API". Some data was retrieved normally as JSON, other data was baked into script-tags in HTML files or snippets that were returned by endpoints. This meant that the application had to rely on complex and error-prone regex-statements such as the ones shown below in Listing 4.2 to extract the data. With the rewritten API-interface it was possible to like this.

```
alignedImageRegex = new RegExp('src="(media\/[a-z0-9\/.-]+)')
boxImageRegex = new RegExp('src="(.*)\?')
ocrRegex = new RegExp('<script>.*", (".*").*<\/script>')
```

Listing 4.2: Regex-statements for extracing image-, box- and OCR-data from HTML files

Another improvement that was made, now that the server was keeping track of the boxes of each match, was to remove the code responsible for saving that data locally in the browser. Aside from obvious benefits like lower memory consumption, this also led to a much simpler dataflow in the client-application, making it easier to understand and maintain. A class that was removed entirely thanks to this optimized handling of data can be seen in Listing 4.3

```
export class MoveService {

  lastMatchId: number = -1

  private current$ = new BehaviorSubject<OcrResponse | null>(null)
  public current = this.current$.pipe(
    filter((it: OcrResponse | null): it is OcrResponse => it != null)
  )

  constructor(
    private api: ChessreaderApiService,
    private spinner: SpinnerService
  ) { }

  nextMoves(matchId: number, selectedBoxes: RecognizeMovesRequest) {
    this.lastMatchId = matchId
    this.api.recognizeMoves(matchId, selectedBoxes).pipe(observable =>
      withSpinner(this.spinner, observable)
    ).subscribe(it => {
      this.current$.next(it)
    })
  }
}
```

Listing 4.3: Service for storing the OCR-response of the server while the client was transitioning to the review stage.

This service was required when the frontend relied on the old ChessReader application as the "API" because the server was only able to fulfill the OCR-request when the coordinates of all boxes were transmitted. In the review stage of the client however, this data is not available due to the review page and the box-selection page being two completely different components. So either the design philosophy of keeping components separate could be violated or a service that would keep the OCR-data in memory on the client would need to be created.

Some other small features were also implemented, like the ability to sign up and log in. All matches have to be associated with a user and each endpoint requires authentication to be accessed. A very simple component was created for that, where users can input their credentials, after which the client requests a JWT token from the server (see chapter 4.2.1 section Authentication). A very small helper service then attaches this token to each request automatically and handles exceptions when the server rejects the authorization.

Chapter 5

Results

After the changes to the ChessReader tool and the construction of the corpus were completed, analyzing the corpus in terms of all the criteria annotated is next. This chapter contains the results of that analysis.

After collecting 95 images from various sources 64 images of scorecards were included in the corpus. The images that were not used, were mostly cut because they were either so poorly lit that they were unreadable, or in some cases, a part of the cells was not on the picture. Some also were not included in the corpus due to there being already enough of a certain type (ex. easy English) sheets represented.

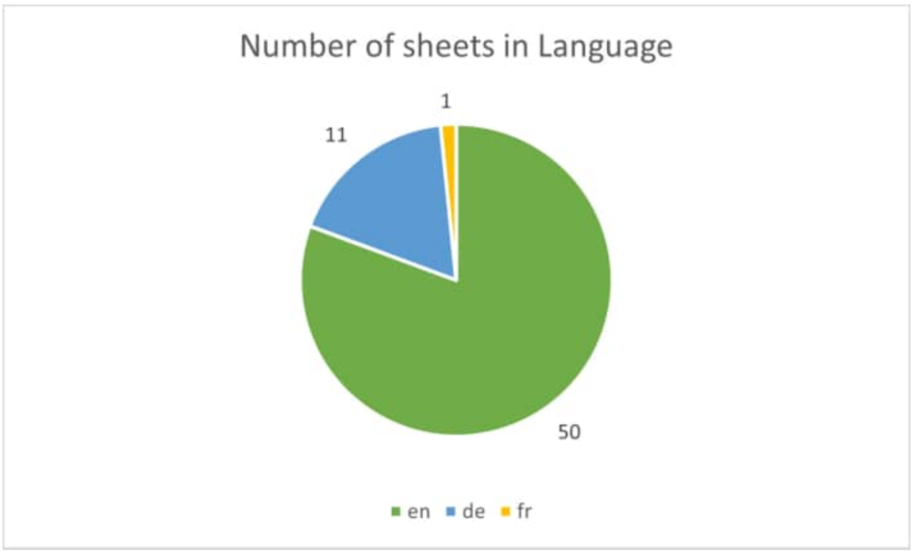


Figure 5.1: Languages represented in the corpus.

As can be seen in Figure 5.1 the majority of the sheets were English SAN. Two scoresheets were spread over two images, therefore the number of sheets ends up being 62.

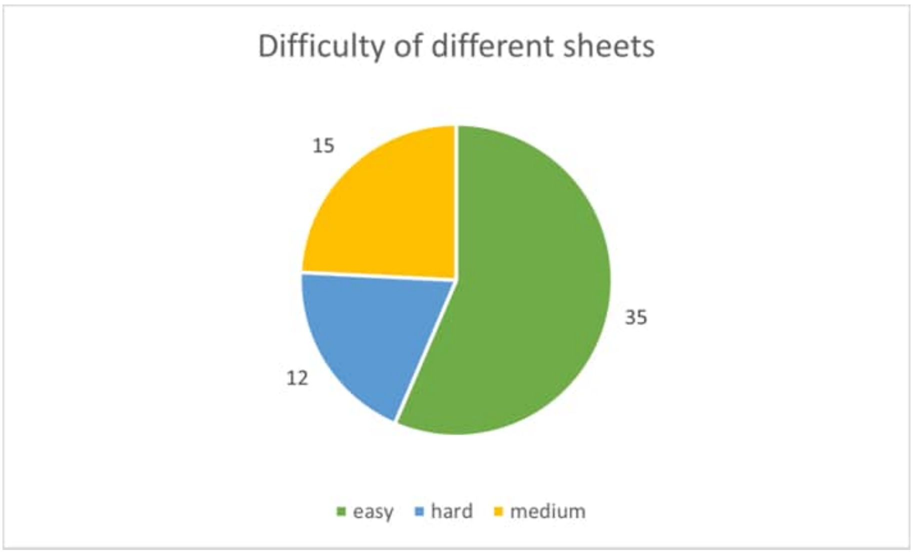


Figure 5.2: Number of sheets per difficulty rating.

Difficulty	**english**	**german**	**french**	**Total**
easy	30	5	0	35
medium	13	1	1	15
hard	7	5	0	12

Table 5.1: Sheet difficulty per language.

Figure 5.2 shows the difficulty of the sheets represented, with the distribution across the languages shown in table 5.1.

languages	**low contrast**	**dark shadows**	**skewed angle**	**incomplete table**	**more than 33% illegible**	**no lines**	**average game length**
en	16	7	4	0	0	0	31.92
de	0	0	1	1	4	5	36.36
fr	0	0	0	0	0	1	74
Total	16	7	5	1	4	6	33.39

Table 5.2: Metadata difficulties per language.

Table 5.2 provides a quick overview of the most common issues that lead to harder difficulty ratings for every language. There are clear gaps in the corpus that still need to be covered to provide a better-balanced distribution.

Overall 4140 moves were annotated with 8.5% of them being classified as hard to read and 1.8% containing a notation error.

Chapter 6

Discussion

This chapter will compare the results from Chapter 5 to the criteria outlined in Chapter 2.4 and how well they were met. Afterwards the lessons learned during this Bachelor thesis and what the next steps are in terms of ChessReader are discussed.

6.1 Goals achieved

To reiterate, the following criteria were set for the corpus:

1. Include games from different languages, game lengths, image quality, writing quality, etc. to arrive at a representative sample.
2. Create a clear outline as to what criteria will be recorded and how they will be annotated.
3. Make sure that the images and scorecards included mirror real-world examples and problems, not just artificially created, "good" examples.
4. Include criteria and annotations that will support other future tasks and research as well.
5. Every image and its metadata should have a unique identifier that links them together.
6. The corpus should be published in a way to be accessible and contain a public usage license.

The corpus includes mostly English and German sheets because they were easily obtainable. While this is better than having just a single language represented, it is definitely not as broad as outlined at the beginning of this thesis. A somewhat even spread across all difficulties within the languages was achieved, but there are still clear gaps. Currently, there are way fewer hard English and medium German sheets than needed for a balanced corpus. The results also showed that the reasons for the difficulty ratings vary too much between languages. For English sheets, lighting was most often the cause for a hard difficulty rating, while for German sheets the legibility and missing lines on the scoresheets were a more common cause for harder difficulty ratings. It was ultimately underestimated how difficult it would be to find scoresheets that met all these different criteria, as well as the time it would take to annotate them.

With chapter 3 being entirely devoted to describing the corpus, the second criterion was met to an acceptable standard. There was certainly a trade-off between trying to include information as detailed as possible while also maintaining a clear way of defining how annotations are done.

The majority of the images included were taken by the players that wrote the scoresheets. In order to maintain a better balance for lighting reasons a couple of images were also taken by the annotators. In some cases scanners were used, leading to a quality potentially much higher than it would be if someone took a picture at a chess tournament using a phone camera. However, it is not unimaginable that someone would use a scanner to be able to digitize these scoresheets using a better picture quality. Some examples were excluded as they were so hard to read that a good portion of the game would have just been marked illegible. These types of examples however would not have been of much use anyways. Therefore this criterion has also been met to a sufficient standard.

As the focus was on creating a dataset for testing the ChessReader application, the fourth criterion was definitely neglected somewhat. A good effort was made to include all the information needed for the corpus to be used to improve digitization tasks. However, there are still uses for other tasks, such as table extraction algorithms or possibly analysis of chess shorthand usage across different languages. While nothing was specifically excluded because it would not be relevant to digitization, nothing was specifically included that could be used to expand the usefulness to other areas.

An ID system that would be unique for each image in the corpus was used, such that any metadata or image could be identified by its unique ID. The outlined naming scheme also made sure that a piece of metadata, meaning the CSV file or the JSON files could always be related to the original image. Since the filename included language and game length as well, it would also be possible to notice discrepancies in case an original file gets mislabeled. Therefore this criterion is met.

A creative commons license was added to the corpus, which was the best way to make all the data generated available to the public. Since any identifying information was removed from all the pictures as well, there are no privacy concerns apart from someone being able to match the handwriting directly, which is something that could not be prevented without losing the real-world nature of the corpus.

6.2 Lessons learned

One main take-away is how difficult defining a corpus can actually be. There have been many discussions during this thesis on how to structure files and what exactly the definition of "hard to read" is. Eventually, it had to be accepted that sometimes there is no one "right" way to do things and that it was instead important to document how and why something was done so that others can verify the work. In the end, a good starting point for further expansion was reached. The corpus would be usable for the evaluation of ChessReader and similar tools.
This challenge in being consistent despite some properties being a "subjective" evaluation led to the decision, that only one person was going to annotate all the sheets. On one hand, this made sense because it prevented slightly different interpretations from affecting the outcomes of the annotations, but it also hugely decreased the speed at which the corpus progressed. For the future, a larger time frame for notation would need to be allocated.

Another underestimation at first was the difficulty of dealing with legacy code, that was not at all or not correctly documented. The original approach of trying to wrap the existing code in a new interface was harder than expected. No big rewrites were made, but trying to get access to the data needed took more time than planned and was far more complex than expected. The most time was spent reading through the code in order to figure out how it worked and how to avoid major rewrites.

Chapter 7

Outlook

Now that there is an established starting point for the corpus, the next steps in this area would be to expand on it, to include more languages and scoresheets to make the corpus more representative of the chess world at large.

The next thing to do could be to actually test how well ChessReader in its current iteration performs on the corpus and figure out how the tool could be improved based on the findings. For example, the outputs of the different APIs could be processed in a more sophisticated way to try and improve accuracy or efficiency.

After taking a look at the application that was modified during the thesis, there are a few things that can and should be done before ChessReader is officially released to the public. On the one hand, the user experience on the client can be further improved by designing and testing the interface to help users navigate the website and efficiently add new scoresheets, for example by uploading them to the server in the background instead of blocking the entire client until the upload has completed. Another feature that would be incredibly useful is a way to see, edit and redownload PGNs for games that have already been uploaded.

Taking a look at the server-side part of ChessReader it is apparent that there is a lot of work to do when it comes to cleaning up the legacy code left over from previous works. It lacks accurate documentation and has grown organically during the development by several different teams. In order to improve maintainability and to make it easier to add new features to the application, a rewrite should have high priority. This would also enable future improvements to the speed and accuracy of OCR and table recognition processes, which take quite some time at the moment. This part of the code would also need to be modified to add more recognizable languages to ChessReader.

Another interesting topic to research could be the data the users generate themselves. Of course, first research must be done if it would be legal to use this data for analysis, but it could provide insights into how the application is used, but also how exactly the commercially available OCR-tools from Google, Microsoft, and Amazon perform on the different handwriting styles in a setting that does not consist of real sentences or words.

Appendix A

Technical documentation

In this chapter, the application itself is described, the process of how to run and develop the application locally, as well as how to deploy a new release.

A.1 Language and framework

As the existing application was written in Python using the Flask-Framework, the decision was made to stick with this decision. Enabled the reuse of existing core functionality and set the focus on the front-facing interface and endpoints.

Besides that, Python also has the advantage, that it is generally fairly easy to understand and learn, which makes it easier for future developers to understand the code and develop new features. It is also very popular when it comes to scripts and other smaller projects, so it is a widely known language a lot of developers should already have come in contact with.

When it came to frameworks there was the option to switch to the other popular Python web framework, Django, or to stick with the solution used up to now, Flask.

Django comes with many features already built in, like caching, authentication, Really Simple Syndication (RSS), etc. It also supports rate-limiting and internationalization out of the box, which is very useful when building big projects with international reach.

Flask on the other hand is much more lightweight, and its functionality is mostly added using libraries and extensions. This makes it much more flexible and great for smaller projects and gives the developer the freedom to use any other technologies they want, like NoSQL-databases.

In the end, the decision was made to stick with Flask. This was mainly due to the fact that a complete rewrite of the whole application from scratch was not the focus of this thesis, but also because Flask supports so-called RESTful services.

A.2 Installation prerequisites

- Root privileges
- Docker
- Python development environment (preferably a virtual environment)
- Angular development environment

A.3 System architecture

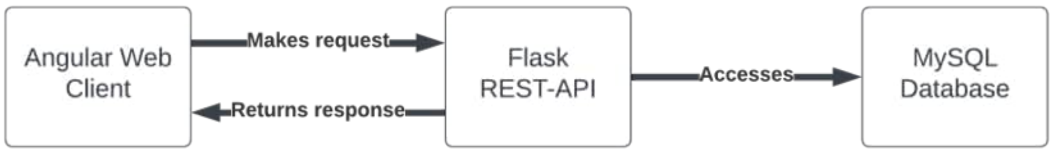


Figure A.1: Current system architecture of ChessReader

- Angular Application
- Flask Backend
- Database

A.4 Local installation

A.4.1 Frontend

First, clone the repository from https://github.zhaw.ch/chessreader/ChessReader_Angular

After having downloaded the repository, install the dependencies of the project using the Numpy package manager and the downloaded package.json file.

A.4.2 Backend

First, clone the repository from https://github.zhaw.ch/chessreader/ChessReader-API

After having downloaded the repository, create a new virtual environment and install the dependencies using the pip package manager and the provided requirements.txt.

In addition to setting the environment variables using the OCR service credentials, as explained below in the subsection OCR-APIs, you will may also need to make adjustments to the values in the provided config file.

```python
import os

basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):
    SQLALCHEMY_DATABASE_URI = os.getenv("DATABASE")
    JWT_SECRET_KEY = os.getenv("JWT_SECRET_KEY")
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    APP_FOLDER = os.path.dirname(__file__)
    UPLOAD_FOLDER = f"{os.path.dirname(__file__)}/media"
    DOWNLOAD_FOLDER = f"{os.path.dirname(__file__)}/media"
    BASE_URL = os.getenv('BASE_URL')
    API_CALLS_TIMEOUT = 18
```

Listing A.1: The config file

You can leave most values as they are. The only fields you need to adjust are the ones that are set using environment variables. You can either hardcode the values using this file or set them using environment variables. Changing the file itself is not recommended, use environment variables instead. The values you need to set are explained below:

The SQLALCHEMY_DATABASE_URI is the Uniform Resource Identifier (URI) to the database on the database server you have set up. If you are using a MySQL server the URI will look something like this:

```
mysql://user:password@ip:port/database
```

SQLAlchemy also supports other servers. To check which are supported and how to structure the URI check the SQLAlchemy documentation[13]

The JWT_SECRET_KEY is the secret used to create new JWT tokens when a user logs in. This should be a random string of letters and numbers in order to make it difficult for malicious actors to create fake tokens by guessing the key.

BASE_URL is the base Uniform Resource Locator (URL) of your API. This includes both, domain and port number, but not any paths, and is required for getting the images of the scoresheets to the client. Also, make sure it does not end with a slash. It should look like this:

```
https://chessreader.org:5000
```

A.4.3 Database

If you want to use a local database, installing Docker for your operating system (https://www.docker.com/) and downloading a MySQL-image is recommended.

If you prefer a different flavor than MySQL, any SQL-database will work as long, as you configure the protocol in the backend config file accordingly.

A.4.4 OCR-APIs

The following sections explain how to obtain the credentials required to call the different OCR-APIs created by Microsoft, AWS, and Google.

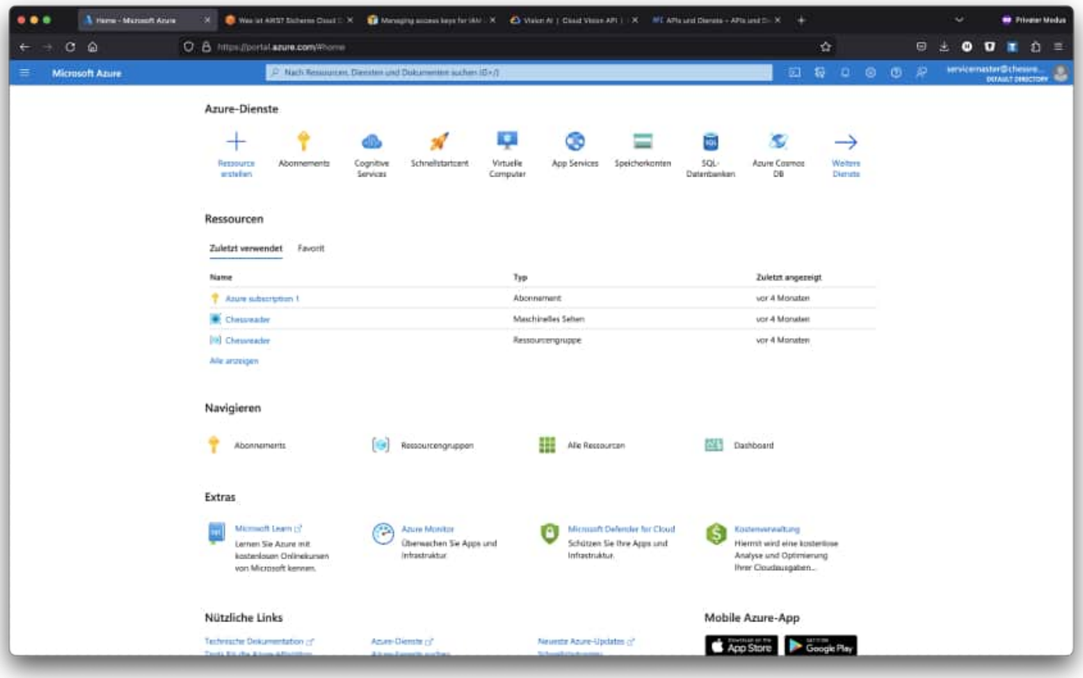
If you have control of the ChessReader accounts for these services already, you can log in using the provided credentials. Otherwise, you can also create a new account, although this is not recommended for maintainability reasons

The credentials you obtained by registering the application with the services listed below need to be saved to environment variables. The name of the variable is specified for each service below

Keep in mind that the exact layout and process of each service page may change if the respective tool is updated.

Microsoft Azure Cognitive Services

To get API credentials from Microsoft Azure you need to first log into Azure Portal (https://portal.azure.com). When you see the screen as shown below, click on "Add Resource".



Name	Typ	Zuletzt angezeigt
Azure subscription 1	Abonnement	vor 4 Monaten
Chessreader	Maschinelles Sehen	vor 4 Monaten
Chessreader	Ressourcengruppe	vor 4 Monaten

Microsoft Learn: Lernen Sie Azure mit kostenlosen Onlinekursen von Microsoft kennen.

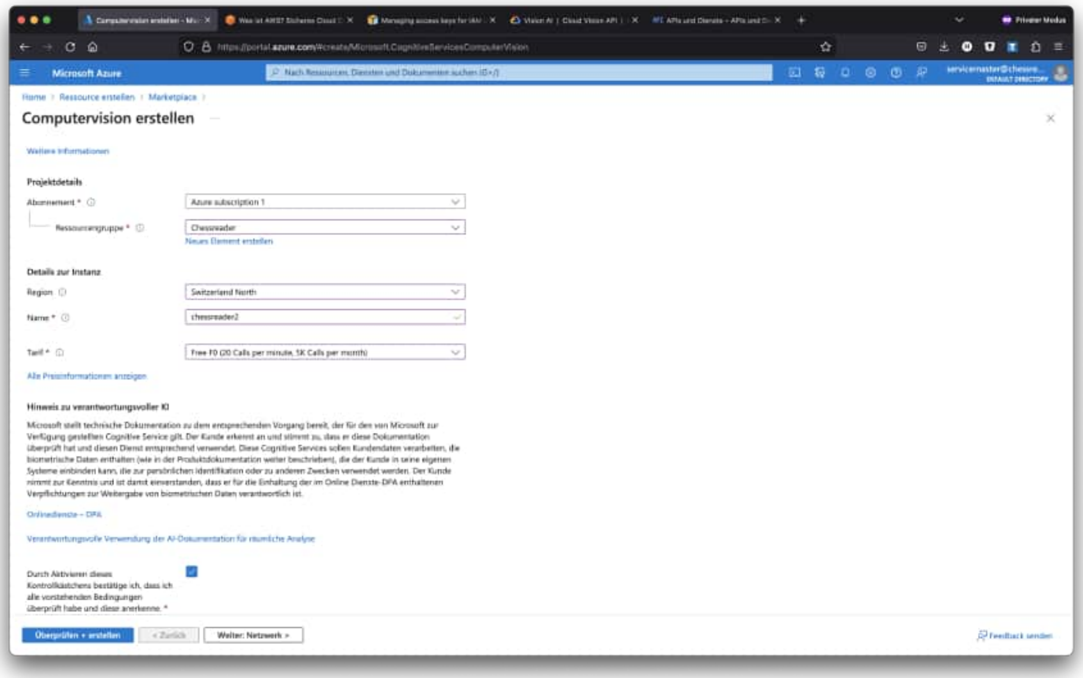
Azure Monitor: Überwachen Sie Apps und Infrastruktur.

Microsoft Defender for Cloud: Schützen Sie Ihre Apps und Infrastruktur.

Kostenverwaltung: Hiermit wird eine kostenlose Analyse und Optimierung Ihrer Cloudausgaben…

Figure A.2: Azure Portal Startpage

You will then be presented with a list of all available Microsoft services. Search for "Maschinelles Sehen" in the "KI + Machine Learning" category and create a new resource. You will be redirected to a setup wizard helping you through the process.


Microsoft Azure
Home › Ressource erstellen › Marketplace ›
Weitere Informationen
Neues Element erstellen
Alle Preisinformationen anzeigen
Onlinedienste – DPA
Verantwortungsvolle Verwendung der AI-Dokumentation für räumliche Analyse
Überprüfen + erstellen
< Zurück
Weiter: Netzwerk >
Feedback senden

Computervision erstellen

Projektdetails

Abonnement *	Azure subscription 1
Ressourcengruppe *	Chessreader

Details zur Instanz

Region	Switzerland North
Name *	chessreader2
Tarif *	Free F0 (20 Calls per minute, 5K Calls per month)

Hinweis zu verantwortungsvoller KI

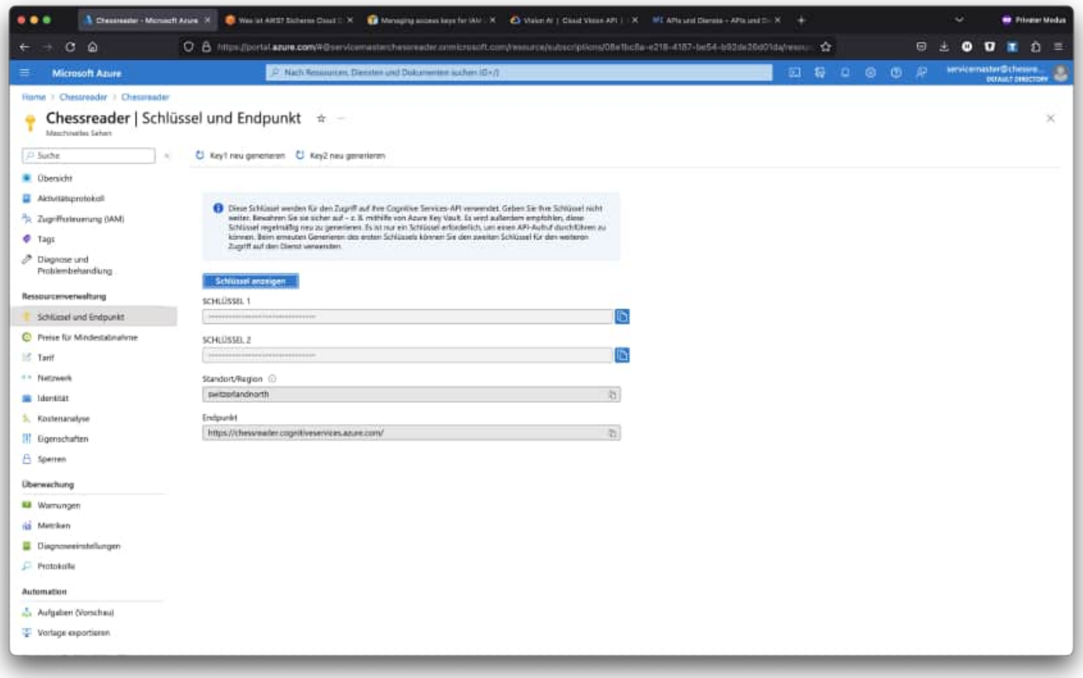
Microsoft stellt technische Dokumentation zu dem entsprechenden Vorgang bereit, der für den von Microsoft zur Verfügung gestellten Cognitive Service gilt. Der Kunde erkennt an und stimmt zu, dass er diese Dokumentation überprüft hat und diesen Dienst entsprechend verwendet. Diese Cognitive Services sollen Kundendaten verarbeiten, die biometrische Daten enthalten (wie in der Produktdokumentation weiter beschrieben), die der Kunde in seine eigenen Systeme einbinden kann, die zur persönlichen Identifikation oder zu anderen Zwecken verwendet werden. Der Kunde nimmt zur Kenntnis und ist damit einverstanden, dass er für die Einhaltung der im Online Dienste-DPA enthaltenen Verpflichtungen zur Weitergabe von biometrischen Daten verantwortlich ist.

Durch Aktivieren dieses Kontrollkästchens bestätige ich, dass ich alle vorstehenden Bedingungen überprüft habe und diese anerkenne. * ☑

Figure A.3: Computervision Wizard

Fill the fields with values appropriate to your needs, after which you can skip straight to creating the resource. If you do not have a resource group you can add this resource to, you can create one without leaving the wizard.

After completing these steps, all that is left is to get your access key and endpoint for the API. Do that by going to the Computervision-resource you have just created (it should show up on the startpage if you are not redirected automatically).



Diese Schlüssel werden für den Zugriff auf Ihre Cognitive Services-API verwendet. Geben Sie Ihre Schlüssel nicht weiter. Bewahren Sie sie sicher auf – z. B. mithilfe von Azure Key Vault. Es wird außerdem empfohlen, diese Schlüssel regelmäßig neu zu generieren. Es ist nur ein Schlüssel erforderlich, um einen API-Aufruf durchführen zu können. Beim erneuten Generieren des ersten Schlüssels können Sie den zweiten Schlüssel für den weiteren Zugriff auf den Dienst verwenden.

SCHLÜSSEL 1: ••••••••••••••••••••••••••••••••

SCHLÜSSEL 2: ••••••••••••••••••••••••••••••••

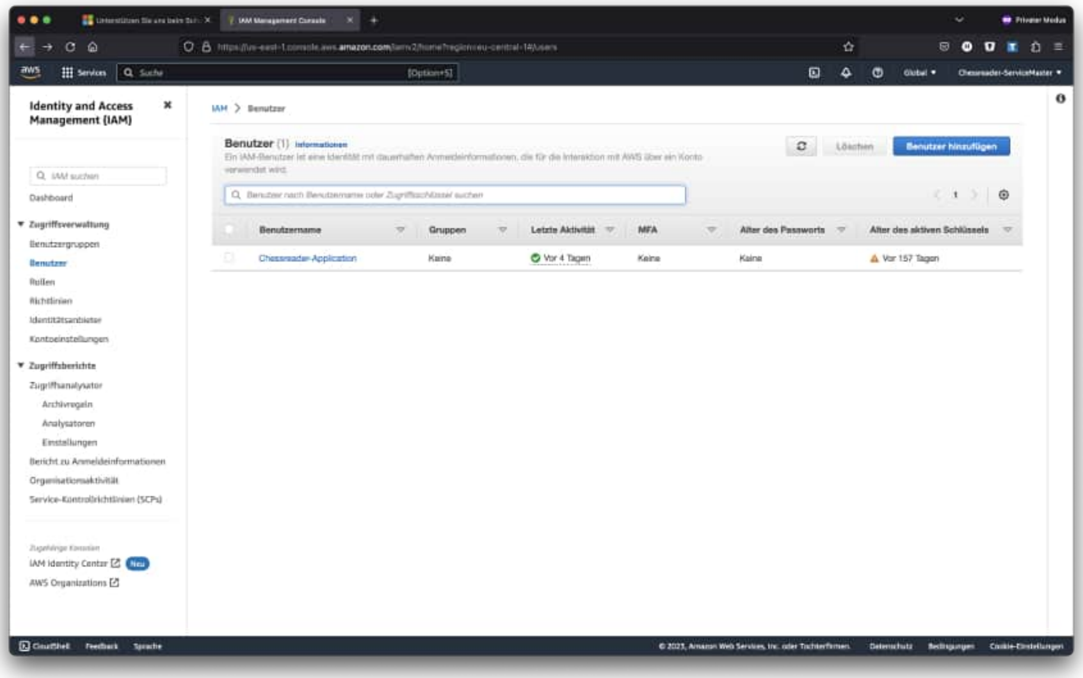
Standort/Region: switzerlandnorth

Endpunkt: https://chessreader.cognitiveservices.azure.com/

Figure A.4: Key and Endpoint screen

On the left is a navigation menu. Go to "Schlüssel und Endpunkt", then click on "Schlüssel anzeigen" to reveal the keys.

AWS Cognito Services
Log into the AWS Console (https://aws.amazon.com) and navigate to the "Identity and Access Management (IAM)" as seen below. You can do so by searching for "IAM" in the search bar at the top of the page



Benutzer (1) Informationen

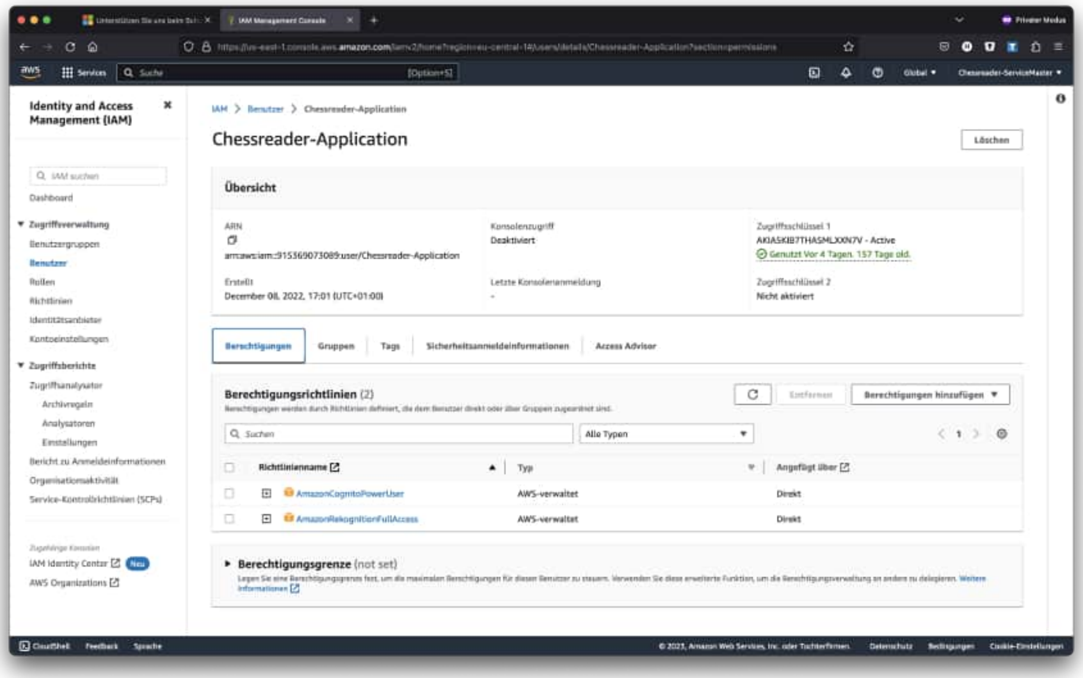
Ein IAM-Benutzer ist eine Identität mit dauerhaften Anmeldeinformationen, die für die Interaktion mit AWS über ein Konto verwendet wird.

Benutzername	Gruppen	Letzte Aktivität	MFA	Alter des Passworts	Alter des aktiven Schlüssels
Chessreader-Application	Keine	✓ Vor 4 Tagen	Keine	Keine	⚠ Vor 157 Tagen

© 2023, Amazon Web Services, Inc. oder Tochterfirmen.

Figure A.5: AWS Identity and Access Management

Click on "Benutzer hinzufügen" and give it the permissions seen in the screenshot below.



Chessreader-Application

Übersicht

ARN arn:aws:iam::915369073089:user/Chessreader-Application	Konsolenzugriff Deaktiviert	Zugriffsschlüssel 1 AKIASKIB7THASMLXXN7V - Active Genutzt Vor 4 Tagen. 157 Tage old.
Erstellt December 08, 2022, 17:01 (UTC+01:00)	Letzte Konsolenanmeldung -	Zugriffsschlüssel 2 Nicht aktiviert

Berechtigungsrichtlinien (2)

Berechtigungen werden durch Richtlinien definiert, die dem Benutzer direkt oder über Gruppen zugeordnet sind.

Richtlinienname	Typ	Angefügt über
AmazonCognitoPowerUser	AWS-verwaltet	Direkt
AmazonRekognitionFullAccess	AWS-verwaltet	Direkt

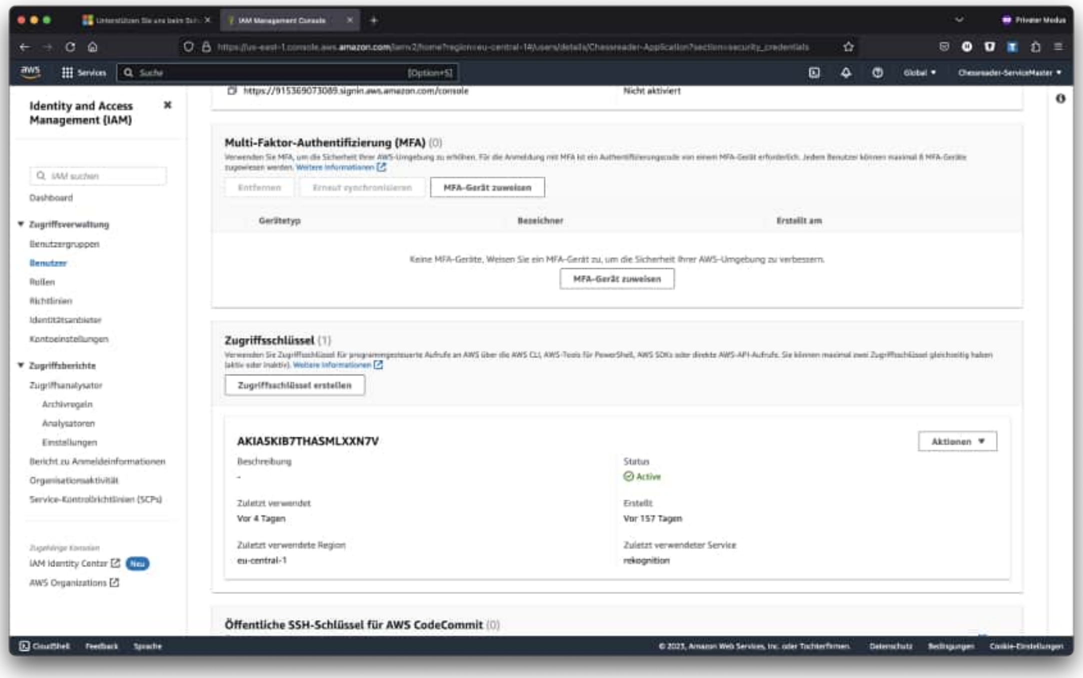
Berechtigungsgrenze (not set)

Legen Sie eine Berechtigungsgrenze fest, um die maximalen Berechtigungen für diesen Benutzer zu steuern. Verwenden Sie diese erweiterte Funktion, um die Berechtigungsverwaltung an andere zu delegieren. Weitere Informationen

© 2023, Amazon Web Services, Inc. oder Tochterfirmen.

Figure A.6: Permissions

After completing the steps in the wizard, navigate to the user you have just created in the IAM. Switch to the tab "Sicherheitsanmeldeinformationen", scroll down, and click on "Zugriffsschlüssel erstellen". In the wizard that opens select "Anwendung wird ausserhalb von AWS ausgeführt". Give your key a name and click on Create. Keep in mind that AWS will never show you this key again! Save it somewhere secure. If you happen to lose it, you will need to remove the one you lost and create a new one.



https://us-east-1.console.aws.amazon.com/iamv2/home?region=eu-central-1#/users/details/Chessreader-Application?section=security_credentials

https://915369073089.signin.aws.amazon.com/console Nicht aktiviert

Multi-Faktor-Authentifizierung (MFA) (0)

Verwenden Sie MFA, um die Sicherheit Ihrer AWS-Umgebung zu erhöhen. Für die Anmeldung mit MFA ist ein Authentifizierungscode von einem MFA-Gerät erforderlich. Jedem Benutzer können maximal 8 MFA-Geräte zugewiesen werden. Weitere Informationen

Gerätetyp	Bezeichner	Erstellt am

Keine MFA-Geräte. Weisen Sie ein MFA-Gerät zu, um die Sicherheit Ihrer AWS-Umgebung zu verbessern.

Zugriffsschlüssel (1)

Verwenden Sie Zugriffsschlüssel für programmgesteuerte Aufrufe an AWS über die AWS CLI, AWS-Tools für PowerShell, AWS SDKs oder direkte AWS-API-Aufrufe. Sie können maximal zwei Zugriffsschlüssel gleichzeitig haben (aktiv oder inaktiv). Weitere Informationen

AKIA5KIB7THASMLXXN7V

Beschreibung -	Status Active
Zuletzt verwendet Vor 4 Tagen	Erstellt Vor 157 Tagen
Zuletzt verwendete Region eu-central-1	Zuletzt verwendeter Service rekognition

Öffentliche SSH-Schlüssel für AWS CodeCommit (0)

© 2023, Amazon Web Services, Inc. oder Tochterfirmen.

Figure A.7: Permissions

Google Cloud Vision Services

Log into the Google Cloud Console (https://cloud.google.com). After logging in you will need to create a new project for the application. You can do that using the drop-down menu on the top bar.

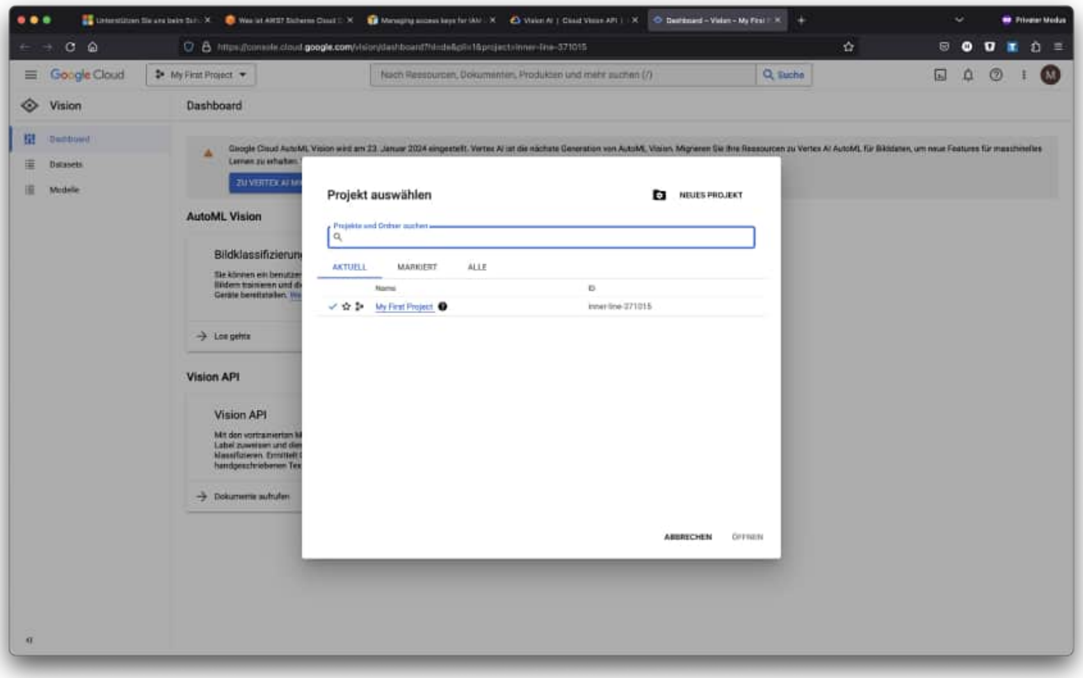
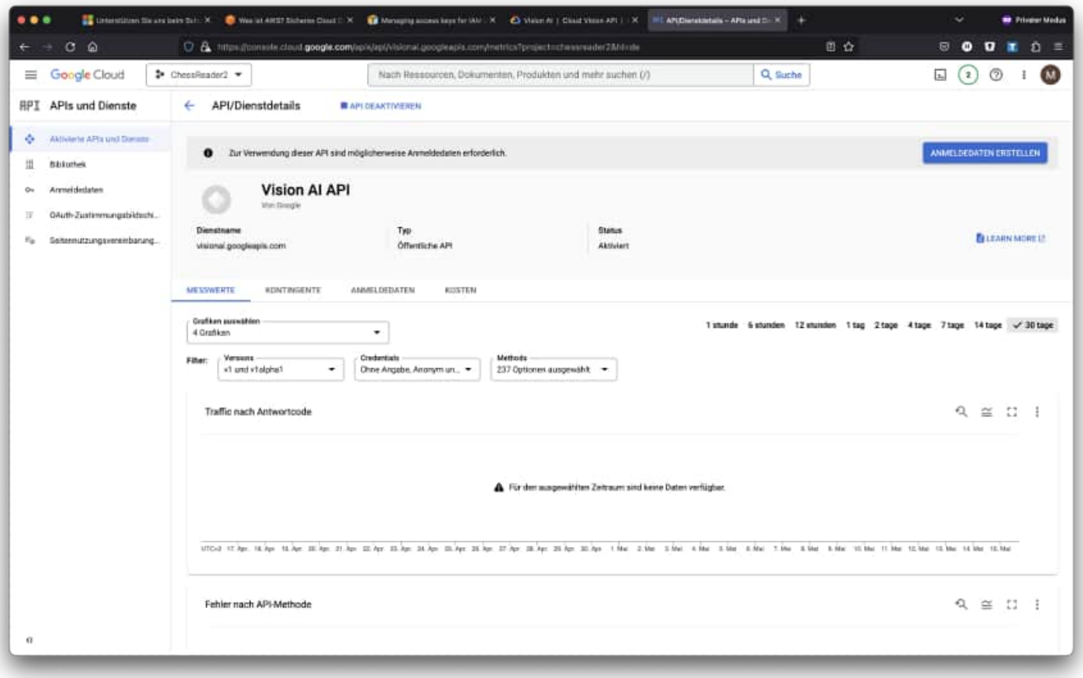


Figure A.8: Create a new project

Search for "Cloud Vision API" using the search bar and activate the resource for the project you have just created. You will then be redirected to the page shown in the screenshot below.



Zur Verwendung dieser API sind möglicherweise Anmeldedaten erforderlich.

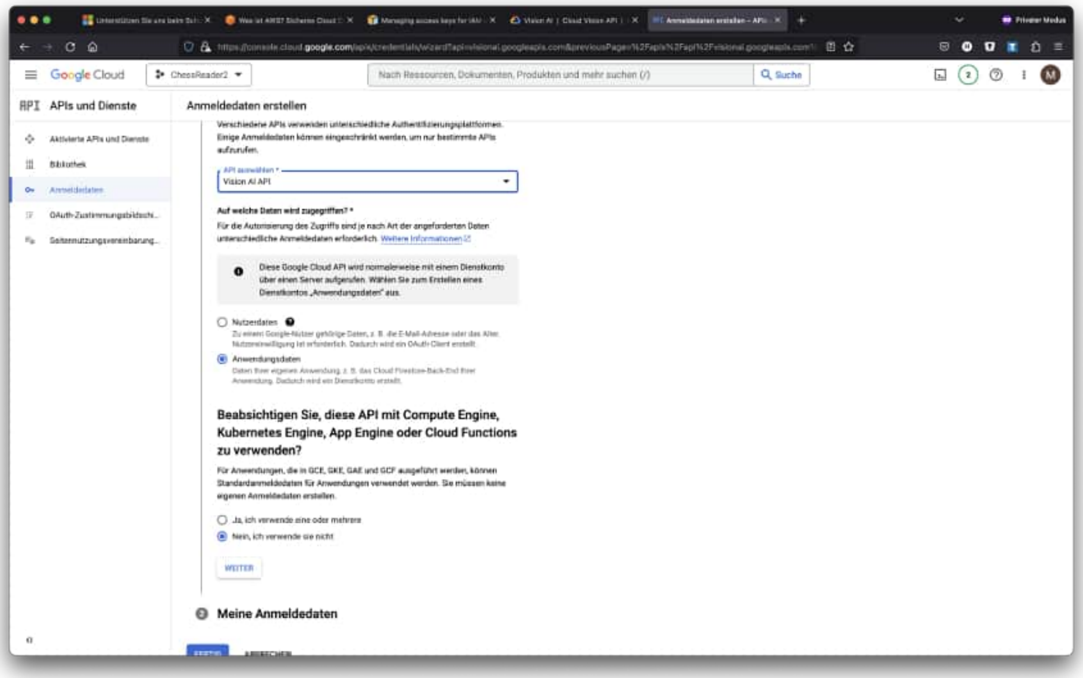
Dienstname: vision.googleapis.com
Typ: Öffentliche API
Status: Aktiviert

Grafiken auswählen: 4 Grafiken
Filter: Versions: v1 und v1alpha1 · Credentials: Ohne Angabe, Anonym un... · Methods: 237 Optionen ausgewählt

Für den ausgewählten Zeitraum sind keine Daten verfügbar.

Figure A.9: Permissions

Next, click on "Anmeldedaten erstellen" and fill out the wizard that pops up.


Anmeldedaten erstellen

Verschiedene APIs verwenden unterschiedliche Authentifizierungsplattformen. Einige Anmeldedaten können eingeschränkt werden, um nur bestimmte APIs aufzurufen.

API auswählen *: Vision AI API

Auf welche Daten wird zugegriffen? *

Für die Autorisierung des Zugriffs sind je nach Art der angeforderten Daten unterschiedliche Anmeldedaten erforderlich. Weitere Informationen

Diese Google Cloud API wird normalerweise mit einem Dienstkonto über einen Server aufgerufen. Wählen Sie zum Erstellen eines Dienstkontos „Anwendungsdaten" aus.

Nutzerdaten [✓ Anwendungsdaten]

Nutzerdaten: Zu einem Google-Nutzer gehörige Daten, z. B. die E-Mail-Adresse oder das Alter. Nutzereinwilligung ist erforderlich. Dadurch wird ein OAuth-Client erstellt.

Anwendungsdaten: Daten Ihrer eigenen Anwendung, z. B. das Cloud Firestore-Back-End Ihrer Anwendung. Dadurch wird ein Dienstkonto erstellt.

Beabsichtigen Sie, diese API mit Compute Engine, Kubernetes Engine, App Engine oder Cloud Functions zu verwenden?

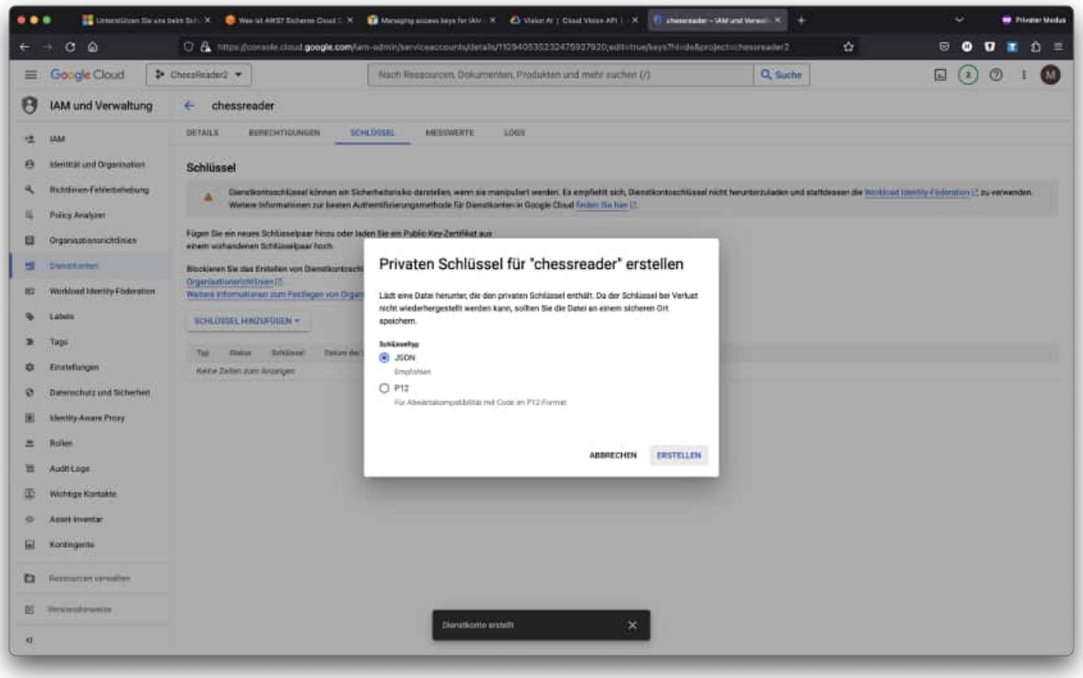
Für Anwendungen, die in GCE, GKE, GAE und GCF ausgeführt werden, können Standardanmeldedaten für Anwendungen verwendet werden. Sie müssen keine eigenen Anmeldedaten erstellen.

Ja, ich verwende eine oder mehrere [✓ Nein, ich verwende sie nicht]

Google Cloud
ChessReader2
Nach Ressourcen, Dokumenten, Produkten und mehr suchen (/)
Suche
APIs und Dienste
Aktivierte APIs und Dienste
Bibliothek
Anmeldedaten
OAuth-Zustimmungsbildschi...
Seitennutzungsvereinbarung...
WEITER
Meine Anmeldedaten
FERTIG
ABBRECHEN

Figure A.10: Creating a new app-account

To get the access key needed for the API, search for "IAM" in the search bar on the Console startpage. On the navigation menu on the left go to "Dienstkonten" and select the account you have just created. On the page that opens, navigate to the tab "Schlüssel" and create a new key. Select JSON as the key format. After finishing the key creation, it will automatically be downloaded by your browser.


Privaten Schlüssel für "chessreader" erstellen

Lädt eine Datei herunter, die den privaten Schlüssel enthält. Da der Schlüssel bei Verlust nicht wiederhergestellt werden kann, sollten Sie die Datei an einem sicheren Ort speichern.

Schlüsseltyp

[✓ JSON] Empfohlen

P12 Für Abwärtskompatibilität mit Code im P12-Format

Figure A.11: Creating a new access key

A.5 Configuring the credential_store.json file

Using the credentials you have obtained by following the previous section, you can now set the corresponding values in the credential_store.json file.

The file looks like this

```
{
  "az": {
    "endpoint": "",
    "key": ""
  },
  "gl": {
    "type": "",
    "project_id": "",
    "private_key_id": "",
    "private_key": "",
    "client_email": "",
    "client_id": "",
    "auth_uri": "",
    "token_uri": "",
    "auth_provider_x509_cert_url": "",
    "client_x509_cert_url": ""
  },
  "rk": {
    "aws_access_key_id": "",
    "aws_secret_access_key": ""
  }
}
```

Listing A.2: Censored credential_store.json file

You should be able to just copy-paste the credentials you have obtained to the file.

A.6 How to run the project

To run the client-side application locally, navigate to the top-level folder of the source code and run the command

```
$ ng serve
```

To run the server-side locally, navigate to the top-level folder and run the command

```
$ flask run
```

If you need debug functionality you can run

```
$ flask run --debug
```

If you run the command with the debug argument, you need to set your Python path to include the source code due to a bug in the Werkzeug library[14]. To do that on a Mac you can run

```
$ export PYTHONPATH=${PYTHONPATH}:ChessReader-API
```

Please make sure you have also set all environment variables correctly and installed all packages.

A.7 Releasing a new version

Building and releasing a new version of ChessReader is very straight forward and unless major changes were made to the underlying workings should only require running a few commands.

A.7.1 Building the frontend and backend

Building the frontend and backend is very easy. You only have to run a single command per repository. To build the project run the command

```
$ docker build -t chessreader/{{subproject}}:latest .
```

where you should replace {{subproject}} with either *chessreader_angular* or *chessreader_api*, depending on which repository you want to build. What this does is install all required packages and compile the code before packaging everything into an image that is saved locally. To see how you can upload the resulting image to dockerhub, see the next section.

A.7.2 Publishing to dockerhub

First log into the dockerhub account you want to publish the image to using

```
$ docker login
```

An official ChessReader-Account does exist and you might have been given the credentials to it.

After building the project you can push the images to your account using

```
$ docker push {{image_name}}
```

where you have to replace {{image_name}} with the name of the image you have built beforehand. If you have followed this guide that is either *chessreader/chessreader_angular* or *chessreader/chess-reader_api*

A.8 Source code structure

This part contains a short overview of the code structure in the two repositories.

A.8.1 Frontend

The basic structure of the frontend follows the default setup of an Angular application. To read more about this setup you can visit the Angular documentation[15] In order to read more about the frontend consult the semester thesis "Digitalization of Chess Scorecards"[12]

core

The core folder contains files that do not directly interact with the user, but provide functionality from interfacing with the API and authentication of requests to catching errors.

The most important file in this directory is the file chessreader-api.ts which contains all functions calling the different endpoints of the API and returning the results.

pages

The pages directory contains all components that act as pages. These are the components that are referenced when defining new routes and are responsible for managing the entire workflow assigned to this page.

Each subdirectory contains the files associated with a page as well as subcomponents that are used exclusively by the respective page component.

shared

In the shared directory, you will find all components that are more general and not associated with a specific page. Here you can also find a file called mat-all.ts. In it, all material-components are imported into the application so they can be used without worrying if they are available or not.

A.8.2 Backend

There are four main folders in the backend-repository, as well as some additional files, which are explained below

database

In this directory you will find the database initialization function as well as the file containing the models used by SQLAlchemy to build the database. In the db.py file, you can also set the database to reset each time you restart the flask app. Keep in mind that this does not remove the pictures that were uploaded via the API. To delete those as well you have to remove the contents of the media folder manually

media

This folder may not exist when first downloading the repository. If that is the case it will be created automatically. It contains all images that were uploaded as well as all cut-out images of single boxes, which can then be sent to the client.

model

This is the directory where the magic happens. It contains the pipeline, used to perform all calculations, and is responsible for contacting the OCR-APIs, and supporting files. All files in this folder should be considered legacy files. Documentation of functions may be inaccurate or incomplete, if available at all. Code that resides here should be replaced and moved to a different folder, in order to prevent unstructured changes that accumulate over time.

resources

In the resources folder, you will find all files that are responsible for serving content to clients of the API. The routes file defines which path lets clients access which resource. Resources are handling all web requests and responses.

Other important files

Other important files include

- the app.py, which serves as the entry point of the application
- the config.py, which contains important global variables, required for correct operation
- the credentials_store.json containing the access keys for the OCR-APIs
- the requirements.txt, listing all packages that are required by the application and their version numbers. **PLEASE UPDATE THIS FILE** (including version numbers) if you make any changes to the required packages

Appendix B

API-Documentation

The following is an overview of the currently available endpoints to the API. It lists URLs, required payload, and authentication. Also included is a short description of what the endpoint does.

End-point: Login

Retrieve a JWT token, which can be used to access the other endpoints.

Method: POST

> ```
> http://host:5000/api/auth/login
> ```

Body (raw)

```
{
    "email": "mail@example.com",
    "password": "securePassword"
}
```

Authentication noauth

Param	value	Type

Response

Success:

```
{
    "token": "jwt-token-for-user",
}
```

- -

End-point: Signup

Create a new account

Method: POST

> ```
> http://host:5000/api/auth/signup
> ```

Body (raw)

```
{
    "email": "helibum1@gmail.com",
    "password": "password",
    "name": "Lukas-Test"
}
```

Authentication noauth

Param	value	Type

- -

End-point: Get match image url

Get the url to the image of a match

Method: GET

> ```
> http://host:5000/api/auth/signup
> ```

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

- -

End-point: Get Matches

Method: GET

Get all matches of a user

> ```
> http://host:5000/api/matches
> ```

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

- -

End-point: Get Match

Get a match by its id

Method: GET

> ```
> http://host:5000/api/matches/<match_id>
> ```

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

- -

End-point: Get PGN String

Get the PGN string for a match, using its match_id

Method: GET

> ```
> http://host:5000/api/matches/<match_id>/pgn
> ```

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

- -

End-point: Get Match Boxes

Get the detected table-boxes of a match, given by its match_id

Method: GET

> ```
> http://host:5000/api/matches/<match_id>/boxes
> ```

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

- -

End-point: Trigger OCR-Scan

Trigger the OCR for a match, selected by its match_id

Method: PATCH

> ```
> http://host:5000/api/matches/<match_id>/scan
> ```

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

- -

End-point: New Match

Add a new match to the database. The picture needs to be uploaded in this request as a formdata-file

Method: POST

> `http://host:5000/api/matches`

Body formdata

Param	value	Type
file	chess_scorecard.png	file

Authentication bearer

Param	value	Type
token	jwt-token-for-user	string

List of Figures

Bibliography

[1] H. J. R. Murray, *A history of chess.* Benjamin Press, 1986.

[2] L. Thomas, "The Psychological Drama of the World Chess Championship," *The New Yorker*, Dec. 2021, section: the sporting scene. [Online]. Available: https://www.newyorker.com/sports/sporting-scene/the-psychological-drama-of-the-world-chess-championship

[3] "Portable network graphics," 2023. [Online]. Available: https://en.wikipedia.org/wiki/PNG

[4] N. Ambrosini and G. Hellinger, "Digitalization of Chess Scorecards," *unpublished*, Jun. 2022.

[5] N. Majid and O. Eicher, "Digitization of handwritten chess scoresheets with a bilstm network," *Journal of Imaging*, vol. 8, no. 2, 2022. [Online]. Available: https://www.mdpi.com/2313-433X/8/2/31

[6] M. Wynne, L. AHDS Literature, L. (Organization), Arts, and H. D. Service, *Developing Linguistic Corpora: A Guide to Good Practice*, ser. AHDS guides to good practice. Oxbow Books, 2005. [Online]. Available: https://books.google.ch/books?id=41ZJAAAAYAAJ

[7] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, p. 160018, Mar. 2016. [Online]. Available: https://doi.org/10.1038/sdata.2016.18

[8] FIDE, "FIDE Handbook." [Online]. Available: https://handbook.fide.com/

[9] "Understanding Chess Notation." [Online]. Available: https://www.dummies.com/article/home-auto-hobbies/games/board-games/chess/understanding-chess-notation-192295/

[10] C. B. Yann LeCun, Corinna Cortes, "Mnist handwritten digits database," 2023. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[11] S. An, M. J. Lee, S. Park, H. Yang, and J. So, "An ensemble of simple convolutional neural network models for MNIST digit recognition," *CoRR*, vol. abs/2008.10400, 2020. [Online]. Available: https://arxiv.org/abs/2008.10400

[12] L. Boner and M. Hostettler, "Digitalization of Chess Scorecards," *unpublished*, Dec. 2022.

[13] "Sqlalchemy documentation." [Online]. Available: https://docs.sqlalchemy.org/

[14] C. Seibert, "Flask absolute import bug in debug mode." [Online]. Available: https://chase-seibert.github.io/blog/2015/06/12/flask-werkzeug-reloader-python-dash-m.html

[15] "Angular Documentation." [Online]. Available: https://angular.io/docs