THM | CAMPUS GIESSEN | MNI
TECHNISCHE HOCHSCHULE MITTELHESSEN
Mathematik, Naturwissenschaften und Informatik

Master Thesis

Expanding the web app "arsnova.cards" into a learning platform with an automatic multiple choice system and monitoring tools for the learning process

to obtain the academic degree

Master of Science (M. Sc.)

submitted to the
Department of Mathematics, Natural Sciences and Computer Science of
the Technischen Hochschule Mittelhessen
from

Curtis Adam

on April 2021

Supervisor: Prof. Dr. Klaus Quibeldey-Cirkel
Co-Supervisor: M. Sc. Sebastian Süß

Affidavit

I hereby assure that I have prepared the present work independently and exclusively using the literature and aids indicated. The paper has not been submitted to any other examination board in the same or similar form and has not been published.

Gießen, April 30, 2021 Curtis Adam

Outline

The content of this master thesis creates an overview of the .cards application. It presents the individual features that a user can access while using the frontend or backend. Afterward, a comparison with common learning platforms is being made. After that, the features that have been implemented within the timeframe of the development project and during the creation of this thesis will be presented in further detail. These include the display and evaluation of learning statistics for individual users, as well as the implementation of an automated system for evaluating submitted answers. This is followed by an analysis of the results from a UX report, which was conducted by students in an SWT module during the winter semester 20/21. Finally, a list of open issues follows, which represents a high priority for the further development of the web app.

Contents

1. Introduction

The flashcards web app .cards [Tea21b], has been used for several semesters at the Technische Hochschule Mittelhessen for awarding bonus points. Students are being credited for the repeated learning of flashcards. The Leitner algorithm is used for regular repeated learning. Additionally, the web app offers the possibility to receive bonus points for lecture notes.

1.1. Problems and goals

Since the end of 2019, the learning behavior of the participants is logged. A lecturer can view this in the statistics of his activated bonus learning phase. It turned out that the majority of the participants completed the daily learning workload within a few minutes, although a time requirement of at least one hour per learning workload is set. The flashcard web app had to be extended so that compliance with the prescribed learning time is guaranteed. Several new features had been considered for this goal:

- The client Pomodoro timer will be logged and synchronized on the server.
- A single/multiple choice question system will be introduced.
- A new learning mode that respects the minimum number of Pomodori.

To make the user take a break during a learning phase, a screen locking mechanism is introduced, which can't be bypassed. Furthermore, the answers for the new question system are not transmitted directly to the client database to make cheating more difficult.

Other focal points of the development project are related to improvements of the content management process, as well as traffic and performance optimizations. For the latter, the web app in its current version is unusable on mobile networks. During the first page load, several megabytes of data are sent to the client. Lastly, some modules of the web app need to be refactored and documented so that subsequent developers can continue to maintain .cards.

2. About .cards

The learning platform .cards, is a web application based on the Node.js web framework Meteor [Sof21b] since the development of version 3.0 in the year 2016. It has been enhanced with several new features over the years. On this platform, users can create new content in the form of a learning card, which is stored as a group inside a card set. The content of these cards can be made available to the public and, if the card type allows it, repeatedly learned with algorithms such as Leitner [Stu21] or Woźniak [Gar08]. As of the time of this thesis, only the Leitner algorithm has been fully implemented. The web app has been enhanced with several new features after the release of 3.0, some of the major ones include:

- **Markdeep:** A markup language developed by Morgan McGuire, which can be seen as an extension of Markdown [Mor21]. With it, it's possible to create diagrams, embed videos, or even style the content with CSS. This language is used for all user-created content on the learning platform. Moreover, it's possible to export the created card content into a Markdeep only HTML file, which can run independently from the .cards web app itself.
- **Automatic awarding of bonus points:** Students can enroll in different card sets and learn the cards of these at regular intervals. Details such as the maximum learning workload or the deadline can be configured individually by the lecturer for each card set. In addition, the lecturer can view detailed statistics about the learning behavior of each student, as long as they've enrolled into a bonus.
- **Transcripts:** Students can earn bonus points for a module if they upload their transcripts for the lecturer onto the .cards web app. The lecturer has the option to review said scripts and rate them based on a 5-star system. The number and the minimum overall quality of the submitted transcripts are determined by the lecturer and the bonus points are dynamically calculated based on said rating.

There are two active major deployments: arsnova.cards and linux.cards. The first one is used by THM students and lecturers and allows them to freely create and share content. Besides that, it's also the only deployment that enables the transcript feature. The second installation is more moderated and only allows a set number of people to create and publish new content. Said content is used to prepare students for the LPIC-1 | LPIC-2 certification tests. Students which learn regularly will be offered full payment for the certification process.

2.1. Installation Guide

Detailed instructions on how to install the web app can be found in the readme of the official GitLab repository. There are two forms of installation: Native and with Docker. The native installation only covers Linux. It is recommended for Windows users to use the docker guide.

2.2. Landing Page

The first route that unregistered users will be redirected to. It provides quick access to a demo mode, which showcases examples of all available card types, as well as demonstrating the Pomodoro timer, which is being used for the implemented learning modes during a break. The only other visible navigation elements redirect towards an explanation of the platform itself, as well as the general terms and conditions and data privacy.

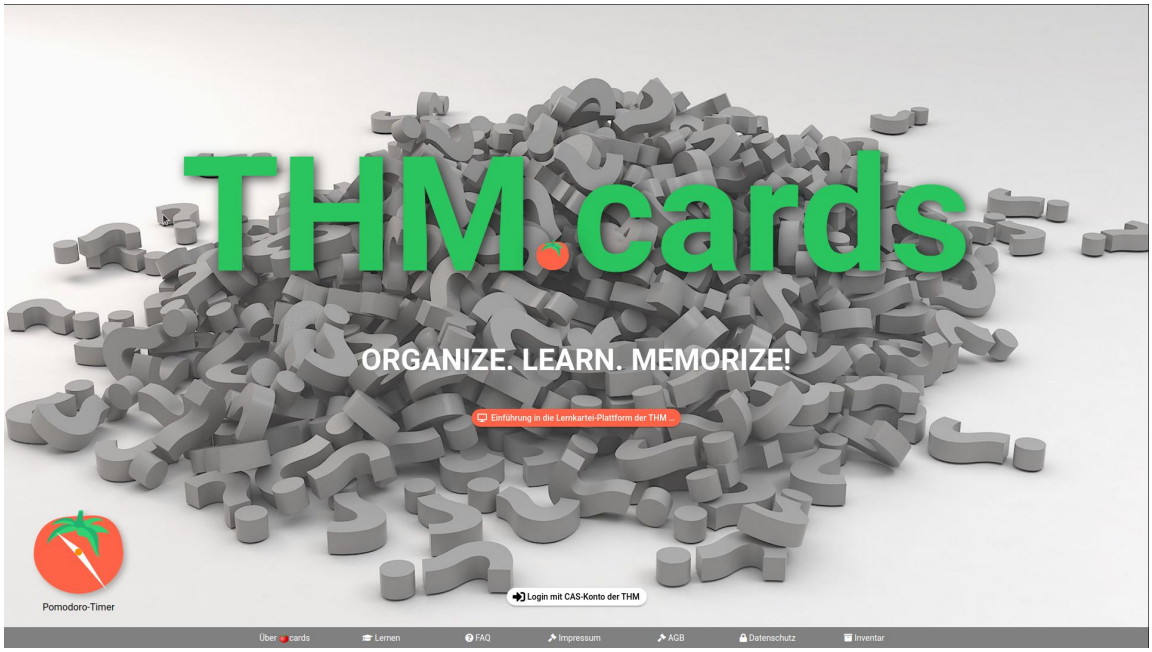


Figure 2.1.: .cards landing page

2.3. Account roles

There exist a total of 7 different account types: Guest, free, edu, pro, lecturer, proofreader, and admin. Each deployment can decide which roles should be available except for the

admin role, which is always required to exist at least once. Registration is done with services such as Facebook, Twitter, Google, CAS, or a built-in registration system specifically designed for the .cards web app.

- **Guest**: Displays the lowest type of permission that a user can have. A Guest is in general, not a true account that gets stored inside the database of the web app. Rather it's a session stored in local storage, which allows access to all free frontend content such as cards and card sets. Users of this role can only watch content and not use features such as the repeated learning algorithms, as they require a database entry for the user itself. Server administrators can decide if guest accounts can be used on the current deployment.
- **Free**: Is the lowest permission that a user can have while being registered inside the web app. This and any other higher role requires registration with any type of the listed above login services. This role is similar to a Guest account, with the exception that the user can use the learning algorithms.
- **Edu**: Is designed for students and has a different functionality depending on the .cards installation. On arsnova.cards, edu users are allowed to create and publish content while on linux.cards, they are only allowed to view and learn. Compared to the free role, edu users can sign in to an active learning bonus.
- **Pro**: Can always create content and in addition to making it just freely available for everyone, set a price tag to make the content accessible. This feature will notify a lecturer role who can decide if the pro user is allowed to sell his content or deny the request. Users with a role lower than pro will only see a preview of a maximum of 5 cards while the content hasn't been purchased. Any higher role will have always full access without paying anything. The pro role is disabled in all current official deployments and will be first fully implemented with a later release.
- **Lecturer**: Is allowed to start a bonus for a specific card set or repetitorium. This role can also review pro content before it becomes available for everyone.
- **Proofreader**: Have full access to all the frontend content. They can edit already available content or, if necessary, delete them. To make the administration easier, they can also access private content which they don't own.
- **Admin**: Displays the highest available permission. They can do everything that a proofreader can do with the addition of having access to the backend. This allows them to make quick adjustments to the entire server, such as disabling mail and push notifications, as well as managing all user accounts on the web app, as long as the account in question isn't another admin.
- **First login**: A special role that gets only used during the registration process. It prevents the user from entering the frontend as long as the user didn't accept the general terms and conditions.

- **Blocked**: Tells the system that the user is not allowed to access any type of content while using this account. Instead, the user will be redirected to a permission denied page. Only admins can set the blocked role to another user.

2.4. Table of contents

The Frontend is categorized into different navigation groups that the user can access. "Topic pool" allows access to everything that has been made publicly available, "My content" only displays the card sets that the user created, "Workload" will show every active learning phase that the user registered to, and "Transcripts" will display both private and bonus transcripts. If the user got the admin role, then there will be an extra option named "All" which displays all the content that has been created on the platform.

Figure 2.2.: .cards topic pool filter

Each category offers a filter and search bar, to quickly find specific types of content. Furthermore, each item is referenced with labels which can be used to quickly filter for specific types.

2.4.1. Wordcloud

There are two different display methods: The table of content, which is the default option for each route, as well as a word cloud, which displays the titles of each element. The size of the title depends on the size of the content: Elements with less content will appear smaller compared to those with a lot of content.

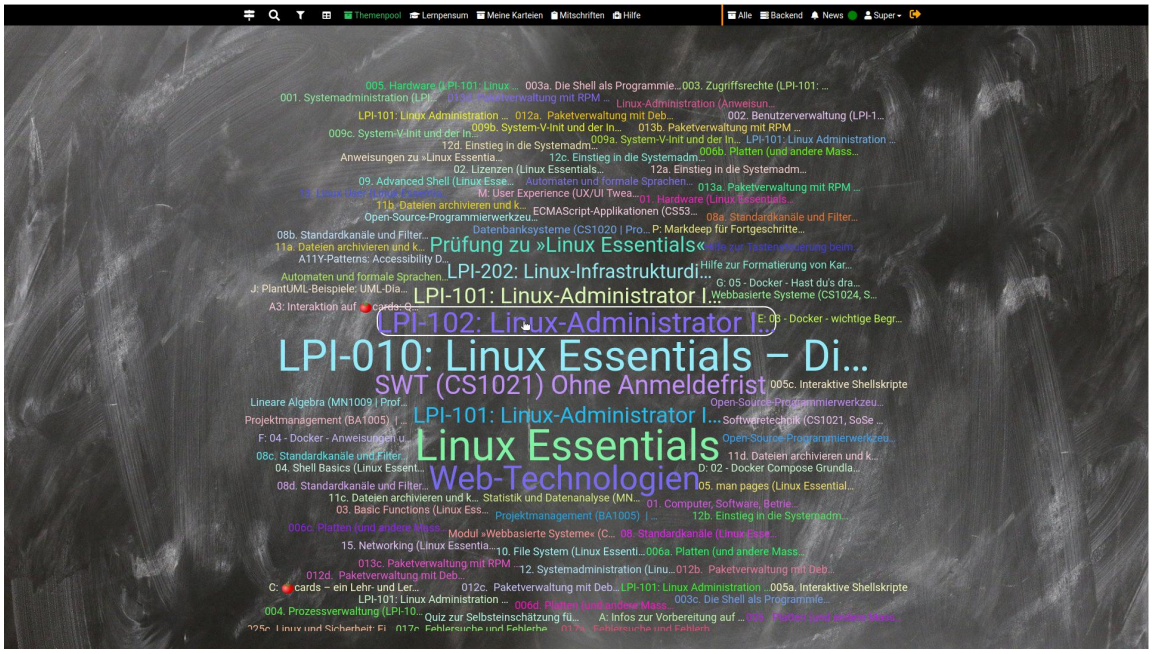


Figure 2.3.: .cards wordcloud

2.5. Card sets / Repetitorium

features and layout of the cards depend on the selected card type of the card set. Each card set can only include one specific card type. To get around this limitation, the user can combine several different types of card sets with a repetitorium. This is a special type of card set, which only references the content of others and can't contain any content of its own. This has the benefit, that the referenced content doesn't have to be duplicated and stored as new data inside the database and also quickly allows for different combinations, depending on which contents a single module might require.

2.6. Welcome screen / Signpost

The welcome screen is a bootstrap modal [Inc13] that opens the first time the user is logged in with an account into the frontend. The content provides quick navigation links to commonly used features as well as highlighting specific card sets, which are marked by the admin as points of interest.

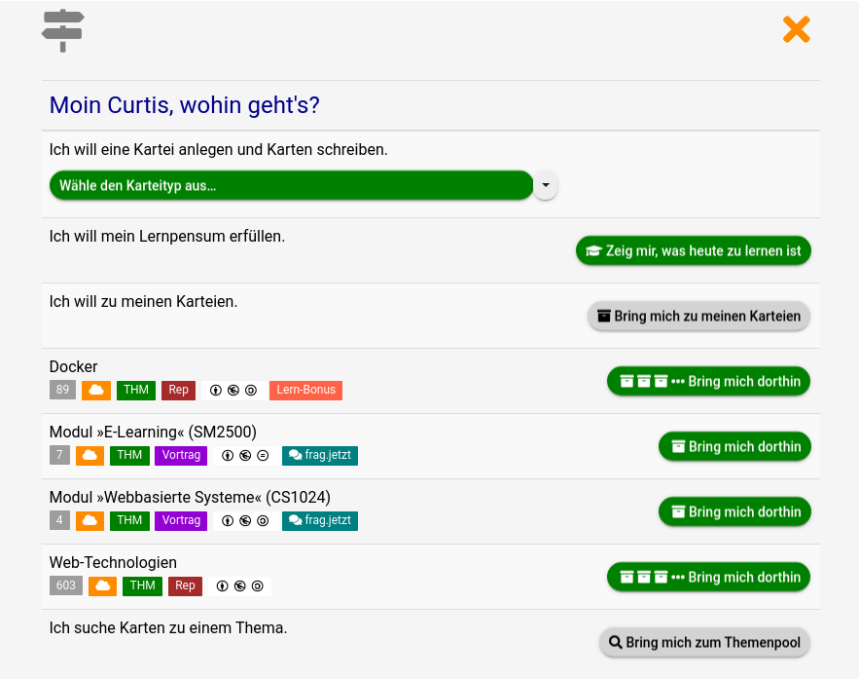


Figure 2.4.: .cards welcome screen for logged in users

2.7. Card Types

There are currently 21 different card types. These can have between 1 - 6 different sides, which is the perfect amount to display them as a cube, as well as contain additional properties. The platform is also configured to allow the developer to customize the available card types through the use of a configuration file. This includes the number of sides, the order of the side, and the behavior of the content. A later version will allow the administrator to adjust this setting per deployment without having to recompile the app. Either through direct manipulation of a .json file or with an interactive editor, located inside the backend UI.

Name	Sides	Learning Modes	Misc
Lerneinheit	5	Yes	
Vokabeln	2	Yes	
Mitschrift	3		
Glossar	3	Yes	
Zitat	3	Yes	
Prüfung	3	Yes	
Anweisung	3	Yes	
Fachliteratur	6	Yes	
Notiz	1		
To-dos	1		
Fotokartei	2		
MC-Quiz	2	Yes	Automated-Quiz
Pattern	6	Yes	
Formel	4		
Vortrag	1		ARSnova apps integration
Aufgabe	2	Yes	
Zielerreichung	2		
Inverses Fragen	2	Yes	
Kubus	6	Yes	ARSnova apps integration
Bonus-Kartei für Mitschriften	None		
Quiz	2	Yes	

Table 2.1.: .cards card types

2.8. Transcript-Bonus

A teacher can distribute bonus points to his students if they submit their transcripts which they've created during a lecture. The .cards web app has a new card type for this purpose, which will automate this process.

2.8.1. Review-System

Before students can submit transcripts for a bonus, the teacher must first configure the card set that will be used for storing said transcripts. A date for the individual lecture days is required, as well as the time for the activation, photo submission, and for digital post-editing time. A topic can also be defined for each lecture. If no topic is defined, the title of the card set is used. To allow the teacher to select multiple dates in the UI, the jQuery extension MultiDatesPicker [Luc17] is being utilized.

After a lecture is over, the web app is going to unlock it for the upcoming submissions. Students have a limited time to link their transcript to the lecture before the digital post-editing starts. For this purpose, an extra button is provided within the editor for the transcript. Already linked transcripts can be viewed by the teacher. After the digital post-editing time has expired, the teacher can rate the content of the transcript. Here he can use a review mode, which displays all unevaluated transcripts in a presentation view. The transcripts are rated by a 5-star system. If there is a general deficiency, the transcript can also be rejected without awarding a star.
The final score of the awarded bonus points is composed of the average ratings of the submitted transcripts, as well as the minimum number of transcripts required to receive the maximum points.

2.8.2. Transcript overview

There are two overviews for the submitted transcripts: 'Overview' shows a summary statistic, grouped by user. This view also shows the bonus points received and the number of submitted transcripts. This information can also be exported as a .csv file, as is the case with the Leitner bonus. The 'submissions' view shows a list of the individual transcripts, sorted by the day of submission and the status of the transcript:

- **Locked:** The transcript is currently undergoing digital post-editing and cannot yet be reviewed.
- **Not reviewed:** The digital post-editing period has expired and the transcript is ready for review.
- **Accepted:** The transcript has been accepted. This status also displays the number of stars awarded.
- **Denied:** The transcript was not accepted.

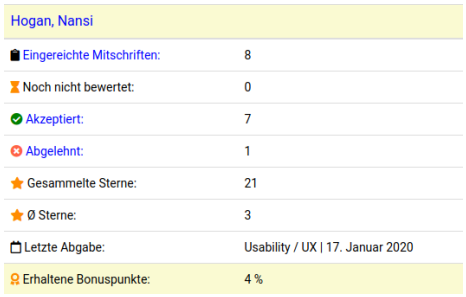


Figure 2.5.: .cards transcript bonus overview

Next to each status field, there is an additional link to a modal that explains the reason for the rejection or an explanation of the star rating.

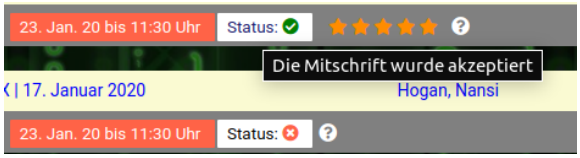


Figure 2.6.: .cards transcript status labels

2.9. Presentation & Demo

Displays the created content of the selected card set and provides navigation options on each side of the display. This is the route, that users will for most of the time while either just looking at cards or also while learning them. By default, desktops display each card in a 3D mode, which can be manually disabled.

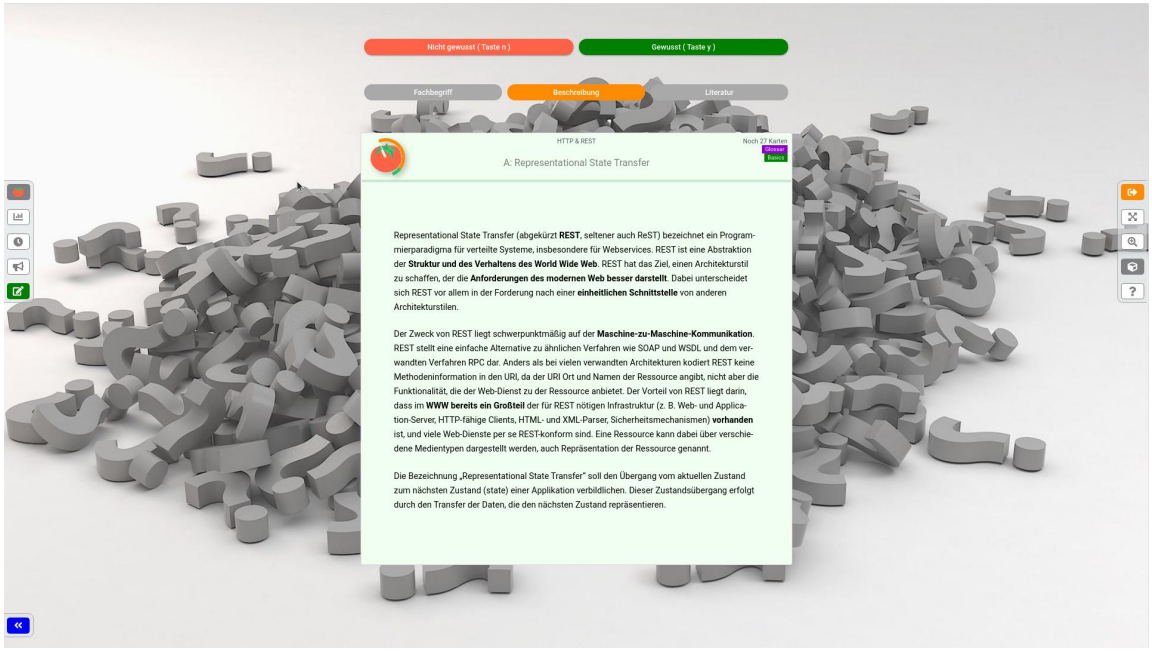


Figure 2.7.: .cards presentation view

2.10. Editor

New cards and transcripts can be easily created by a built-in content editor. This editor allows the user to insert content written in a Markdeep syntax on the left side and provide a preview on the right side in real-time. The preview can either be displayed in a 3D cube format, used by desktop and tablets, or a smartphone preview, which is limited to 2D mode only. This route is limited to desktop and tablets, due to layout limitations.

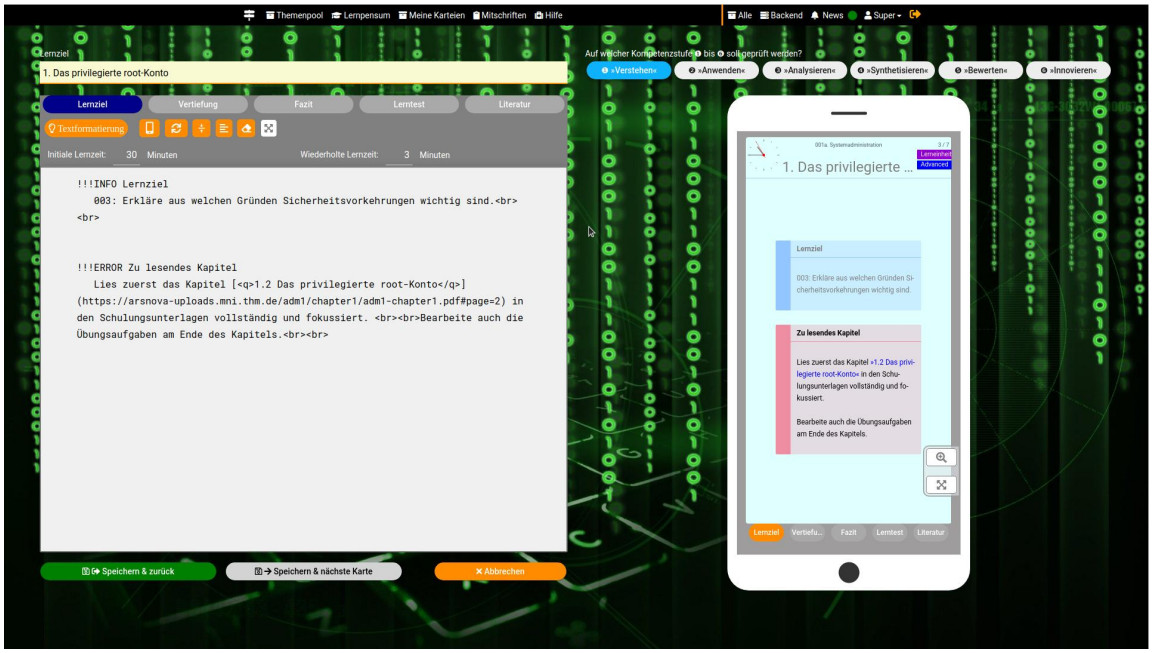


Figure 2.8.: .cards editor with preview

2.11. Leitner & Woźniak algorithms

The web app .cards offers two learning algorithms for repeating learning: Leitner and Woźniak. As of the time of this writing, only the Leitner algorithm has been fully implemented. In this mode, each card has a learning status of 6 individual boxes, numbered from 1 to 5 and 6 (learned). Every card starts at the first box and gets moved up to a higher numbered box, if the user answers the card correctly, and moved back if the answer was incorrect. The behavior for incorrect answers can differ between each .cards server installation. The user has a maximum workload of cards that he needs to answer in between a set deadline. New cards are assigned on the next day if the current workload has been completed or the deadline has been reached. Each card receives an individual cool down depending on the box they've moved into. For example in a learning phase which is supposed to last for several months, box one can have an interval of 1 day, box 2 of 3 days, box 3 of 7 days, box 4 of 21 days, and box 5 of 84 days. The user completed the learning phase if a set date got reached or if all cards have been moved into the last box. To determine the ideal workload for a set time frame and box interval, lecturers have access to a simulator.
There are two ways a user can send his answers to the server: Either by saying if he knew or didn't know the answer by himself or by selecting the right answers through a multiple-choice question. For the latter, the server checks if the selected answers match with the solution stored inside the database and then moves the card to the according box.
If the user learns inside a bonus, then a lecturer can see their current status and learning history. Additionally, the .cards app web app automatically calculates a set amount of received bonus points for the user, based on how many percent of all cards reached the final box.

The Woźniak mode offers a selection of 6 answer options: Ranging from "No clue" to "Known immediately". Only cards which haven't been answered with the last option will be repeatedly learned.

2.11.1. Leitner simulator

In this mode, a complete learning phase is simulated, from the day of activation to the end. Six dates are set to create a snapshot of the current learning status, which can be navigated to. The last date is the day when the learning phase reached its official end.
Here the lecturer has the option to adjust the daily workload, the intervals as well as an error rate for each subject. After each adjustment, the simulator automatically calculates the new status. The fields in the error rate indicate how many cards out of the total learning workload will not be known inside the associated box. As an example for a learning workload of 400 cards: 50

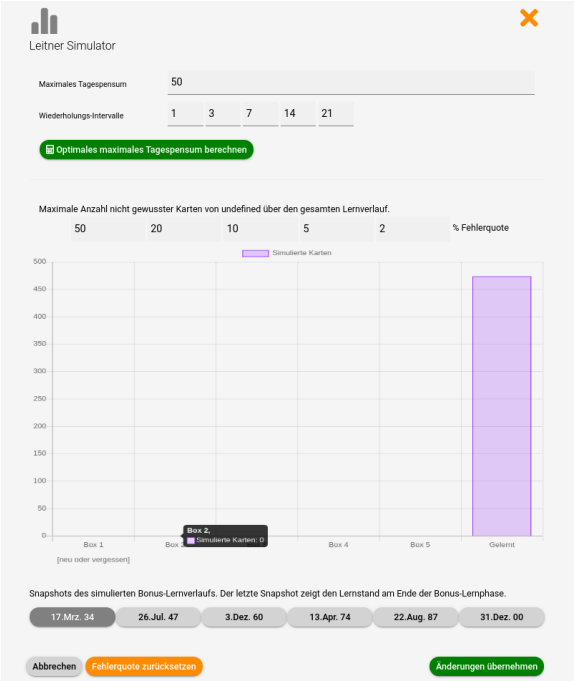


Figure 2.9.: .cards leitner simulator modal

2.12. Backend

This site can be only accessed by admin accounts. For security reasons, admins cannot be added through the web app but only directly via the database.

- **Dashboard:** Displays global statistics of the app including the number of card sets, repetitories, cards, registered and online users.
- **Users**: Allows admins to manage each user. Here they can change the active roles, edit information such as username and e-mail, or even remove them and their content from the platform.
- **Matomo:** Displays user statistics for the current.cards server installation. From which regions the users are coming from, how they've reached the .cards web app, and which pages they've visited.
- **News:** Allows admins to set a message of the day. Each message can have a start and end date, as well as the area in which it is supposed to be displayed. There are currently three different areas that can be defined: Landing page, frontend, or both.
- **Settings:** Can be used to change settings that don't require an entire server start to be applied. These include globally toggling e-mail and push notifications, changing the URL for the Plant-UML server as well as testing the notifications with a specific user target set inside the backend.

3. Versus other repeated learning flashcard apps

There exists a wide variety of flashcard apps for the last couple of decades. Some of them are desktop applications while others are pure web apps. This thesis is going to compare .cards with four popular alternatives, sorted by release date:

3.1. Mnemosyne

Released to the public by Peter Bientsman in 2006, Mnemosyne is a free flashcard app that offers installers for several operating systems, including Windows, Linux, Mac, and Android. It's open-source and using the AGLPv3 license, excluding the sync client, which has LGRLV3 [Pet21]. The software is still actively maintained by the original developer, with the recent stable release numbered 2.7.3 dating back to November 23. 2020.

Figure 3.1.: Mnemosyne user interface

The user has the option to choose between a variety of languages, after a successful installation inside the settings menu, including English and german. Text, font, and background

color can be also adjusted in an extra settings tab. Mnemosyne allows creating cards with rich content. Audio, video, HTML, javascript, flash, or latex can be used to create questions and answers. It does also include a google text-to-speech translation service. There is an option to download pre-made card sets, which cover languages, mathematics, science, traffic, and more. The software comes pre-installed with four different card types, however, it's possible to add more through the use of plugin:

- **Front-to-back only:** Consists of two sides, one question and one answer. The answer is hidden and needs to be filled in by the user.
- **Front-to-back and back-to-front:** Similar to the first option, however, Mnemosyne is going to create a sister card that flips the content of the sides around. Updates between both cards are being synced between each other. Both cards won't be scheduled on the same day to avoid artificial easy recalls.
- **Vocabulary:** Includes four sides: "Foreign word or expression", "Pronunciation", "Meaning" and "Notes". Two sister cards will be created: One "Recognition" card with the foreign word as the question and a "Production" card with the meaning as the question. The pronunciation and notes will be placed on the answer side of each card.
- **Map:** Indented with the use of graphical features, it allows users to create questions about the placement of locations. Three fields are required for this purpose: The name of the location, as well as a "Blank" and "Marked" map. The system will create a sister card that swaps the question between the blank and marked versions.

Each card can be assigned to individual tags for filtering, as well as an initial knowledge grade, ranging from 0 for "Yet to learn" to 5 for "Already learned".

The app offers four methods for learning: "Scheduled" → "Forgotten" → "New" and "Study new unlearned cards only" offer answer grading options ranging from 0 to 5, while "Cram all cards" and "Cram recently learned new cards" can only be answered with either "Wrong" or "Right". The number of active cards for a schedule can be filtered with card types and tags. It's possible to learn ahead of the next schedule if the current one got completed. The learning statistics can be viewed via a button in the menu. A bar chart is used for all categories except for the last one:

- **Schedule:** Shows in how many days the system will present the cards to the user again. Better known cards will be presented at a later date than cards that haven't been known by the user. This Graph is only used if the user learns cards with the grade rating mode.
- **Retention Score:** A graph history of many cards the user could remember in %.
- **Cards added:** A history of how many cards have been added to the system. This also accounts for cards that have been previously deleted.
- **Cards learned:** How many cards the user learned in the previous days.

- **Grades:** Show the grade given to each available card. Cards that haven't been answered will be highlighted as }Not Seen. This view is not used by the cram modes.
- **Eassiness:** The efficiency factor of all selected cards, displaying how easy each one is. The initial value starts at 2.5 (center) and can either lower if the card is difficult or higher if the card is easier.
- **Current Card:** Displays textual information about the currently active card including grade, easiness, thinking times, lapse, and next repetition.

The last thing to mention is the Sync-Client, which allows synchronizing all the content between different devices. This applies both to individuals and the exchange of data between them.

The .cards web app handles the scheduled learning through the use of the Woźniak learning algorithm. At the time of this writing, there are no options to review your learning progress in this mode. This feature will be added to a later date once the work on the Leitner algorithm has been completed.

3.2. SuperMemo

SuperMemo is a closed source learning software, which specialized in spaced repetition learning. It was created by SuperMemo World and Piotr Woźniak and is accessible on the web or can be installed as an app [WP21]. The user has the option to learn different languages or general knowledge. The content is offered through courses, which can be either free or charged for. The subscription model offers a monthly subscription which offers access to a select amount of premium courses or is given unlimited access by purchasing access at a higher price. As of April 5t 2021, it has 931 courses, with 712 being free.
The app provides a filter and search function to find the right content quickly and efficiently.

Users can create a course once they've signed in. Each course consists of a description, a category that has to be selected from a preset list, and multiple cards. A card contains two sides and is styled with basic features such as bold, list, and underline. In addition, it's possible to upload audio and image files, though these share a file size limit of 50 MB. These courses are meant to be for personal use and can be shared between friends. They won't show up inside the public course list and also don't have the option to be sold.

SuperMemo differentiates between new cards and repeated cards. Both are given a set maximum for the active workload session that the user is supposed to complete, with an option to continue learning once it has been achieved. The user is given the following three answer options: "I Don't Know" in red, "Almost" in yellow, and "I Know" in green. A workload session lasts as long as the user doesn't know the answers to the cards in his active queue. Not known cards will be repeated until they are known. The active progress of the queue is being highlighted as a bar over the answer options, with the colors correlating to

the submitted answers. Based on the answer, a cooldown in days is given to a card before the system can use it again for repeated learning.

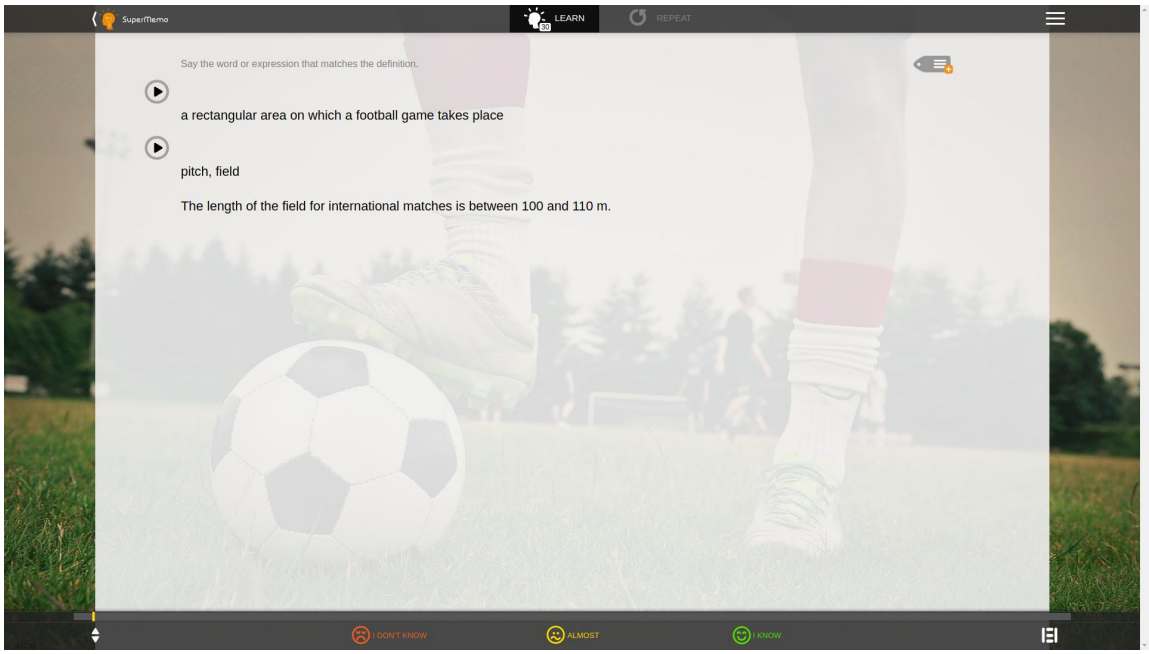


Figure 3.2.: SuperMemo user interface

The size of the workload can be set individually per course under the “Plan” tab. Here the system will also estimate the amount of time that the user might be spending per day to complete his workloads.

The learning history and statistics give the user just some basic information. The history displays the amount of new and repeated cards that the user learned through the use of a calendar, while the statistics only show a summary of the spent time learning and the number of cards that the user learned in between a set time window.

One exclusive feature is the dictionary system. It uses a SuperMemo exclusive dictionary which can be used to look up words for different languages. Each language supports audio assistance for the pronunciation of the searched term. Also, it can be used to translate text between different languages with the help of a Google Translate integration.

Compared to the .cards web app, SuperMemo has a more basic and sometimes more easy-to-use navigation. The placement of the filter navigation on the left side of the screen allows for better filter options, due to the lack of space concerns. The editor on the other hand is rather simplistic. It is only designed to create flashcard content with very basic features, while on .cards and Mnemosyne, it’s possible to even use HTML elements and diagrams. As for now, it’s also the only app with an integrated dictionary. The .cards web app used to have a translation integration with BEOLINGUS [Che21] and DeepL [Gmb21] in the past, but it had to be removed due to changes of the services API.

3.3. Anki

Released in 2006 by Damien Elmes, Anki is an Open-Source spaced repetition learning flashcard system [Dam21]. It is available on desktops for Windows, Linux, and Mac under the GN AGPL v3+ license in 48 languages and on Android / IOS on the GNU GPL v3 license with 14 languages. Services such as Anki-Web can be used as an alternative and are required for the synchronization feature.

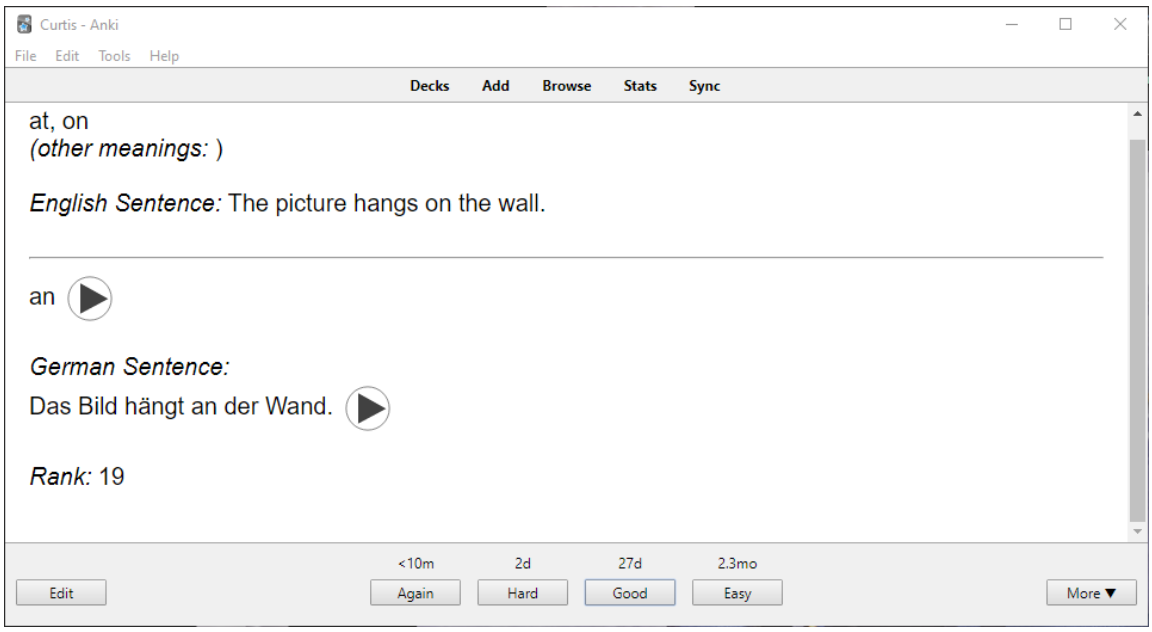


Figure 3.3.: Anki user interface

When starting the app, the user can choose between different profiles. Each profile contains its cards and learning progress. Each card is stored inside a deck, which is similar to the concept of card sets that are being used at the .cards web app. A card only contains one side for the answer and question, but the built-in editor allows customizing the template of each. The user can design a custom amount of text-editor fields that he wants to use for the content. Each field can support a multitude of editing options including HTML, Latex, Markdown, MathJax, images, or audio files. The order of the fields can be adjusted in the field creation dialog window and the card template editor. Tags can be used to categorize the content, as is the case in Mnemosyne.

The template editor also allows the addition of custom CSS attributes for the entire card. Anki also allows you to import cards from other users.

The algorithm differentiates between three categories, “New” highlighted in blue, “Learning” highlighted in red, and “To Review” highlighted in green. All three make up the total workload that has to be completed. An option is provided to set a threshold for each category.

The system offers the user the following response options that he can choose from while answering the cards:

- **Again:** The answer was too difficult and needs to be rechecked. The system is going to put the card back into the workloads active queue.
- **Hard:** Similar to the "Again" answer option, however, the card will first be shown again after several minutes, if the amount of remaining cards allows it.
- **Good:** The user doesn't need to see the card for today anymore and thus it can be put away with a minor cool-down of 1 day.
- **Easy:** The card doesn't have to be learned for a longer time and receives a cool-down of several days.

The stats window can display either the history of a single deck or be combined with a group of decks, named a "Collection". The review can be capped to a predetermined date of either a month, three months, half a year, or an entire year. The record stats contain information such as how frequently each answer option got utilized, how much time has been spent with learning the selection, or how many cards have been added.

The editor seems to have more setting options compared to the .cards web app. It is possible to create and save custom templates. Nevertheless, the display of the content is confusing. The front and back are displayed at the same time, which can lead to the user not being able to assign them correctly. Also, most of the setting options are hidden between many windows. For the answer options, times are highlighted so that the user is more likely to know what will happen to the card that's being presented if they click that button. This is a feature that should be considered for the .cards variant of Woźniak as well, through the use of tooltips.

3.4. Quizlet

Quizlet is a spaced repetition online platform designed for students and teachers [Inc21e]. It's owned by Quizlet. inc, which got founded by Andrew Sutherland in October 2005. The app has been made publicly available in 2007, with the recent update being released on 26 January 2021. It offers 18 different languages, including English and german. Account registration is mandatory to use the app. The registration can be either performed through social media logins from Google, Facebook, and Apple, or by creating a new account through the built-in registration system.

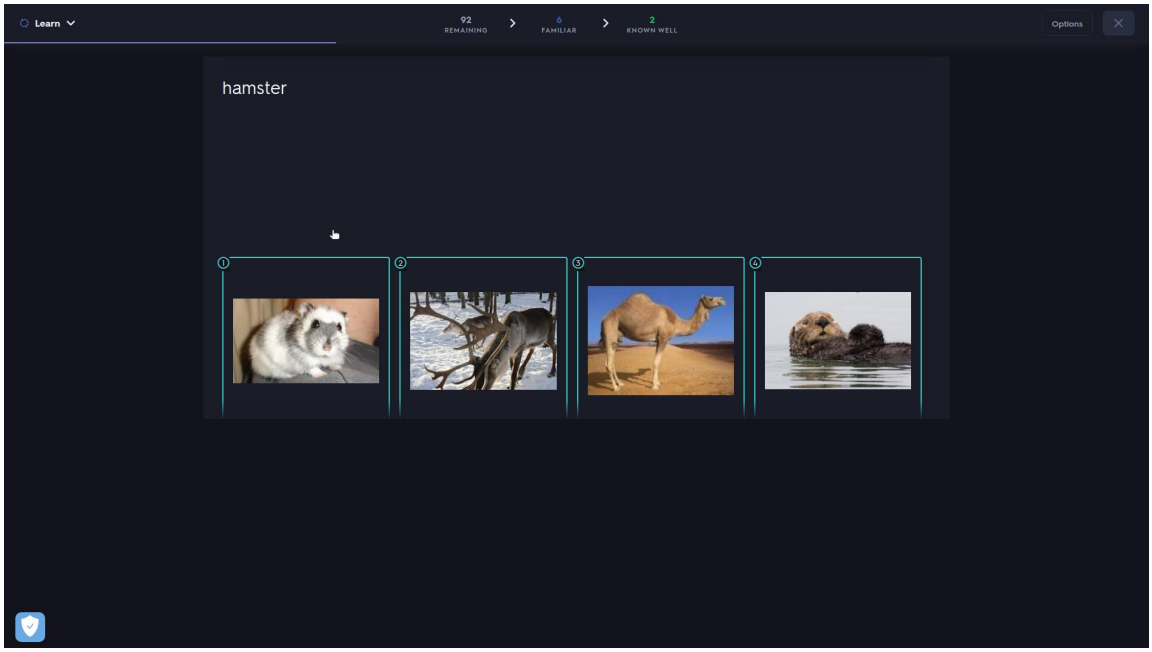


Figure 3.4.: Quizlet user interface

Use of the platform is free, however, the user will have to deal with advertisements and some features can be only accessed through a subscription model. Quizlet uses one account type for students and another one for teachers. Each account type has limited functionality in the free version and requires a subscription plan to use everything:

Quizlet Go: € 2.50 / month, Billed at €29.99 for each year

- Offline access
- Ad-free studying

Quizlet Plus: € 3.50 / month, Billed at €41.99 for each year

- Offline access
- Ad-free studying

- Rich text formatting
- Custom images and audio
- Create diagrams
- Scan documents
- Personalized study paths
- Progress insights
- Smart grading

Quizlet Teacher: € 3.99 / month, Billed at €47.88 for each year

- Ad-free studying
- Multiple-choice options, custom images/audio, rich text formatting, and diagram support for cards
- Tracking of student progress
- Custom teams, audio, and switching sets without changing groups in Quizlet Live
- Create an unlimited number of classes
- Scan in documents as sets

Cards are stored inside study sets, which can be organized through the use of folders. Each study set can be assigned to a class to create different content combinations. A study set consists of a title, an optional description, and at least two cards. Each card has a term, definition, and an optional image. Additional functionality such as voice recordings or multiple-choice options requires a "QuizletTeacher" subscription.

There are eight study modes, of which three are designed around gamification. One of the gamification modes, named live, requires a teacher account.

- **Flashcards:** The user is presented with all the cards in the set. The system automatically shows the card's response after a few seconds and moves on to the next card. This automatic process is optional and the order of the cards can be shuffled.
- **Learn:** The user is presented with all the cards from the set or class. A card must be answered correctly at least twice for it to be considered learned. The user is presented with four answer options based on the answers from the set. As an option, you can filter by answer type "Flashcard", "Multiple-choice", "Written answer", and "Fill in the blanks". Additionally, you can choose whether the front and back of the card are used as a question.

- **Write:** The user must write the answer in a text field and submit it. Multiple answers can be added by separating them with a slash (/), comma (,), or semicolon (;). The settings can be used to define whether only one correct answer is required for multiple answers to be marked as correct. If there are grammatical errors in an answer, they're going to be highlighted. During a round, each card is presented once and it is noted which ones were answered correctly and which ones were answered incorrectly. At the end of the round, the user can start a new one with all the cards that have been answered incorrectly.
- **Spell:** The cards are divided into small groups and used as rounds. The user is being dictated the answer through audio and has to write it down. The next card is not presented until the answer is correct. When a card has been answered correctly twice, it is marked as learned. After each round, a status report is displayed about the previous round and the overall score.
- **Test:** A set number of questions are presented. These consist of a combination of written, matching, multiple-choice, and true and false questions. The number of answers and the types to be used can be adjusted in the settings. After the user has filled in all the fields, he can submit them as a single answer. The score is calculated at the end from the number of correct answers.
- **Match:** A gamification variant. Here the user is presented with the front and back of a small selection of cards and must match them correctly by dragging them to each other. Correctly matched content is removed from the board and incorrect answers will reset it to its initial position. During the learning process, a timer runs that records the elapsed time in 100-millisecond intervals. When all elements have been matched, a leaderboard is being displayed, which allows the user to compare his time against that of others.
- **Gravity:** In it, the user has to save a planet from falling asteroids. Before starting the game, the difficulty level, as well as the answer type, must be selected. The player can remove the asteroids by entering the correct answers in a text box. If an asteroid reaches the planet, the correct answer is displayed and must be confirmed by typing it in. You get points for removing asteroids before they reach the planet and a point deduction for answering the question incorrectly. Not intercepted asteroids later appear in red, which earn the player bonus points. After some asteroids are removed, the player's level increases and they move to another planet. After each level, the speed and number of asteroids increase. The game ends when the player fails to intercept several asteroids. Here, as in the game "Match", a leaderboard is displayed with which you can compare your score with that of other participants.
- **Live:** This game requires several participants and one person who has a teacher role. The players are divided into small teams. Each member of a team is asked the same question but shown different answers. Only one member has the correct answer, this forces communication below the team to find the correct answer. The teacher's screen displays the team scores. If the question is answered correctly, the score of the team increases by one point. If a team answers a question incorrectly, the score is reset

to 0. This prevents a team from winning by trying to answer quickly. After a team wins, the teacher can evaluate each team's score or start a new round.

The app appears as a combination of .cards and arsnova.click [Tea21a]. It is possible to use it only for learning the content, as well as to use it playfully in a team, which is presented live. Quizlet offers many options for content creation, but most of the elements of it are hidden behind a paywall. Yet, it is generally the most used and popular app at the moment, judging by the content statistics.

3.5. Summary

The following is a summary of the previous comparisons: Compared to the other web apps,

Features	cards 3.14	Mnemosyne	SuperMemo	Anki	Quizlet
OS	Web app	Windows, Linux, Mac, Android, iOS	Web app	Windows, Linux, Mac, Android, iOS	Web app
Open Source	x	x		x	
Mobile First	x	(x)	x	(x)	x
Public content	Around 350 sets	Around 384 decks	931 courses with 712 being free		
Languages	1	30	15	48	18
Content styling	Markdeep, MathJax	HTML, Flash, JavaScript	Text, Images, Video	HTML, Template, LaTeX, MathJax	Text, Images, MC,* Rich Text* * Requires a subscription
Translation services			Google-Translate		
Plugin support		x		x	
Learning algorithms	Leitner, Woźniak	Woźniak	Woźniak	Woźniak	Unspecified
Paid content	None	None	Content subscription	None	Account subscription

Table 3.1.: App comparison summary

.cards is the only one that is open source. Mnemosyne and Quizlet are more specialized in desktop or mobile apps and therefore, except for the mobile clients, do not have a fully responsive design. In the area of available languages, .cards is lacking compared to the competition: only german is offered here, while the other apps support 15 - 48 languages. The content of cards can be designed in detail thanks to the use of Markdeep. Other apps also support many formats, but among the web apps, only simple content is possible as long as the user does not pay anything for it. SuperMemo is currently one of the only apps that offer direct translation. However, translation services could be installed later in

Mnemosyne and Anki thanks to plugin support. The learning modes specialize in Leitner in .cards, and Woźniak in the other application. Exactly which algorithm is used in Quizlet could not be determined.

4. New Features and improvements

Since the start of the development project in summer 2020, many features have been implemented to automate the retrieval of statistics and the ratings of answers. Furthermore, the system now increasingly checks whether users adhere to the instructor's specified time. These goals have been completed in their first version and are already in production. However, during the editing phase of this thesis, concepts have already been developed to remove the limitations of the initial implementations.

4.1. Leitner-Bonus

At the beginning of the development of .cards version 3.0, an early version to award bonuses was implemented. This variant allowed a lecturer to view the current learning status of his students. In the first implementation, only the names of the students were displayed, as well as the positions of the cards in the boxes. Additionally, there was the possibility to export this information as a .csv file. This seemed to be sufficient at the beginning, but it turned out later that some students did not use the system as originally intended. Some students always just quickly answered unlocked cards, so that these moved up to a higher box, without learning the actual content from them. Because of this, it became clear that detailed insight into the learning behavior is needed to better assess whether the student has intensively engaged with the topic that he's supposed to learn. For this, more statistics have been introduced in 2020 so that the lecturer can easier check the learning behavior of his students. The display of the list takes place through a call to a Meteor method, which

Benutzer	Erinnerungsdienst	Lernbeginn	Letzte Aktivität	Bearbeitungszeit	Ø Antworzeit	Box 1 [1]	Box 2 [3]	Box 3 [5]	Box 4 [7]	Box 5 [14]	Gelernt	Bonuspunkte	Protokoll	Lernstand	Verwalten
Bonus-Teilnehmer 1	E-Mail & Push	1. Februar 2021	27. März 2021	62 Stunden und 30 Minuten	2 Minuten	1	0	9	154	80	164 [40 %]	80 %			
Bonus-Teilnehmer 2	E-Mail & Push	1. Februar 2021	30. März 2021	66 Stunden und 4 Minuten	4 Minuten	51	172	56	71	47	11 [3 %]	5 %			
Bonus-Teilnehmer 3	E-Mail & Push	2. Februar 2021	31. März 2021	17 Stunden und 9 Minuten	22 Sekunden	0	0	4	7	90	307 [75 %]	100 %			
Bonus-Teilnehmer 4	E-Mail & Push	1. Februar 2021	31. März 2021	19 Stunden und 57 Minuten	31 Sekunden	0	3	11	97	137	160 [39 %]	78 %			
Bonus-Teilnehmer 5	E-Mail & Push	1. Februar 2021	30. März 2021	36 Stunden und 52 Minuten	2 Minuten	1	100	78	78	85	66 [16 %]	32 %			
Bonus-Teilnehmer 6	E-Mail & Push	1. Februar 2021	30. März 2021	60 Stunden und 14 Minuten	3 Minuten	1	77	175	85	56	14 [3 %]	7 %			
Bonus-Teilnehmer 7	E-Mail & Push	1. Februar 2021	30. März 2021	24 Stunden und 17 Minuten	20 Sekunden	21	66	48	82	66	125 [31 %]	61 %			
Bonus-Teilnehmer 8	E-Mail & Push	1. Februar 2021	15. März 2021	33 Stunden und 36 Minuten	1 Minute	93	176	42	47	46	4 [1 %]	2 %			
Bonus-Teilnehmer 9	E-Mail & Push	1. Februar 2021	25. März 2021	72 Stunden und 1 Minute	2 Minuten	0	3	7	44	185	169 [41 %]	83 %			
Bonus-Teilnehmer 10	E-Mail & Push	1. Februar 2021	15. März 2021	19 Stunden und 27 Minuten	1 Minute	52	195	46	72	38	5 [1 %]	2 %			
Bonus-Teilnehmer 11	E-Mail & Push	1. Februar 2021	23. März 2021	12 Stunden und 57 Minuten	49 Sekunden	150	119	71	37	20	11 [3 %]	5 %			
Bonus-Teilnehmer 12	E-Mail & Push	1. Februar 2021	30. März 2021	59 Stunden und 48 Minuten	1 Minute	2	4	23	111	108	160 [39 %]	78 %			
Bonus-Teilnehmer 13	E-Mail & Push	5. Februar 2021	25. März 2021	25 Stunden und 41 Minuten	29 Sekunden	76	163	55	58	50	6 [1 %]	3 %			
Bonus-Teilnehmer 14	E-Mail & Push	1. Februar 2021	29. März 2021	31 Stunden und 38 Minuten	42 Sekunden	0	0	3	8	146	251 [62 %]	100 %			
Bonus-Teilnehmer 15	E-Mail & Push	1. Februar 2021	10. März 2021	35 Stunden und 54 Minuten	1 Minute	63	9	152	86	98	0	0 %			

Figure 4.1.: .cards bonus learning status overview

returns a fully assembled list statistic. As a first approach, the use of Meteor subscriptions

and the calculation of the statistics on the client-side was attempted, but due to the amount of data in the modules and card sets used by the THM, this turned out to be impractical. The loading of the statistics would end up taking several seconds on the client-side. Due to the synchronization of subscriptions and lower processing speed.

4.1.1. Reworked database schema

To be able to store more statistics, the database schema had to be adapted. New MongoDB entries were needed for the individual activation days of a student, as well as the submitted answers and processing times for each card. To preserve the information from older learning events, these new objects were given a session ID that increments when a new learning workload is created, either by joining a bonus or starting a private learning session.

```
{
    "_id" : STRING, // Primary key.
    "cardset_id": STRING, // Foreign key of the cardset.
    "user_id": STRING, // Foreign key of the user.
    "session": INTEGER, // The session group for the date.
    "isBonus": BOOLEAN, // Part of a bonus?
    "missedDeadline": BOOLEAN, // Triggered by deadline?
    "resetDeadlineMode": INTEGER, // How was the reset performed?
    "wrongAnswerMode": INTEGER, // Where to move wrong answers?
    "notifications": OBJECT, // Notification history.
    "createdAt": DATE, // Date at which the object got created.
    "pomodoroTimer" : OBJECT, // Pomodoro settings.
    "strictWorkloadTimer": BOOLEAN, // Is a strict timer mode active?
    "timer": OBJECT, // Timetracking for log modal.
    "learningStatistics": OBJECT, // Learning statistics for this date.
}
```

Listing 4.1: Leitner 1.0 activation day object

To ensure that this information is still available when a user account is deleted, the system has been extended to anonymize all of the user's old learning content. The MongoDB field user_id, is replaced with a new field called user_id_deleted, which is given an incremental global number so that the content can still be assembled correctly.

4.1.2. Improved bonus overview

Now that more information about learning behavior is being stored, it's also possible to integrate this into the overall overview. These include the student's last activity, based on when they last answered a card, as well as the total completion time and average response time. These are updated each time a question is answered on the server and the final result is stored in new MongoDB fields created to save processing time.

```
"learningStastics": { // Stored in leitnerTasks, workload and cardset.
  "answerTime": {
    "median": INTEGER,
    "arithmeticMean": INTEGER,
    "standardDeviation": INTEGER
  },
  "workingTime": {
    "sum": INTEGER
  }
}
```

Listing 4.2: Leitner 1.0 learning statistics object

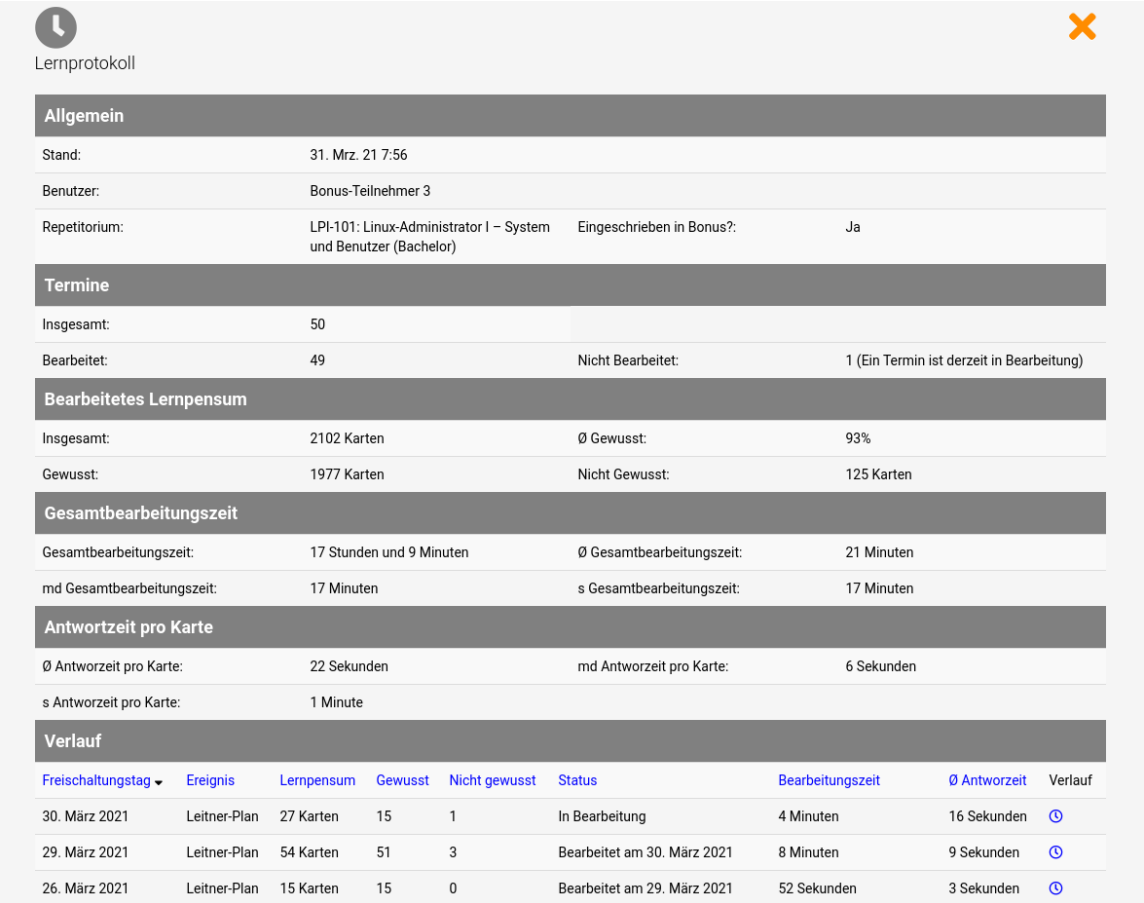
For the calculation of the active learning time as well as for the answering time statistics, the math library math.js [jon21] is used. This library provides functions for calculating the arithmetic mean, median, and standard deviation. The results are stored in a millisecond format so that these can be displayed with moment.js [Fc21a] in a more human-readable format.

To better verify that a student is also using the e-mail and push notification alert service, icons are now displayed for these if they have been activated. The display of the e-mail had a greater relevance for the CAS login service because here the students had to manually insert their e-mail address inside their profile, which some students abused by inserting meaningless addresses so that they avoid receiving daily notifications. With the new login system, however, this data is now inserted automatically.

Lastly, a sorting function was implemented for most of the columns. For this, a sorting function of Underscore.js [Jer+21] is utilized, which receives the list sent by the server and sorts it according to the active sorting criterion, which is stored in a Meteor session, so that the change will be reactive. The default sorting for the individual displays is stored in a config file.

4.1.3. History modal

The first information modal view for a single user. It shows basic information such as the user data, how often an active workload has been completed, and information about the total learning time as well as answer times per card. In the lower area, a list is displayed, which shows one entry per activation day. An activation day takes place whenever the student learned his active cards and the server can assign new cards during the execution of the cronjob. The event field shows whether the activation day was triggered by the completion of the last learning workload or by exceeding the workload deadline. Via the button "History" the user can open another modal, which contains further information about the selected activation day.


Lernprotokoll

Allgemein			
Stand:	31. Mrz. 21 7:56		
Benutzer:	Bonus-Teilnehmer 3		
Repetitorium:	LPI-101: Linux-Administrator I – System und Benutzer (Bachelor)	Eingeschrieben in Bonus?:	Ja
Termine			
Insgesamt:	50		
Bearbeitet:	49	Nicht Bearbeitet:	1 (Ein Termin ist derzeit in Bearbeitung)
Bearbeitetes Lernpensum			
Insgesamt:	2102 Karten	Ø Gewusst:	93%
Gewusst:	1977 Karten	Nicht Gewusst:	125 Karten
Gesamtbearbeitungszeit			
Gesamtbearbeitungszeit:	17 Stunden und 9 Minuten	Ø Gesamtbearbeitungszeit:	21 Minuten
md Gesamtbearbeitungszeit:	17 Minuten	s Gesamtbearbeitungszeit:	17 Minuten
Antwortzeit pro Karte			
Ø Antwortzeit pro Karte:	22 Sekunden	md Antwortzeit pro Karte:	6 Sekunden
s Antwortzeit pro Karte:	1 Minute		

Verlauf

Freischaltungstag	Ereignis	Lernpensum	Gewusst	Nicht gewusst	Status	Bearbeitungszeit	Ø Antwortzeit	Verlauf
30. März 2021	Leitner-Plan	27 Karten	15	1	In Bearbeitung	4 Minuten	16 Sekunden	
29. März 2021	Leitner-Plan	54 Karten	51	3	Bearbeitet am 30. März 2021	8 Minuten	9 Sekunden	
26. März 2021	Leitner-Plan	15 Karten	15	0	Bearbeitet am 29. März 2021	52 Sekunden	3 Sekunden	

Figure 4.2.: .cards user history with global stats

4.1.4. Log modal

A detailed view of the selected activation day. It contains the same global information as the history modal. In the list, all answered cards of the selected activation date are listed. The content column combines the title of the card, as well as the content of the question side, referenced by a hyperlink. The answer time contains the total learning time, from viewing the question to sending the answer to the server.

In its current implementation, pauses are also included, for this a better approach to capture the response time will be presented later. The learning state shows the box in which the card is located after the answer. The pre-sorting is based on the submission of the answer. The newest answers get listed first.

Verlauf

Inhalt	Karteityp	Antwort	Antwortzeit	Lernstand	Abgabe
Zugriffsrechte: 001: Zugriffsmodus	Glossar	Gewusst	7 Sekunden	Gelernt	31. Mrz. 21 7:51
Zugriffsrechte: 002: access control lists	Glossar	Gewusst	11 Sekunden	Gelernt	31. Mrz. 21 7:51
Zugriffsrechte: 003: SUID Bit	Glossar	Gewusst	23 Sekunden	Gelernt	31. Mrz. 21 7:52
Zugriffsrechte: 004: SGID Bit	Glossar	Gewusst	17 Sekunden	Gelernt	31. Mrz. 21 7:52
Zugriffsrechte unter Linux: 001: Setzt Dateiattribute für ext2 und ext3 Dat...	Anweisung	Gewusst	3 Sekunden	Gelernt	31. Mrz. 21 7:52
Zugriffsrechte unter Linux: 006: Erlaubt die Manipulation von ACLs.	Anweisung	Gewusst	12 Sekunden	Gelernt	31. Mrz. 21 7:52
2. Prozesszustände: !!!INFO Lernziel 006: Erkläre wie man mithilfe ...	Lerneinheit	Gewusst	4 Sekunden	Gelernt	31. Mrz. 21 7:52
Verwaltung von Prozessen: 004: Startet ein Programm so, dass es immun gege...	Anweisung	Gewusst	1 Minute	Gelernt	31. Mrz. 21 7:53

Figure 4.3.: .cards user log

4.1.5. Stats modal

Shows the current learning state of each card. This modal was already implemented in Summer 2016 with the introduction of .cards version 3.0, but in the following years, it was extended with new functionality. Information about the user is shown first, then comes a dropdown menu to filter the cards. This menu is available for repetitories and allows to display of individual referenced card sets. The cards are sorted by box and difficulty level. The labels of the boxes contain the internals from when the cards can be resubmitted.

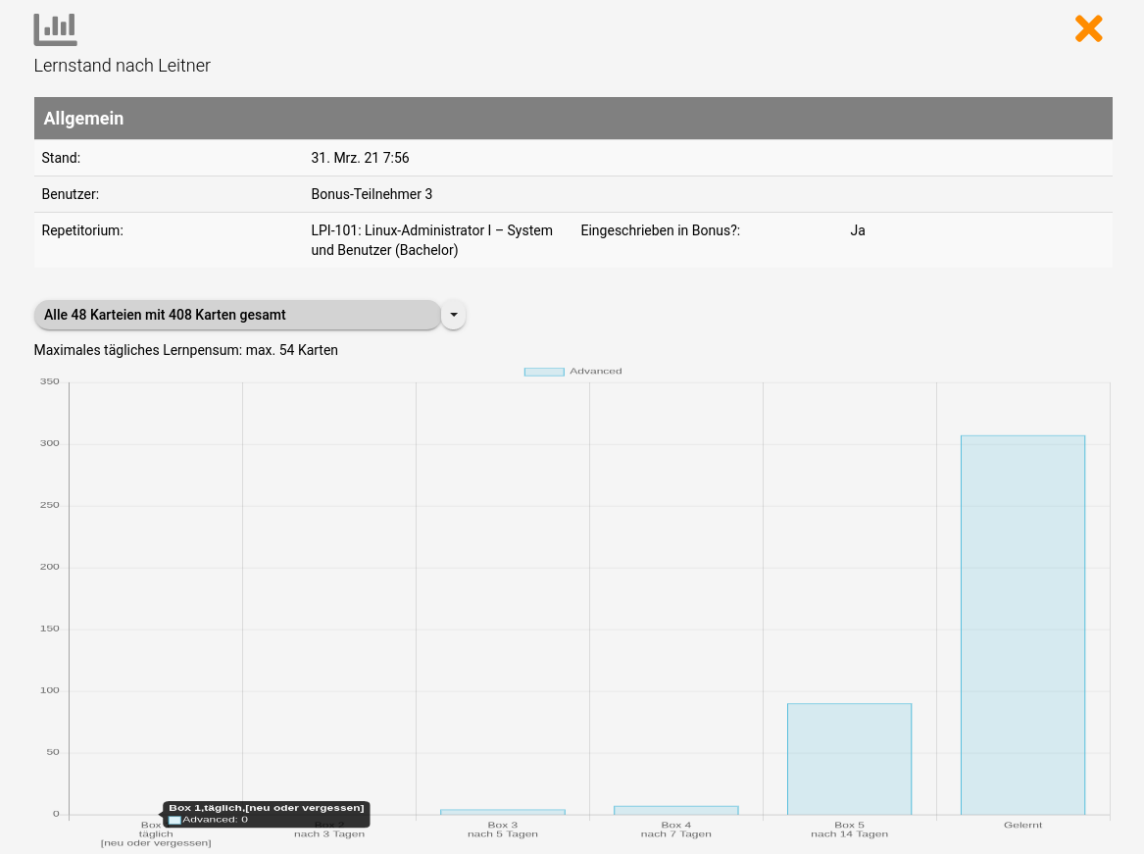


Figure 4.4.: .cards user stats

4.1.6. Upcoming improvements

Unfortunately, all features couldn't be implemented in the planned timeframe of the development project. Below is a list of all incompleted features, the 2.0 database schema, and the listing of the most difficult cards in a learning phase. These issues will most likely be completed around summer 2021.

4.1.6.1. 2.0 database schema

The database schema currently in use allows a lot of the old information to be saved, but some critical information is still lost when the user either deletes their profile. Furthermore, there is a problem that due to poor planning in older versions, some items cannot be saved with the current format, including settings from bonus learning phases that have already ended. At last, some labels have changed over time so that they are no longer appropriate for their intended purpose.

For this reason, it was decided to revise the current schema and migrate the legacy data to it. The new schema also provides the basis for further functionalities such as determining the most difficult card within a bonus learning phase.

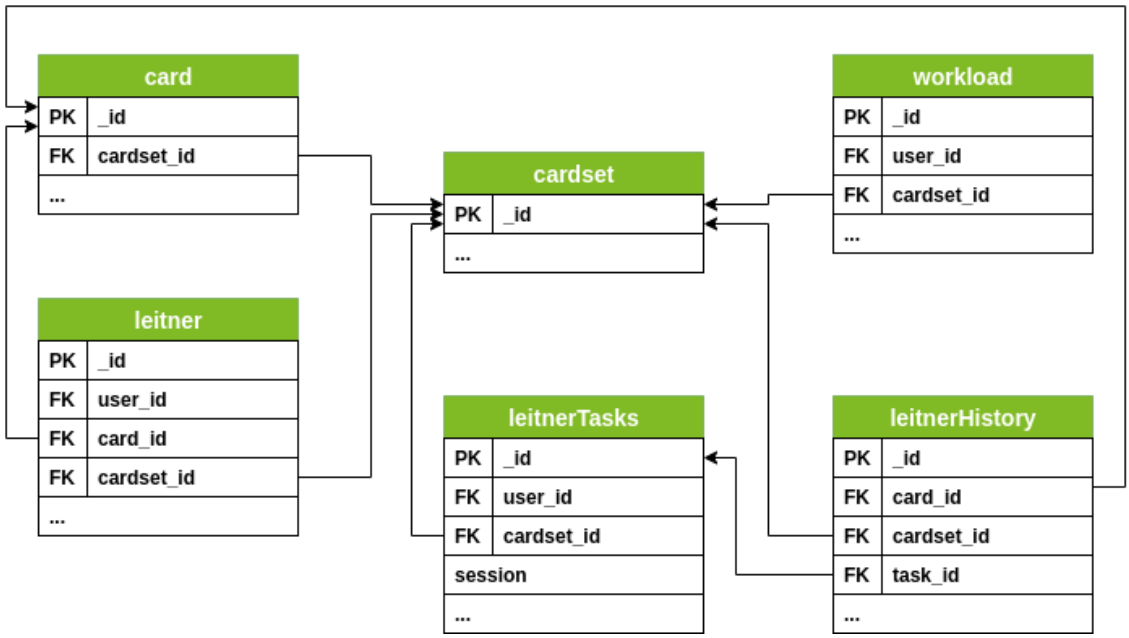


Figure 4.5.: .cards leitner 1.0 MongoDB document relations

The information for the learning phase of individual cards is currently stored in a collection called “Leitner”. As soon as this phase got finished, these objects are deleted from the database. To retain them, each object is given a new ID that references the user’s workload. Another problem is that the information about the learning phase is stored in the object of the card set. For a card set to have multiple learning phases, this must be moved out to a new collection, which all other Leitner elements reference for filtering.

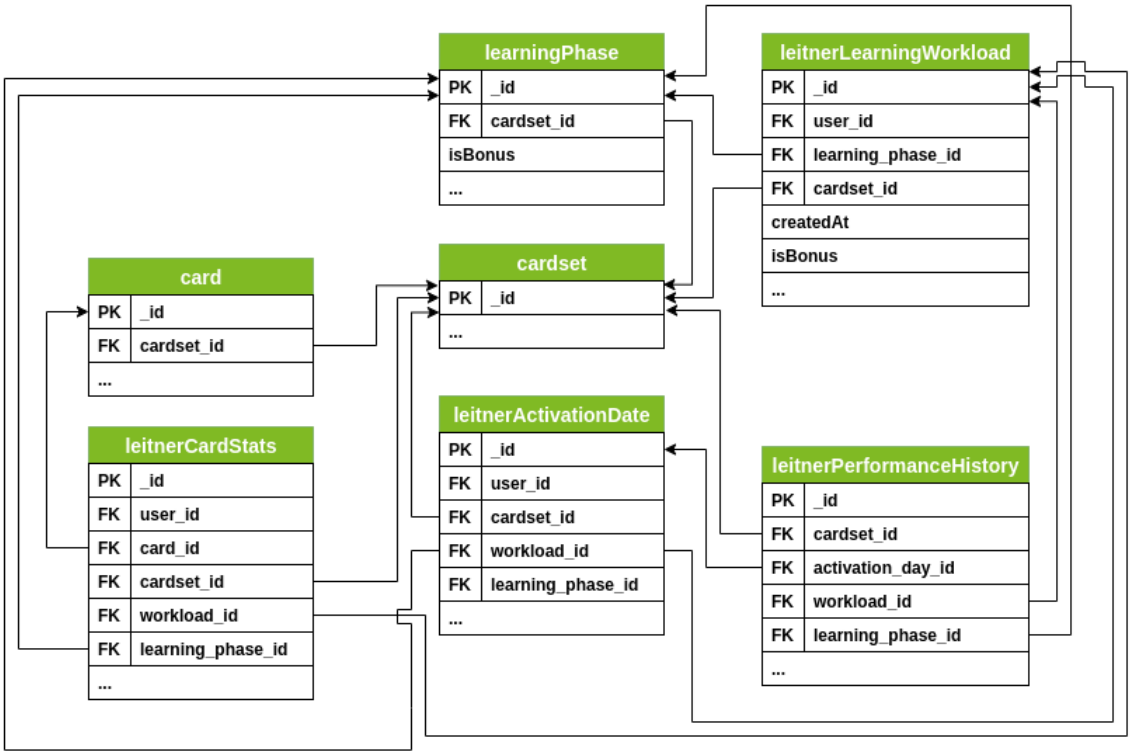


Figure 4.6.: .cards leitner 2.0 MongoDB document relations

The adaptation of the collection names is rather unproblematic with the help of IDE's like Webstorm, the addition of new MongoDB fields causes a bigger problem. To simplify the sorting in the future, a "createdAt" field is now additionally stored for the "Workload" and new "LearningPhase" collection, as well as a name for the learning phase, which is later used in a dropdown list to switch to already completed bonuses.

4.1.6.2. Most difficult card

Assumes the implementation of the 2.0 database schema. After each client answer, the server calculates the total number of correct and incorrect answers and updates them in two collections that contain information about the card within that learning phase:

- **LeitnerUserCardStats:** Contains information of the card assigned to a single user.
- **LeitnerBonusCardStats:** Contains summarized information from leitnerUserCard-Stats within a bonus.

. In addition to logging the incorrect and correct answers, a value must also be included which will be used for weighting the sorting order. A card with the same correct/wrong ratio but a bigger total of answers given has a higher weight than a card with fewer answers.

```
"difficulty": { // Stored in leitner.
  "totalAnswers": INTEGER,
  "wrong": INTEGER,
  "weight": INTEGER
}
```

Listing 4.3: Difficulty rating object

The user is provided with a modal containing a list of all the cards answered so far, sorted by difficulty. This is offered in two versions: A view for the user about his learning status and a version for the lecturer to determine the most difficult card within the bonus phase. Through this logging, it is also possible to find out which card has never been presented by the system before.

4.1.6.3. Improved workload view

Currently, we display each individual learning phase item that the user is actively learning with the topic pool layout. This has a limitation that due to lack of space not much information can be displayed such as whether the user is in this learning phase within a bonus.

Figure 4.7.: Current layout of workload items

Since a user usually has only a small number of active learning phases, it would be recommended that the design be revised to provide multiple columns of information for each learning phase. A similar design concept has been implemented previously for another feature: the display of bonus points for the submission of transcripts.

Hogan, Nansi	
Eingereichte Mitschriften:	8
Noch nicht bewertet:	0
Akzeptiert:	7
Abgelehnt:	1
Gesammelte Sterne:	21
Ø Sterne:	3
Letzte Abgabe:	Usability / UX | 17. Januar 2020
Erhaltene Bonuspunkte:	4 %

Figure 4.8.: New layout for a single workload item

This gives the user the ability to see if it is a learning phase for private learning or a bonus. Additionally, there is now enough space to display the number of Leitner and Woźniak cards. By implementing the 2.0 database schema, it will also be possible to display already completed learning phases within the route. For this, one can again orientate himself on the route of transcripts, which offers a button for selecting the route below the main navigation. In this case, it would be buttons for " Active" and "Completed".

4.1.6.4. Strict timer

At this moment, the guideline values that the lecturer can set for the Pomodoro timer are only optional for the users. If a user finishes his learning phase workload before the guideline goal was reached, then he has still completed his workload. According to statistics from previous events, the learning time for many users is a fraction of the guideline.

2. März 2021	Leitner-Plan	50 Karten	48	2	Bearbeitet am 3. März 2021	9 Minuten	10 Sekunden
1. März 2021	Leitner-Plan	50 Karten	48	2	Bearbeitet am 2. März 2021	18 Minuten	22 Sekunden
26. Februar 2021	Leitner-Plan	50 Karten	48	2	Bearbeitet am 28. Februar 2021	13 Minuten	16 Sekunden

Figure 4.9.: Example of an user finishing too early

What the Strict mode is supposed to achieve is that the times of the Pomodoro timer are respected. This means that if, for example, the timer has been set to four Pomodori (Four working phases with two small and one large break), then the learning phase workload will not be credited until the timer has completed them. How the server can find out what the status of the timer of each client is, is discussed in the section " Pomodoro Timer". There are two approaches how to implement this mode:

1. The server checks whether the Pomodori goal has been reached after the last card answer has been submitted. If not, then the server will add new cards to the user's

active queue. This continues until the number of Pomodori is reached or the server has no more cards that he can send to the user.

2. A minimum average waiting time is being calculated, based on the number of cards selected for the active workload and the Pomodoro timer setting. The user can only send his answer to the server once this waiting time has expired. Here a countdown can be inserted inside the answer button.

Both variants should ensure that the user is more likely to stick to the given learning time on each activation day.

4.1.6.5. intermediate test mode

The learning status of the students needs to be verified through the use of a test. For this purpose, a card set is used, which performs this live with a selection of questions. Right now, the algorithm is configured in a way that in a 24 hours interval a cronjob is triggered, which assigns new cards or checks if a bonus learning phase is over. The end of a learning phase can be defined by selecting a date. But, for an intermediate test to be automated, it is necessary to provide a time specification in minutes. There must be a new entry for the settings of a bonus learning phase, which adjusts the form when checking for an intermediate test.

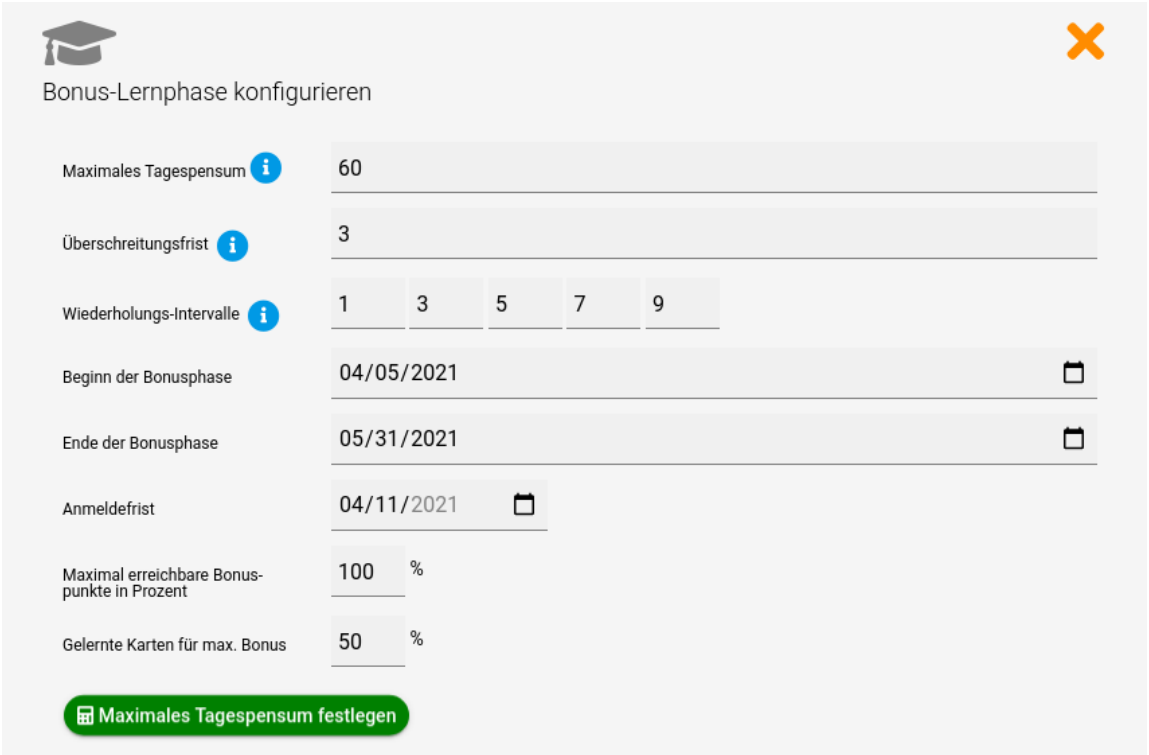


Figure 4.10.: General settings of a leitner bonus form

In this mode, the calendar options, deadline period as well as intervals, and Leitner simulator are hidden. After creating a learning phase, a new entry is stored inside a MongoDB collection, which is scanned every minute by a cronjob. The intermediate test is automatically terminated as soon as the cronjob detects that the time has expired. A timer is also displayed in the user view so that the person knows how much time is left to complete the test. This time must be displayed independently of the Pomodoro timer, otherwise, sessions longer than 60 minutes cannot be displayed.

4.2. .cards Login

Over the past years, users have been able to log in through the CAS service or use one of the social media logins. The CAS service was a requirement for students to participate in the bonus learning phases because individuals with this login are registered as edu users. A bigger problem with the currently available CAS package [Tea19] and the CAS implementation at the THM is, that important information such as the first name, last name, and e-mail address are not transmitted directly through this service. Instead of creating a custom CAS package that can retrieve this information during registration, it was decided to use the built-in registration function of Meteor with the help of the " meteor-useraccount" [Sof16] package. Support for the Bootstrap 3 framework used in .cards [Inc13] and the client-side router FlowRouter [Ltd17] are also already guaranteed.

4.2.1. Internal login

Most of the functionality for registration is already covered by the provided packages. The user can log in to an already existing account, register a new one, or request an already sent verification e-mail or password again. When registering, the user ID, last name, first name, e-mail, and password are required, and the user ID and e-mail must not already exist in the database. The package offers the possibility that the user must repeat his password for confirmation.

To ensure that only users with a university e-mail address can register on our installation, the input field has been equipped with a regexp filter. The list of allowed email addresses can be modified in the repositories of the project. The current location of this list is under `imports/config/serverStyle/global/config.js`.

After registration, the user receives a confirmation email with a verification code. This is sent in its default form as a plain text file only. To make the emails generated by the useraccounts package match the e-mails, for the reminder service of a learning phase, the package " cmather:handlebars-server" is used [CR14]. This allows us, from the server-side, to use the message as Meteor Handlebars [Yeh21] in an email HTML template. The template is located in the folder `server/user-mail.handlebars`. After verifying the email address, the user is redirected to the front end.

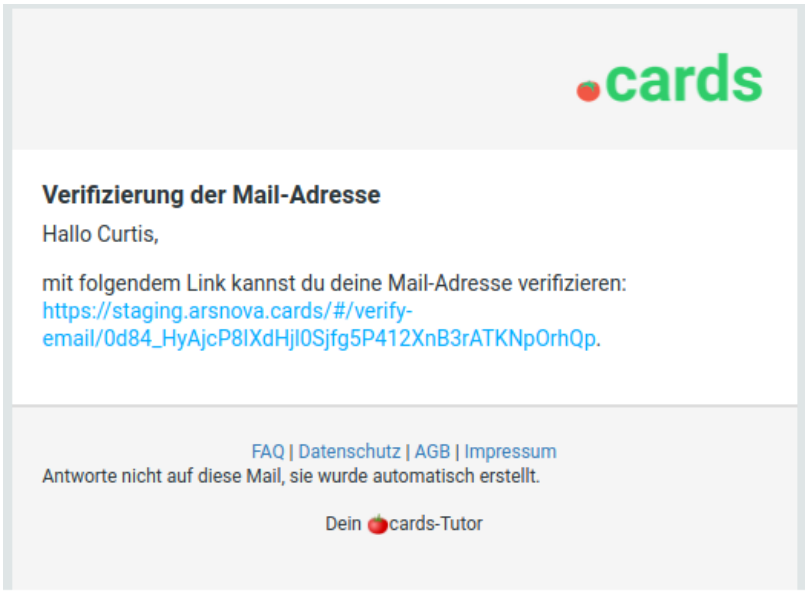


Figure 4.11.: .cards registration e-mail

There were some problems during the implementation in the project phase, which could not be fixed without modifying the user-account packages themselves. Firstly, .cards uses an i18n package for the translations, which is currently not applicable to the internal login [Lim16]. However, i18n support for the internal login seems to be under development ([Sof16]. Another problem is that the behavior of some input fields cannot be customized. On the one hand, it is possible to check the input, but the integration of a repetition of the e-mail address with the events provided by the package does not seem to be possible. The same problem also affects the password field when the user wants to reset it. Finally, there were some problems concerning the styling of the login. While the team provides a package for this that uses bootstrap styling, this does not include the verification confirmation and password reset pop-ups.

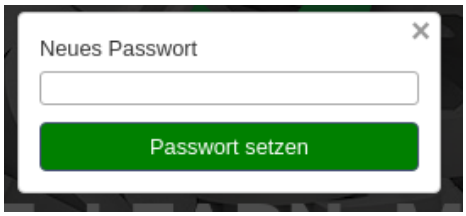


Figure 4.12.: .cards password reset pop-up

4.3. Multiple Choice

Before the development phase, the system had only one answer option for the Leitner learning mode: The user could decide for himself whether he knew or did not know the answer. This learning option has the disadvantage that the person can lie to himself to still get the card into the next higher labeled box. For this reason, .cards needed a system to check the answer. Multiple-choice is a simple and fast solution to implement such a system, as the server only needs to confirm if the answers that the user checked match the ones stored in the database. Before work on the implementation began, it was considered to include a single-choice system, but this turned out to be superfluous since such questions can also be created with multiple-choice. The implementation took place in two phases: Phase one revolved around the integration of the editor so that those responsible for the content in the linux.cards server installation could start creating content. Phase two was about the proper display in the presentation mode, and the communication between client and server in Leitner the learning mode.

4.3.1. Storage format

The multiple-choice answers are part of the card collection in MongoDB. Each content side of a card consists of a string: this includes the MongoDB fields “front”, “back”, “lecture”, “hint”, “top” and “bottom”. Since multiple-choice questions can consist of one question, multiple answers, and hints, this existing format turned out to be inappropriate. As a solution, the card documents have been equipped with a new field containing an object for the answers and questions. The question is stored as a string, while the answers are stored in an array of objects. Each of these objects contains a string for the answer and the hint. Another integer array called “rightAnswers”, stores the index of the right answer from the previous array. Finally, there is a boolean value “randomized” which signals the presentation mode whether the order of the displayed answers should be mixed and another boolean “enabled” which enables the answer for the learning mode.

```
"answers": { // Stored inside the card.
  "rightAnswers": [INTEGER], // ID of the answers that are correct.
  "randomized": BOOLEAN, // Shuffle the content
  "content": [OBJECT] // Contains a String for the answer and explanation.
  "question": STRING // The question before the answers are displayed.
  "enabled": BOOLEAN // Use the content for the learning mode.
}
```

Listing 4.4: Card answers object

Not every card type can automatically use the multiple-choice system, this must be stored beforehand in a config file in the repository. Two settings must be adjusted for this: First, the card type must be specified, which the multiple-choice system should use as a response option. To do this, simply add the ID of the card type to an existing variable. This can be set in the variable “cardTypesWithAnswerOptions” which is located under `imports/config/cardTypes.js`. If the pages that display the questions should not use

any other additional content, the card type must also be set in the variable “cardTypesWithNoSideContent”.

```
// MC Questions
let cardTypesWithAnswerOptions = [11]; // 11 = card type: quiz
let cardTypesWithNoSideContent = [11];
```

Listing 4.5: Card type server settings for MC

Finally, in the metafile for the sides of the card type, it must be specified on which side the question and the answer are located. For this, there is an array named “cardTypeCubeSides”, in the same file, which contains an array 2D array of objects, each object in one of these arrays corresponds to the side of a card. The side with the question needs the value `"gotQuestion": true`, and the sides with the answer need the value `"isAnswer": true` and `"isAnswerFocus": true`, the latter is necessary so that the system knows which of the sides to jump to when the answers are shown since there can be multiple answer sides for the card type.

```
//20: Quiz
  [
    {
      "contentId": 1, // Which string to access for the content.
      "defaultStyle": "default",
      "gotQuestion": true // Displays answers after the markdeep content
    },
    {
      "contentId": 2,
      "defaultStyle": "default",
      "isAnswer": true, // Hide this side if only the question is active.
      "isAnswerFocus": true // Move to this side if the user submitted his answer.
    }
  ]
```

Listing 4.6: Card side properties of the quiz card type

4.3.2. Editor

The edit for mode multiple-choice content uses the same template as the card content editor. When a card type has been marked as multiple-choice, a new line is displayed for the user in his Markdeep editor, which is only responsible for editing multiple-choice questions.

By selecting a dropdown menu, the user can jump between the question and the individual answers. The number of answers can be adjusted by a plus and minus button. When an answer is selected, two more buttons appear in the navigation: the first one shows a checkbox to mark the currently active answer as correct and the second one with the exclamation point is there to edit the answer’s explanation. The last button with the dice is for mixing the position of the answers in the learning mode. All changes made by the user are displayed in real-time in the preview mode on the right side of the editor.

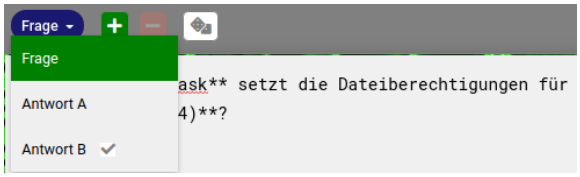


Figure 4.13.: .cards multiple-choice editor navigation

While the number of possible answers is theoretically unlimited, the UI limits the user to a maximum choice of 26 different answers, from A to Z. To avoid unsolvable questions, the .cards client checks for certain criteria before storing them in the database. If a criterion is not met, the user is notified of this via a bert alert [Che20]. The criteria include:

- The question must have content.
- There must be at least two answers.
- All answers must have content.
- At least one answer has been marked as correct.

4.3.3. Presentation

The design had to be compact to be able to display as many answers as possible at once. As a reference, the questions on linux.cards were taken, which were mostly about terminal commands. Here there are usually always four choices, from A to D, each with a one-liner as an answer. To prevent the hints from taking up too much space, they are hidden with a Bootstrap collapse [Inc13] element, which only unfolds when the user clicks on the corresponding button. The label of the answer, as well as the buttons and checkboxes, are located in the same line, above the answer. This alignment allows creating enough space for the content of the answer on mobile devices.

Figure 4.14.: Answer options on an iPhone 6

Uniform display of mixed content initially posed a problem. Because of the structure of the templates, it was not possible to guarantee the same order on a card. Thus, the order of the front side, with the backside could not match. Each side of a card is a template, which independently renders the content. A solution for this problem was provided by the npm package "knuth-shuffle-seeded" [Tim15]. During the call of the presentation mode, a random number is generated with a `Math.random()` function. Now, when the Meteor client shuffles the answers for the presentation, a seed and the array are passed to the shuffle functions. The seed is a combination of the ID of the card and the global random number. The template can then use this to uniformly shuffle the content for each side of

the cards.

```
static setNewRandomizedNumber () {
  randomizedNumber = Math.random(); // Set randomized number for active session.
}

static randomizeAnswers (cardId, answers) {
  return shuffle(answers, cardId + randomizedNumber); //Use a third party library to keep
      the same order on all sides of a card.
}
```

Listing 4.7: Functions used to shuffle answers

4.3.4. API & cheating prevention

All the content in .cards is controlled by Meteor subscriptions. Which subscriptions are needed for which page is controlled by the client-side router "FlowRouter" [Ltd17]. By default, the user receives all cards that are connected to the active card set. The problem here is that the client thereby receives the information of the correct answers and, with technical know-how, can read these on his device. This can be abused to cheat during a learning phase. To minimize this problem, the server uses a MongoDB command to remove the object for the answers before returning the cards via a subscription. However, to allow the user to still view some important elements, such as the question and answer options in their client, a meteor method was created that returns the required answers for the active card elements.

The client notes which card is currently in focus via a Meteor session named "activeCard". Based on this knowledge, the client sends a request to the server to the Meteor method "getAnswerContent" as soon as this value changes. The request contains a boolean of whether the answers should be disabled, the ID of the card set as well as an array of the maximum of three IDs, the active card, as well as the next and previous one so that the answers are displayed correctly during the transitions.

```
getCardAnswerContent: function (cardIds, cardsetId, disableAnswers) {
    check(cardIds, [String]);
    check(cardsetId, String);
    check(disableAnswers, Boolean);

    return AnswerUtilities.getAnswerContent(cardIds, cardsetId, disableAnswers);
  },
```

Listing 4.8: Meteor method for sending the answers to the client

In the presentation mode, the client always gets back all the information for the answers, but in the learning mode, it depends on the last interaction. By default, only the content that the client needs to display the content of the question and answers is returned. Which answers are correct and explanations are not returned. As soon as the user has sent his answer to the server, the server gets back an updated content as confirmation, which contains all information for the display of the answer page.

4.3.5. Limitations

At the moment, the client sends a hint to the server when requesting the data, whether the returned response should include all the information or hide the correct answers. Since this is controlled by the client, it can be manipulated so that the server always returns the correct answers, regardless of the currently active view. FlowRouter [Ltd17] is client-side only and cannot be called from the server. Here one would have to look for alternatives so that the server can find out on its own in which view the user is currently working from Presentation view or a learning mode. Even if this problem is solved, there is still the possibility to read the answers via a second account, or through a new tab in the card view. One way to prevent this for the situation in the same account would be to lock the card view while the user is in an active bonus.

4.3.6. Open Issues

The full functionality of the multiple-choice questions has not been implemented yet. There are still two open issues in the repository that need to be worked on:

- **Queued answer mode:** Currently, all answer options are presented directly to the user. In this variant, the user also sends only a single response to the server to register the answers of the active card. The system is now to be extended with another way of displaying the answer options. Which mode to use can be defined in the editor of the card. In this new mode, the user is presented with the answer choices in order and must decide whether the currently displayed answer is correct or incorrect based on two check options. At each check, the selection is sent to the server. If the answer is correct, then the next answer option is presented. If the answer is wrong, the server marks the card as not known and terminates the removal process.
- **Markdeep Export Support:** To export the card content to a Markdeep format, the .json format of the card must be translated to a Markdeep syntax. This is done in .cards by calling a function specialized for this. Since multiple-choice uses a different format compared to the normal card content, the export function has to be modified in order to support it.

4.4. Pomodoro timer and time tracking of answers

The Pomodoro timer shows the user the elapsed time with upcoming breaks in the learning mode as well as in the presentation view. It was integrated into the .cards web app in a software engineering project during the summer semester of 2018. As a reference, an implementation from a CodePen project was taken [Joh18] and adapted to the functionality and design of the .cards Platform.

4.4.1. Server side tracking

The implementation from the software engineering project only ran completely client-side. This means that when the client closes or refreshes its browser, the state of the Pomodoro timer resets. Since sometimes the client refreshes because of a server update, this behavior is not ideal, especially when the person is in a learning mode. For this reason, a concept was engineered that allows the server to track the progress of the Pomodoro timer and tell the client what position the timer is in when it refreshes.

This implementation also forms the basis for the upcoming "Leitner strict mode", to enforce compliance with the Pomodori times. The client communicates with the Meteor API via thre methods: "updateLeitnerTimer", "startLeitnerBreak" and " endLeitnerBreak":

- **updateLeitnerTimer:** The client sends a ping to the server every 60 seconds to credit one minute to its timer. The server then checks if the credit of a minute is possible with a tolerance of a few seconds. If this is possible, the server credits the client one minute on its current timer. For this purpose, a distinction is made between times that occur during learning and times that occur during a break. There are four different states a timer can have:
  1. Is in the learning mode.
  2. Waiting for a response to start the break.
  3. Is in a break.
  4. Waiting for a response to end the break.
- **startLeitnerBreak:** At the start of a break the user gets a SweetAlert2 pop-up [TL21], which must be confirmed to enter the break. This will fire an event to the server which will change the state of the timer if the time already elapsed allows it.
- **endLeitnerBreak:** Like "startLeitnerBreak" only here an event is fired to end the break.

```
"timer": { // Stored in leitnerTasks.
  "workload": { // Stores info about the passed workload time.
    "current": INTEGER, // Time passed in current workload.
    "completed": INTEGER // Number of workloads completed.
  },
  "break": { // Stores info about the passed break time.
    "current": INTEGER, // Time passed in current break.
    "completed": INTEGER // Number of breaks completed.
  },
  "status": INTEGER, // Is a workload, break or transition modal active?
  "lastCallback": DATE // The last time the client sent a confirmation.
}
```

Listing 4.9: Leitner 1.0 timestamps object

The use of this server-side verification requires the possession of a user account. A guest account or landing page visitors do not have entries in the database to perform this verification. Therefore, this feature is currently limited to the learning mode only. The demo view or presentation mode does not restore the already elapsed time of the old session.

4.4.1.1. Reworking the local time tracking of answers

As of this writing, the response times for individual cards are noted on the client-side and sent to the server. Only the difference between the two dates is recorded: The moment when the client sees the card and the moment when the response is sent to the server. This causes many problems in the statistics: Situations can arise where a client credits a single card for several hours because he left it open in the background via a tab. Additionally, the system can be exploited to send false information to the server, but this is not quite as problematic because the server uses the PomodoroTimer to note the elapsed time down to the minute. A client-side solution is nevertheless necessary because some cards are learned below one minute, which the server cannot register.

A solution to minimize this problem would be an event-based server communication and a local copy. For this, a new MongoDB collection named "leitnerTimeTracking" is introduced, which stores time information about not yet answered cards. The client does at the same time store a copy of this collection in an array, and update it from its side. In this new format, a distinction is made between the question and the answer, as well as whether the time is active learning time or entering/exiting a break.

```
"timelineStats": {
      "question" {
         "active": INTEGER, // Milliseconds
         "break":  {
            "entering": INTEGER // Milliseconds
            "active": INTEGER // Milliseconds
            "exiting": INTEGER // Milliseconds
         }
      }
     "answer": {
         "active": INTEGER, // Milliseconds
         "break":  {
            "entering": INTEGER // Milliseconds
            "active": INTEGER // Milliseconds
            "exiting": INTEGER // Milliseconds
         }
      },
      "submitted": DATE
}
```

Listing 4.10: New time tracking format for Mongo DB

An update of the local list is then always transmitted to the server by the following circumstances:

- A tab or window closes.
- The user skips the card.
- The user replies to the card.
- The user displays the next card (multiple-choice cards only).
- The user exits the learning mode.

If .cards updates because of an update, the client can store the current state in local storage, and restore it when reloading.

When a card is answered, the entry is removed from the array and the "leitnerTimeTracking" collection and put into the " leitnerAnswerHistory" collection.

4.4.2. Games & backgrounds during breaks

The first integration of the Pomodoro timer only had the background switched to black during a break and a large version of the Pomodori clock displayed. To entertain the user during these short breaks within the app, animated backgrounds and games are being offered. During the development phase, preparations were already made for this in the repository, but this feature could not be implemented during the project. Therefore only detailed documentation for the implementation was filed as an issue. The implementation then took place in Web Programming Weeks 1 + 2, which took place in February 2021.

The content of the games and backgrounds was taken from Codepen projects [Cod21], if the license allowed it, and deposited in the public folder of the repository. All content inside the public folder is not compiled and can be called via a URL when the server is started. By calling this, the games and backgrounds on .cards can be used. This has the advantage that the styling configuration of .cards does not change the content of these elements.

For the client to know where these are located, a list is stored in a configuration file, with the metainformation for each available game and background In it, the ID is defined, as well as whether this element should be offered for Safari or mobile. This ensures that if the application does not work due to a missing Safari browser [Ale21], it will not be offered for the browser. Also, you can define a maximum fixed width or height for each game, so that they are not destroyed by the layout if stretching to the entire window size. In a later stage of development, it would also be possible to overlay a small game over an animated background. Finally, the position of the Pomodoro timer for backgrounds can be specified to reduce the overlay of some graphical elements.

```
{
    "id": 1,// The id located in the public folder: backgroundID
    "name": {// The name of the background
      "de": "Matrix",
      "en": "Matrix"
    },
    "clockPosition": "top_right",//position of the clock if not top_right, can be following
        (top|bottom)_(right|left)
    "preview": "preview.png",//The path to preview image after /gameBackgrounds/background$
        {id}/, if not /gameBackgrounds/background${id}/preview.png
    "features": {
      "safari": true,// Enable for safari
      "mobile": true// Enable on mobile
    }
}
```

Listing 4.11: A config entry of a lockscreen background

At the beginning of a break, the app automatically switches to a background. The default background can be defined in a config file under `imports/config/lockScreen.js`. Additionally, .cards fades in a footer navigation that allows the user to switch between backgrounds and games. The navigation itself was implemented using the jQuery plugin Owl Carousel 2 [Dav21]. For the navigation to display a preview for the game or background, an image called `preview.png` must be stored in its folder. The dimensions of this preview image should correspond to the resolution `512×288 px`.

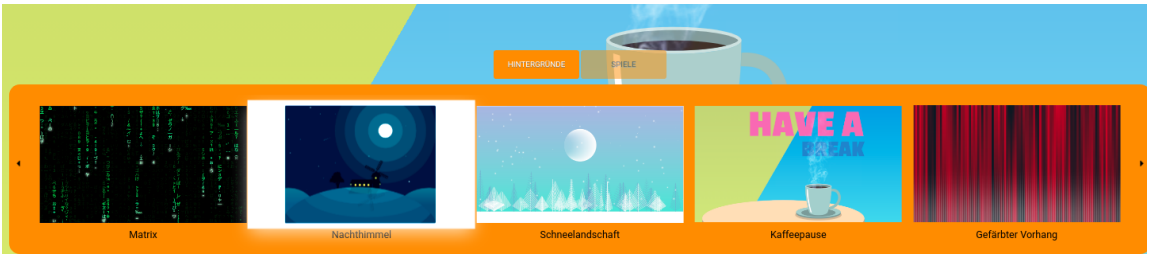


Figure 4.15.: Background selection during an active lockscreen

A major disadvantage with the implementation via iframes is that Meteor cannot communicate with the displayed games & backgrounds. For example, it is not possible to extend this feature so that it can save and restore high scores in the user profile for games. This means that concepts like high score tables or the integration of flashcards in the games are not possible at its current implementation.

4.5. User-Feedback System

One of the most important aspects of a flashcard system is the quality and correctness of the content. Despite careful checks, it can happen from time to time that some content is not correct: Be it grammatical, syntactical, or logical errors. To find and report these errors quickly, we have set up a space for feedback in another application we have developed, where students who are supposed to use the learning content can leave error reports or other comments. One disadvantage that comes from the disconnect between two systems is navigating to the error source itself. The .cards web app does not have direct access to the frag.jetzt database [Tea21c]. Additionally, the feedback entries consist of only one string, which is difficult to reuse in a system for automatic navigation.

Therefore, it was decided that .cards needed an internal solution for error reporting. The minimum requirements were logged in a jour fixe and filed as an issue for the web programming weeks 1 + 2 of March 2021. Users will now find a new button on the left side of the navigation in the card view to report an error. Clicking this opens a modal that offers the sides of the active card as navigation elements. Hereby the system can save on which side and which card the error occurred. The following errors can be reported to the system:

- Spelling error.
- Image is missing.
- Layout is broken.
- Link is broken.
- Other, if none of the errors listed above apply. In this case, the user can describe the error in an input field.

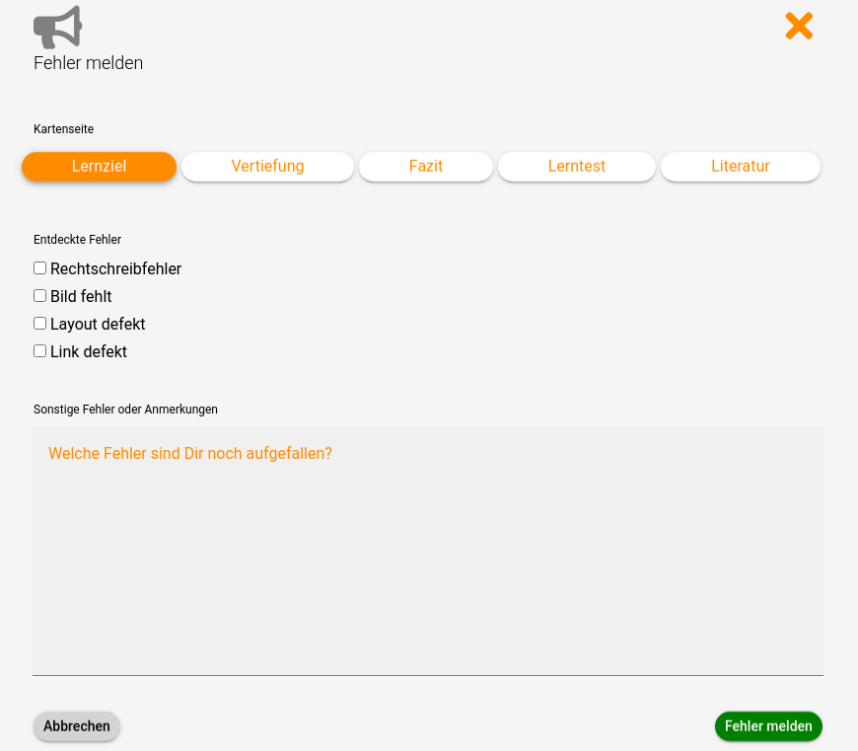


Figure 4.16.: Error reporting modal

After the error is reported, it is stored in a custom MongoDB collection named “errorReporting”. The entry contains all the information needed to construct a direct link to the error, as well as a status if the error has been resolved. Since the client uses a subscription on this collection, new errors are automatically displayed on the app without requiring a refresh. By introducing a new label, it is possible to see if a card set or card still has unresolved errors. The number of unresolved errors is displayed as a number in brackets. For performance reasons, this number is generated by the server after each new report and stored as a field in the affected card and card set.

The error label is interactive and performs different actions depending on the view. When the label is displayed on a card set, it automatically redirects the user to the table of contents. In this view, only the cards that still have an unresolved error are being displayed, as well as the number of unresolved errors per card.

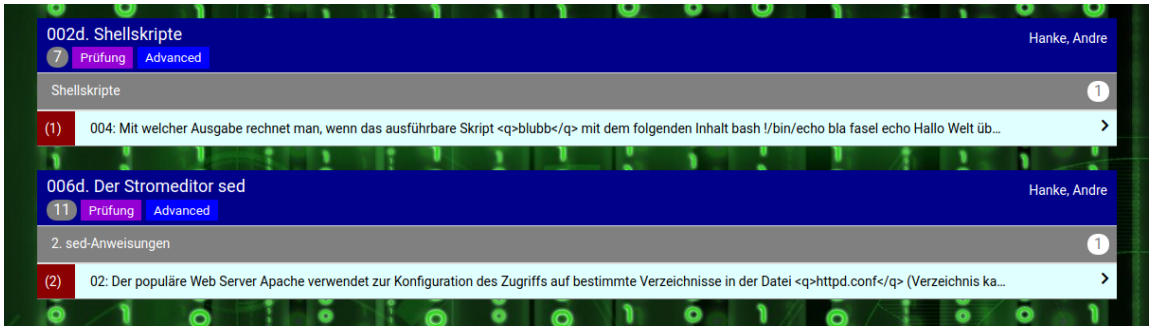


Figure 4.17.: Table of content with active error reporting filter

If the label is displayed on a card, then an overview of all unresolved errors reported for this card is shown. If the user has editing rights for this card, then he can mark it as resolved. Resolved errors remain in the database, but are no longer displayed in the UI.

Figure 4.18.: List of reportet card content errors

4.5.1. Future enhancements

This new feature is already being used by some students. Though, some seem to report errors in the code of application instead of doing so in the associated GitLab repositories. In this case, it might be necessary to adjust the descriptions so that the system should only be used for card content errors. This feature is also still in the first stage of development and will be supplemented with additional functionality in the upcoming months. This includes:

- **Direct link to the card side of the bug:** While the side is currently stored in the database, it is not yet linked to the card. To implement this, it needs refactoring for the display of cards because they currently reset the active side to the first entry when the page is called.

- **E-mail notification on new error:** When a new error is registered, all users with editing rights for this card should receive an e-mail. The e-mail will contain a direct link to the reported card. Users include the owner of the card, as well as proofreaders and admins, as well as the report himself.
- **Overview in profile navigation:** There is currently no view that shows the user the errors they have reported. Here a page would be provided, which contains a table of contents of all related errors. In this, the user can filter for errors that are still unresolved or already resolved.
- **Overview in backend:** Accessible only to the admin and proofreader. Compared to the profile navigation, this view offers advanced filtering options such as filtering by the reporter. For proofreaders to access this view, the backend needs a redesign that allows limited access to it. As of now, only admins have access to the backend.
- **Protcollation of editor:** When an error report is resolved, it is currently not obvious who resolved it. Once the view is customized to show already resolved errors, the view must also list the name of the editor.

4.6. Markdeep

Markdeep is used for displaying various site elements: This includes the info pages, card content, and also card set content. A minified version is placed in the folder `client/thirdParty/markdeep/` and integrated via the client. After the content has been converted to HTML using the Markdeep plugin, it is further customized in some additional steps to support .cards specific features:

- **malicious code analysis:** Since we allow users to create HTML content and share it across the platform, this needs to be sanitized to prevent XSS attacks. Helping out with this is the javascript package "DOMpurify" [Mar21]. Except for allowing iFrames to embed videos or other ARSnova applications, or disallowing styles, the default settings of DOMpurify are used here.
- **adjusting links:** All HTTP links will be adjusted to HTTPS. Additionally, each link opens in a new window so that the user does not accidentally leave the .cards web app.
- **Add lightbox support:** All image elements get a Lightbox2 [Lok21] wrapper and are grouped by side content. This allows the user to view the images in a larger view and easily switch between image groups.
- **Force Media Controls:** Navigation is enabled on all video and audio elements.
- **Customize iframes:** If there is an iFrame, it will be equipped with a special bootstrap class to make it responsive.
- **Adding predefined MathJax commands:** Because we are using a custom installation of MathJax [Tea21d], we need to manually add the MathJax commands provided by Markdeep to make them available to the user.

4.6.1. Reworked Markdeep export

In the lecture for software engineering from the winter semester 2020 / 2021, the students were supposed to make some submissions via a Markdeep export. To make the submissions more readable for the tutors, some adjustments were made to the export. First, the default `starter.md` template, was replaced with the `slate.md` template, which offers more space for the content and a navigation on the left side [Mor21]. This already offers an advantage for PlantUML exports alone, since the diagrams are now displayed large due to the increased width. The navigation caused difficulties, however, because of the way a card is constructed:

Navigation is constructed using HTML headings. `<h1>` for the title of the card and `<h2>` for the side of a card. However, if a heading is used in the content itself, then the output no longer matches the actual intended scheme. To minimize this, the Markdeep content must be adjusted before exporting: The simplest solution is to put two `##` in front of each Markdeep title. Unfortunately, this has the disadvantage that if a title `<h6>` exists, it can

no longer be converted. Additionally, each title must also be unique so that the HTML anchor tags point to the correct content. This was solved by assigning an ascending number to each subsequent title if a title exists more than once.

Finally, an info table was added to get an overall view of the submission. This also serves to ensure that the students have made the delivery with a current version in their local .cards environment.

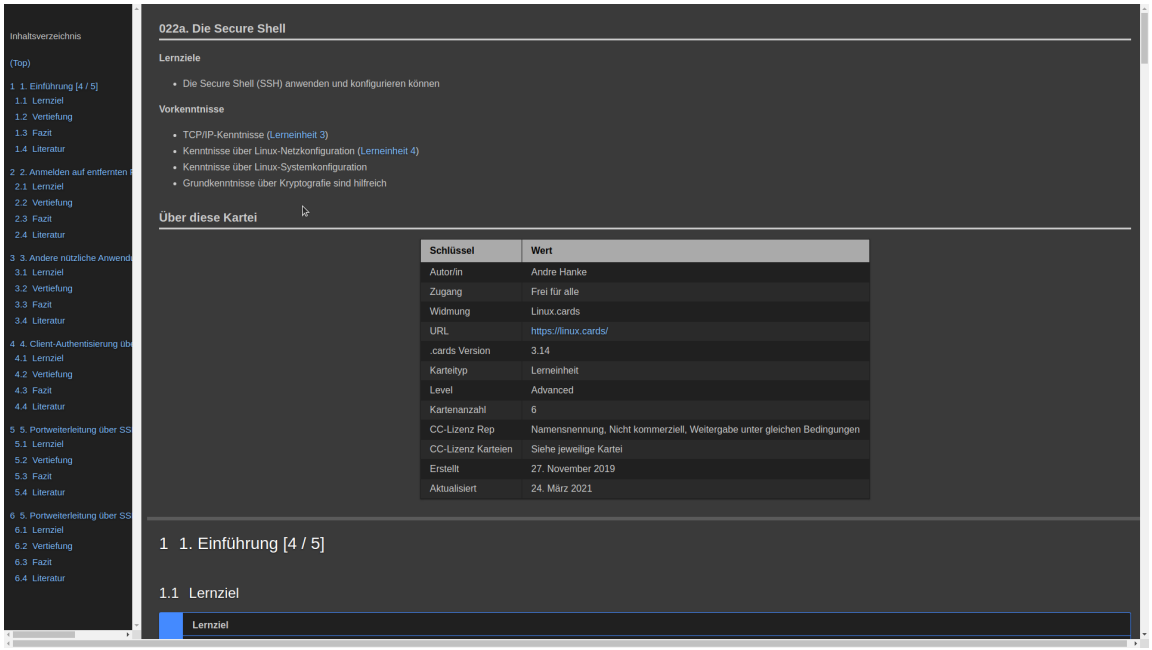


Figure 4.19.: New Markdeep export format

4.7. Fullscreen mode

Originally the .cards web app was used for presentation purposes as well as for repeated learning. Some students had problems presenting their results to the lecturer: Either their browser window wasn't maximized to leave enough space for displaying the content or their toolbars took up too much space. As a result, an automatic fullscreen mode was introduced to simplify this process. With this, the students only needed to start the presentation mode and the window already started in fullscreen. After some time, however, it turned out that forcing the fullscreen mode also has its disadvantages: For example, switching between multiple screens is bothersome. To not scrap this feature completely, the logged-in user has now the option to set the default behavior. A new profile entry has been added to allow setting said behavior for different pages of the .arsnova web app. The following options can be selected:

- **Automatic:** Corresponds to the old implementation. When the user is redirected to this route, the browser automatically switches into fullscreen mode. If the fullscreen mode is exited without leaving the route, for example by opening a tab or pressing escape, a SweetAlert2 warning will message appear, allowing the user to re-enable fullscreen mode. This warning message also appears if the user exits the fullscreen mode by reloading the app, as for security reasons this can only be activated by user input and not automatically [Moz21a].
- **Manual:** To activate fullscreen mode, the user must press a button for it in the right sidebar card navigation. This button is available only in manual mode. No SweetAlert2 warning message will appear when exiting this mode.
- **Manual for active session:** It will ask the user if they want to use automatic or manual mode the first time they enter the route. This selection will be remembered until the user ends his session, or changes their settings in the profile.

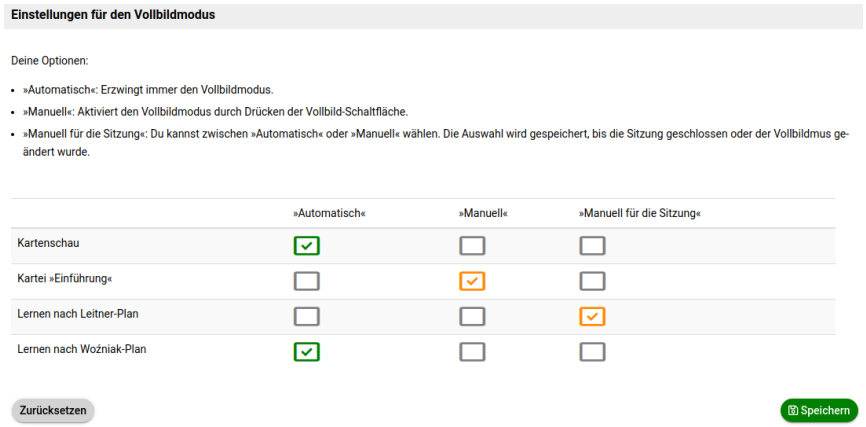


Figure 4.20.: Client fullscreen settings

The server installation can select which of these buttons each user role has access to. If there is no access for a single button, then the settings route is hidden and the default settings are used for the other routes.

```
"fullscreen": {
    "settings": {
      "enabled": [FREE, EDU, LECTURER, PRO],
      "presentation": [FREE, EDU, LECTURER, PRO],
      "demo": [FREE, EDU, LECTURER, PRO],
      "leitner": [FREE, EDU, LECTURER, PRO],
      "wozniak": [FREE, EDU, LECTURER, PRO]
    },
    "defaults": { // Will be used if the user got not access to the fullscreen settings.
        Admin and editors always have full access to the fullscreen settings.
      "admin": {
        "presentation": MANUAL_FULLSCREEN,
        "demo": MANUAL_FULLSCREEN,
        "leitner": MANUAL_FULLSCREEN,
        "wozniak": MANUAL_FULLSCREEN
      },
      ...
```

Listing 4.12: Server fullscreen settings

4.8. Webmanifest

Since we provide two installations with different brandings, it has been necessary to provide different web app manifest files so that the PWA (Progressive Web App) uses the correct one. The header icon also had to reflect both installations. The core problem here is that the header file could not be changed after the build process. One solution would have been to use different repositories for both installations. However, this would have caused a greater maintenance effort. Another option would be to use two different build processes that adjust these files.

The solution to the header problem is a package provided by Meteor [Sof21c] that allows manipulation of it. Before the client receives the HTML DOM from the server, it can manipulate the header through a callback. Here the header information is removed from the `client.html` file and swapped by the content a config file, which uses the correct header information depending on the installation type.

```
1 onPageLoad((sink) => {
2   let headData = config.defaultHeadData;
3   if (Meteor.settings.public.dynamicSettings === "linux") {
4     headData = config.linuxHeadData;
5   }
6   headData.forEach(function (item) {
7     sink.appendToHead(item);
8   });
9 });
```

Listing 4.13: Server onPageLoad function to modify the client header

The web app manifest file is located inside the public folder after the build process and can be manipulated afterward. Here it was sufficient to modify the deployment script of both servers so that the required web app manifest file is moved to the correct location.

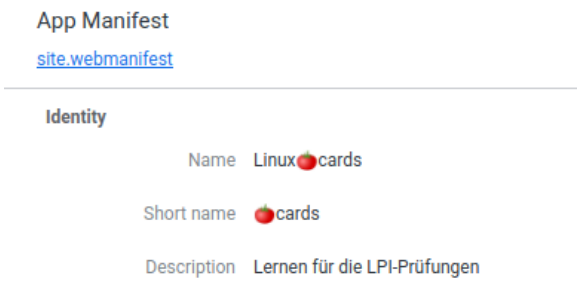


Figure 4.21.: Linux.cards webmanifest identity in Chrome DevTools

4.9. Refactoring

To ease the entry into the development of the .cards web app for newcomers, as well as to enable the implementation of some of the features listed in previous sections, it has been necessary to perform refactoring of some project files. A large part of the problem was that many frequently used functions were located within a single file, making them difficult to find. The goal here has been to restructure these large files into smaller files and subfolders.

4.9.1. API

In the beginning, all Meteor methods, subscriptions, and utility functions were stored in the folder `imports/api`. After the refactoring, the utility functions are now in a new folder `imports/util`. Methods and subscriptions have been split between `imports/api/meteorMethods` and `imports/api/subscriptions`.

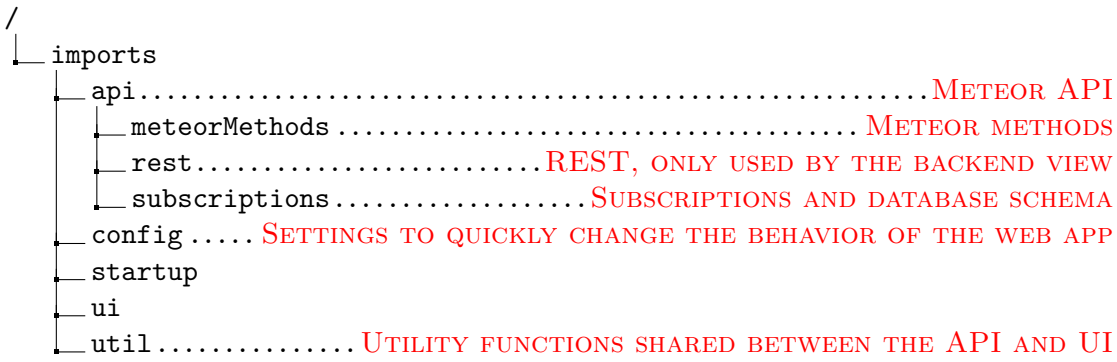


Figure 4.22.: The new API folder structure of .cards

4.9.2. i18n

Similar to API, all translations were stored in one file. This file was reduced over time, and new subgroups were stored into subfolders. Unfortunately, the refactoring for this is still in progress. Most translations are still inside a single file.

4.9.3. Routes und Registerhelper

Contains route entries, which are necessary for Iron.Router [Chr17]. The files can be found under `imports/startup/client/router`. Since these are independent functions, a subdivision into subfolders and files was rather unproblematic. Iron.Router got replaced during the refactoring with FlowRouter, which uses an almost identical syntax.

As it is the case with routes, the registerhelper provided by meteor was also stored inside a single file, located at `imports/startup/client/registerhelper`. These have been as well split up into individual files and folders for better clarity.

4.9.3.1. FlowRouter

It has been planned to update the Bootstrap 3 [Inc13] version in .cards with Bootstrap 4. A direct switch to 4 has not been possible due to jQuery v1 dependencies from other meteor packages. To use Bootstrap 4, it is required to run jQuery v3. Among the packages using the older version was Iron.Router and i18n. Therefore Iron.Router has been replaced with the FlowRouter package [Ltd17]. With this change, the jQuery v1 dependency is now reduced and it is now possible to use dynamic imports for routing. This can reduce the amount of data that is sent to the client.

4.9.4. Server initialization

In the repository, there is a file that is executed by the server during startup, located at `imports/startup/server/boot/initialize.js`. This file is used to perform migration steps, such as updating the demo and accounts for the notification test service.

During 2016, many changes to the database schema took place, which bloated up this file. To ensure that future adjustments can still be made relatively easily, the file was split into small meaningful groups and a helper function was written that displays the current migration step in the console when the server is booting up.

This step was especially necessary when migrating the inventory data from the linux.cards installation to the new Leitner 2.0 database schema. The transfer of the old data can take up to 15 minutes and therefore requires an output on the server console about the current status.

```
I20210426-16:41:00.292(2)? ##### Starting Step "TranscriptBonus Migration"
I20210426-16:41:00.292(2)? 01: Migrating TranscriptBonus deadlineEditing field...
I20210426-16:41:00.293(2)? 01: Nothing to migrate, skipping.
I20210426-16:41:00.293(2)? 02: Migrating TranscriptBonus stars field...
I20210426-16:41:00.293(2)? 02: Nothing to migrate, skipping.
I20210426-16:41:00.293(2)? 03: Migrating TranscriptBonus rating field...
I20210426-16:41:00.294(2)? 03: Nothing to migrate, skipping.
I20210426-16:41:00.294(2)? ##### Step "TranscriptBonus Migration" took 3 Milliseconds to complete
I20210426-16:41:00.294(2)?
I20210426-16:41:00.297(2) (percolate_synced-cron.js:87) SyncedCron: Scheduled "leitnerCron" next run @Tue Apr 27 2021 08:00:00 GMT+0200 (Central Eur
=> Started your app.

=> App running at: http://localhost:3000/
```

Figure 4.23.: Database migration steps on server console

5. UI/UX-Evaluation

Students were tasked in the module software engineering from WS 20/21, to evaluate the .cards web app [Jon+21] using the method "Thinking aloud" [Uwe05]. The evaluation was carried out in the following steps:

1. Students create a BBB (BigBlueButton) [Inc21a] room on the ARSnova server installation and invite the test subject to a web meeting.
2. Using two browsers displayed side by side on the screen, the business use cases of the app will be demonstrated, then the subject will be explained the bonus award and acquisition using an activity diagram.
3. The test subject is then tasked to run through the learning bonus process using by following the activity diagram. Here, different roles are used in both browsers: On one side the role of the student, and on the other the role of the lecturer.
4. After both scenarios have been played out, the test subject is interviewed according to the interview questions from chapter 3.4 [Uwe05] of the "Thinking Aloud" article.
5. Finally, the subject has to fill out a "User Experience Questionnaire (UEQ)" [Tea21e] questionnaire, which the students evaluate at the end.

The evaluation was carried out by a total of 12 students. Here 12 test subjects were interviewed, which are in the age of 18 - 29 years. Most of the subjects are currently pursuing graduate studies, including computer science, teaching, childhood education, and business administration. One person is employed and works in a company that specializes in app retargeting [Bet18].

The following is a summary of the positive and negative aspects that the subjects noticed while role-playing as a student and lecturer in a learning bonus phase:

Positive aspects:

- A wide range of predefined card types covering all possible use cases.
- The bar chart inside the learning status window is visually appealing and informative.
- The Leitner algorithm is automated and the user does not have to worry about which cards need to be learned soon.
- There is already a lot of useful learning content on the platform.
- Color-coding for some of the navigation buttons is considered to be positive. For example, using green on a save button.

- The content of the info dropdowns is structured and informative.
- Icons within the card navigation are mostly self-explanatory. The consistent use of icons throughout the web app is also good and understandable.
- Keyboard shortcuts for answering a question and navigating the card are well chosen.
- The presentation within the learning view is clear and informative.
- "Bonus, transcript, and card types" options of the web app are praised on a functional level.
- Settings can always be found where users expect them to be.
- Having the option to sort individual columns within the learning statistics is helpful.

Negative aspects:

- The main navigation is felt to be too small. It was suggested here that it should be on the left side of the screen in desktop view, especially since there is a lot of unused space.
- It is not obvious that the magnifying glass, signpost, filter, and word cloud are buttons that have nothing to do with the rest of the navigation. A better separation would be desirable here.
- The color design is perceived as "old-fashioned". There are too many colors that do not harmonize with each other. Testers would like to have options to use the theme in a minimalist presentation: A low well-coordinated color palette without background images. A dark mode would also be desirable.
- Most background images are distracting. Noticeable here is the green matrix background, which is used in the settings page for the transcript bonus.
- The placement of the card set details navigation is too chaotic. All controls should be in the same place.
- The info dropdown and label design of the card set details are confusing because they look like buttons.
- Certain color choice makes the text hard to read in some places. Especially the background of the card set description (tomato) was criticized.
- The purpose of the automatic full-screen mode is understood, but it is also found irritating.
- Too much information is collected about the learning behavior of a student, which is perceived as a deterrent for some test subjects. Many do also have a privacy concern about this.
- The intended use of the Pomodoro timer is not immediately obvious. This only becomes clear as soon as the user starts switching to the learning mode.

- Settings for the learning bonus and the bonus transcript need more explanations. A detailed tooltip would make sense in some cases.
- The 3D cube view is too small in some scenarios, especially when not using the whole screen. A reduction of the distance between the 3D cube and navigation elements could improve here.
- Sliders to set the time for the Pomodoro timer should allow the user to select individual minutes instead of 5-minute steps.
- The reset button for the Leitner simulator should not be placed below the modal but above the setting options.
- Hiding a button if the user can't interact with it is confusing. The testers would prefer to render the button in a grayed-out form, with a reason as a tooltip why the button can't be used.

The major points of criticism were related to the color scheme and background images. These were criticized by most testers. To address these issues, work is currently underway on a dark theme that uses a minimalist color palette. An additional goal is to fix the WCAG (Web Content Accessibility Guideline) contrast conflicts [Web18]. Other major criticisms, such as the lack of descriptions for setting a bonus phase, as well as missing settings for the automatic fullscreen mode, have already been integrated into the production.

5.1. Summary of the UEQ evaluation

Subsequently, the results of the 12 individual UEQ evaluations were summarized. It should be noted that the sample size here is very small, and the results may not be representative enough. Small samples usually have several participants up to at least 40.

5.1.1. Mean value per Item

The mean value per item chart 5.1 shows the evaluation of the individual fields that the user can select in the questionnaire. Here, each field offers seven answer options that can be checked. The options -1 to -3 are considered to be bad, 4 is neutral and 4 - 7 are rated as good.

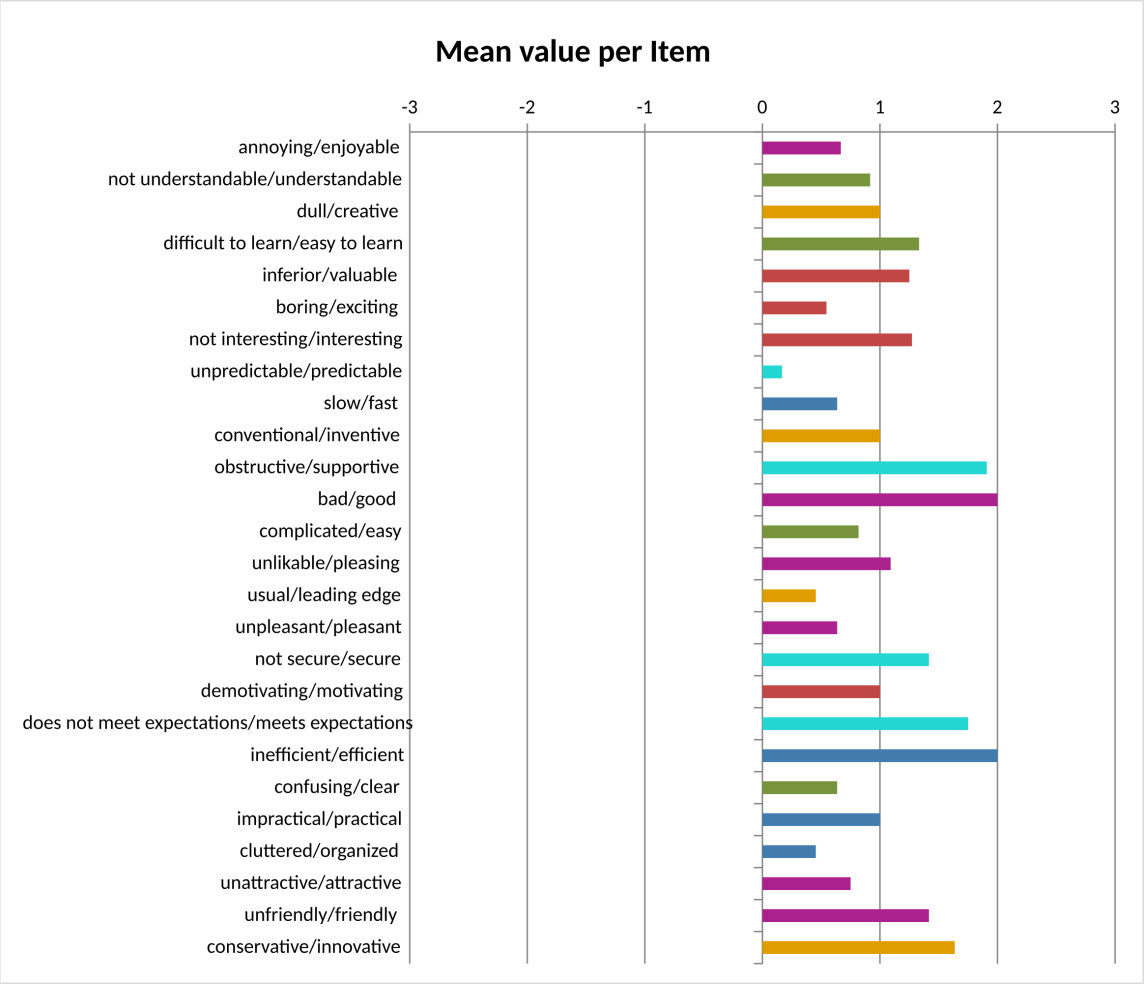


Figure 5.1.: UEQ: Mean value per item

When looking at the chart, it is noticeable that the result is dominantly in the positive range of a mean higher than (> 0.8). Only 9 of the 26 different items are between a mean value of 0.2 - 0.8. There is no item, which is in the negative range (<= 0.8). The results for the items “obstructive / supportive”, “bad / good”, “ does not meet expectations / meets expectations” and “inefficient / efficient” were particularly positive. Less positive results were obtained for the items “annoying / enjoyable”, “boring / exciting”, “unpredictable / predictable”, “slow / fast”, “usual / leading edge”, “unpleasant / pleasant”, “confusing / clear”, “cluttered / organized” and “attractive / unattractive”, this can be largely attributed to the background images and the colorful theme, which can overwhelm the user in an initial condition.

An extremely large variance between the individual results forms with the points “unpredictable / predictable” with a deviation of 3.8, “usal / leading edge” with 3.5, “secure / not secure” with 3.9 and “organized / cluttered” with 3.5. This can be explained that the previous experiences of the test subjects regarding learning apps does vary.

5.1.2. evaluation of scales

The 26 response options are divided into six different categories 5.2. For these categories, a mean value is calculated and shown in two different diagrams. The first one shows the mean value with the scaling with a range of seven units, the same amount that is found in the questionnaire. The black lines within the bars represent an “ error bar”. This indicates a 95

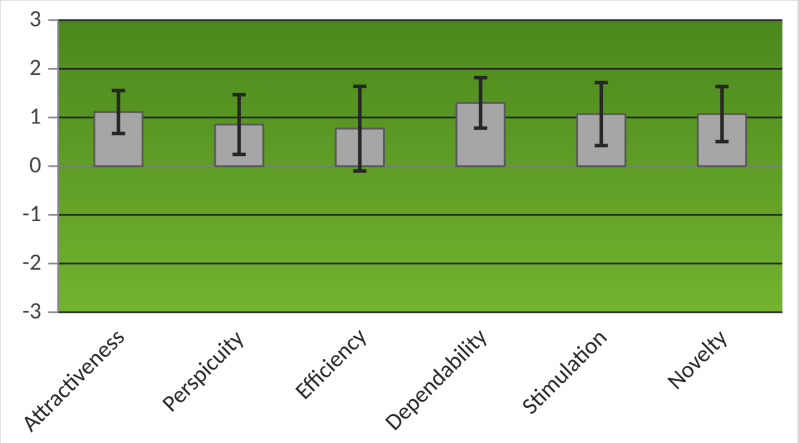


Figure 5.2.: UEQ: scale with error bars

The results of the categories lie between 0.7 and 1.1. The large error bar in the “Efficiency” category is striking. This can be attributed to the fact that some test persons noticed the loading times negatively, and also found the user interface to be overloaded due to the colors and background images.

Below is another chart 5.3, which shows the same diagram in a reduced result spectrum from 2 to -2, because the scale is usually never fully utilized [Tea21e].

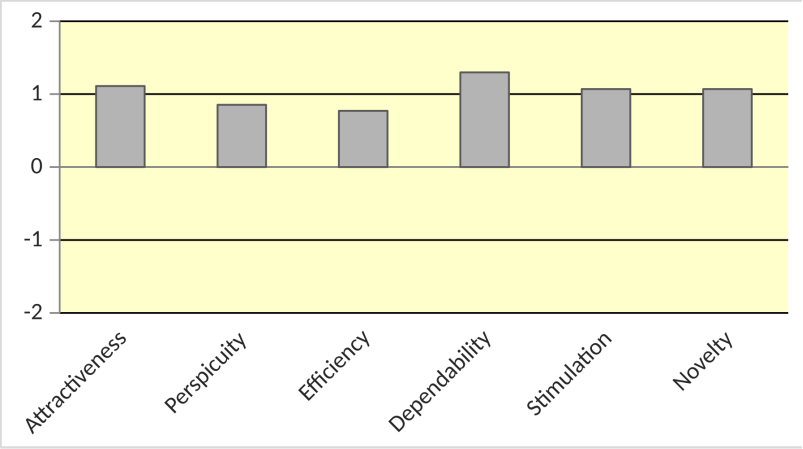


Figure 5.3.: UEQ: Scale without error bars and adjusted threshold

Finally, the chart 5.4 can be summarized into 3 main categories: “ Attractiveness”, “Pragmatic Quality” (Consists of “Transparency”, “Efficiency”, and “Controllability”), and “Hedonic Quality” (Consists of “Stimulation” and “Originality”. In the pragmatic quality, all task-relevant aspects are described, and in the hedonic quality the aspects that are not task-relevant.

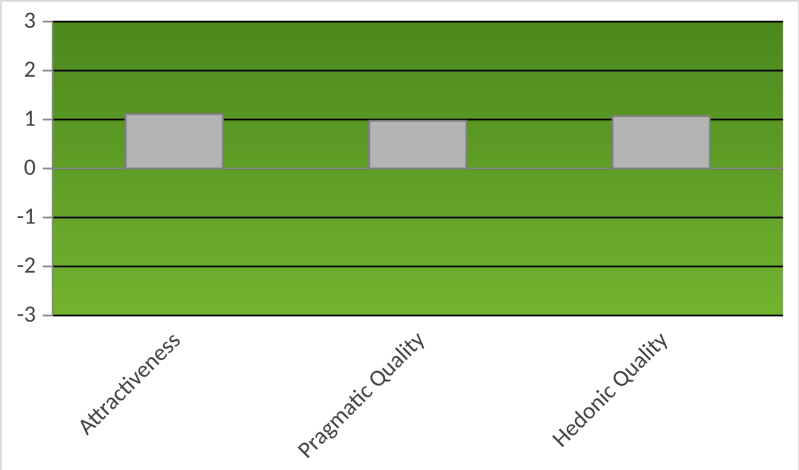


Figure 5.4.: UEQ: Pragmatic and hedonic quality

It is noticeable here that all 3 bars lie between a value of 0.9 to 1.1. The poor result for the pragmatic quality compared to the other bars, is due to the low result for the efficiency.

5.1.3. benchmark

The benchmark chart 5.5, provides a comparison with the benchmark data values provided by UEQ. A better comparison would have been possible using data values from other learning platforms, however, the tests from the software engineer module in the WS 20/21 event do not provide this.

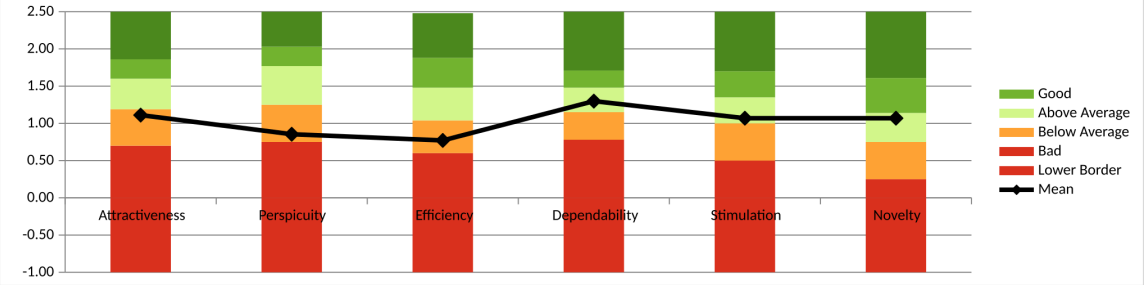


Figure 5.5.: UEQ: Benchmark

It can be seen that half of the categories (“Attractiveness”, “Transparency” and “Efficiency”) are below average and the rest (“Controllability”, “Stimulation” and “Originality”) are above average. Efficiency is the worst result with a value of 0.77 and “Controllability” has the best result with a score of 1.30. Overall, the .cards web app is in the average area: it is partly better than other applications, but also worse in some others. However, none of the aspects stands out in a very positive or negative way.

6. Upcoming enhancements and open issues

Unfortunately, not all goals could be achieved within the development phase and the creation of the master thesis. On the one hand, the complexity of some issues was underestimated, and on the other hand, additional feature requests were added later on. The following is a list of all issues and features that have a high priority for the maintainability and usability of this project.

6.1. Leitner simulator 2.0

The structure of the current implementation for the Leitner simulator is quite simple 6.1: The client creates an integer array with the number of available boxes (1 to learned). Each number corresponds to the number of cards currently in the box. The timeout intervals for each box are simulated by a 2D integer array: The number of the interval indicates the depth of the array. When a simulated day passes, then the numbers of the array get shifted. When a number reaches the highest position, it is going to get removed from the 2D integer array and put back into the box array. When a snapshot is generated, the sum of both arrays gets combined.

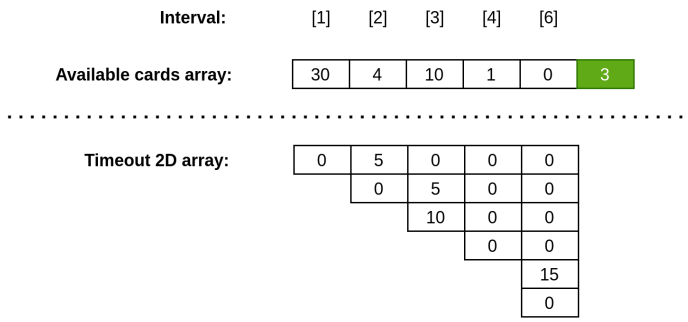


Figure 6.1.: Visual representation of the arrays used by the Leitner simulator 1.0

This variant has the advantage that it can be executed quickly by the client. A simulation over months takes less than a second. The disadvantage here, however, is that the simulator cannot determine what type of card type each card uses. Additionally, there is the problem that when changes are made to the Leitner algorithm, then the simulator must be separately updated to account for the changes. Such a problem occurred in a previous version when

the algorithm placed certain card types directly into the last box if the card got answered correctly. As a result, the simulator no longer matched the implementation.

To minimize this problem, instead of storing a counter of the number of cards, one could store several objects into the array, with each representing a card. This would easily solve the problem with the different behavior of card types, but still, the simulator has to be adapted for other changes. A second solution, which was also explored during the development phase, is a full implementation of the algorithm as it exists on the server-side. This would only require moving the code from the server folder to the imports folder, to make it accessible to the client. In Meteor there is a possibility to create a collection only on the client-side by initializing it with `null`. These client-side collections could then be used specifically for the simulator. This requires an adjustment to the algorithm so that it knows which collection to use. The problem here is that the simulation is slowed down by a significant amount. Since this simulation takes place on the client, the UI is blocked until the function is completed [Moz21b]. One could prevent this with a web worker, but these do not have direct access to the Minimongo DB [Sof21a] and Meteor methods that the client uses.

Another alternative would be to execute the simulation on the server. In this case, a temporary user would have to be created on the server, who is responsible for the contents of a simulation as well as a learning phase for the simulator settings. Once a simulation is triggered, the server only needs to reference these simulation objects. A progress bar could be realized through the introduction of a new subscription, in which the server writes in the current simulation status. The problem with this implementation would be an increased server load.

The selectable error rate currently only causes, that within the entire simulation process, a percentage calculated from the total number of cards, is put back for the selected box. This has been so far the simplest variant to put some cards in the previous box. Applying the percentage directly to a card could only result in the simulator never moving it to the last box (Learned). In reality, the error rate is not static but decreases after time through repeated learning. Therefore, the simulator could be reworked so that each card stores the error rates for each box as starting values. When the simulator then answers a card, the error rate of the affected card and box will be reduced until it reaches a value of 0. The percentage by which the error rate is reduced can then be defined for each box in a reworked UI. Another point is that there are cards with different difficulty levels. Low difficulty levels have a lower error rate compared to high difficulty levels. Here the simulator should also offer the possibility to set the error rate per difficulty level.

To record this information for each card, one could either create a new collection or store it in the already existing collection for card statistics. However, the latter could unnecessarily increase a database with already many existing entries.

6.2. Drag & drop support for table of contents

The table of contents is sorted based on the title and the first side of a card. Cards that have the same subject will be grouped. If the table of contents belongs to a repetitorium, then each card gets an additional parent group based on the card set that they belong to. This solution is quite burdensome for card creators if they have to manage a card set with hundreds of entries.
The problem has already been partially minimized by using the creation date as a second sort option, which the owner can define within the card set settings. To eliminate the problem, the card creator should have the ability to sort all content regardless of content or creation date. The jQuery plugin "draggable" [Fc21b] would be suitable for this approach.

This could be integrated with a special editor view of the table of contents of a card set. If the card creator interacts with a card inside the table, then he can drag it around and see the new layout as a preview. With a confirmation through a save button, these changes are then going to be stored in the database. The following events should be covered for moving:

- The position within a group (Subject of the card) can be moved.
- The cards can be moved between individual groups (Subject of the card).
- A repetitorium can change the positions of the referenced card set.
- The content of a card set, cannot be changed by a repetitorium.

To implement this feature, some additional adjustments would have to be made to the database schema. The card set object needs to store an array with all available titles. The position of the title inside the array determines the display order. In addition, each title also needs to save which card objects belong to it. This can be done by storing said id's inside an array.

```
{
  ...
  "titleGroup": {
    [
      "title": STRING // The display name for the group.
      "cards": [STRING] // IDs of cards that belong to this group.
    ]
}
```

Listing 6.1: Additional fields for a card set to allow manual positiong of cards

The editor then receives a new button, which the card creator can use to create as many groups as desired. This would also speed up the process of assigning a group to a new card.

6.3. Command line questions

An important aspect of the LPIC examinations is the use of the Linux terminal. By implementing the multiple-choice questions, we do already have a setup for asking general questions. However, a command line must be written down without errors, so that the command can be executed successfully. Therefore, it is planned to introduce a new card type called “Command line”, which checks if the user knows the exact syntax of the tested commands.

The card type has two sides: A frontside with the question, and a backside with the answer. The user is offered an input field on the front side within the card, where he can enter his command. The solution can be sent to the server only after an entry is made into the input field. When submitting the solution, the server checks if the answer is correct, as it’s the case with multiple-choice questions, and sends back the result for the display of the backside. In here, the user will see both his input and the correct answer, with highlights where he started doing a mistake, if the answer was incorrect. The server trims the input before comparing it to the solution. The Meteor client and server also have an input length limit.

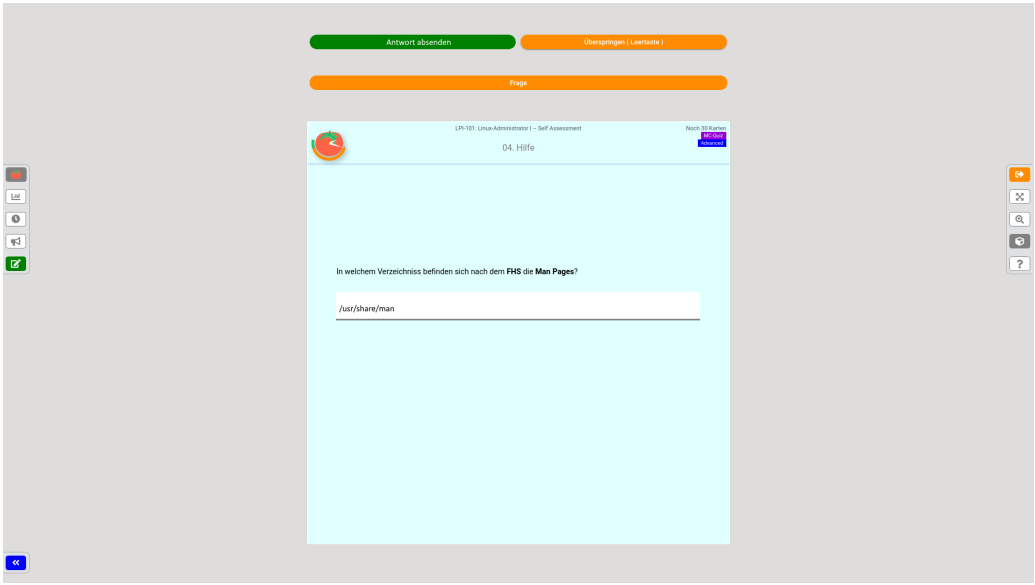


Figure 6.2.: The placement for the input field of a command line question

In the edit view, the card creator has a new tab to jump to the input field. The information for the input field is either stored in the database object for multiple-choice questions, or a new object is created that is only responsible for command line answers. A card cannot be saved until there are is a question and a solution.

```
1 "answers": { // Stored inside the card.
2   ...
3   "textAnswer": String, // Will be compared against the input of the user to determine if
        the answer was correct.
4 }
```

Listing 6.2: Additional field in Card answers object

6.4. NGINX error pages

When a change got deployed to production, it can take some time until the web app is accessible again, depending on the change. For migrations of old data, the waiting times can take several minutes. As long as the web app is booting up, the user only sees an NGINX [Inc21c] error page, which gives the appearance that the server has crashed. To give the user more information, Nginx should enable certain static info pages during this period:

- **Update:** Redirects to an update page while the server is booting up. Once the server is active, the .cards web app is getting displayed.
- **Maintenance:** Can be manually enabled by a system admin when maintenance has to be done on the server.
- **Unavailable:** Used by NGINX on 505 messages. Either when the web app crashed or is unavailable due to other situations not covered by an update or maintenance.

At first, it was first considered that also for 404 messages a special NGINX page can be used. However, doing so will restrict the access to Meteor functionality such as listing that specific database content could not be found. The only possibility, in this case, is to use the error handling page provided by the FlowRouter [Ltd17].

6.5. Content moderation

Installations like the ones found on arsnova.cards allow users to create their content and release it to the public. To guarantee that the content of these card sets is appropriate requires an integration of a moderation system. This system works like the review system for pro card sets, but it is applied to all access categories except “Private”. When a publication is made, a reviewer or admin will receive a notification within the web app, through which they can navigate to the content. The person responsible for the content will then not be able to edit it until the active review is completed. If a card set has been published but needs to be reworked, the user can have a new review done for the new content. The reviewer then sees in a new route the changes that are to be applied and can decide whether they may be incorporated. As long as the review process is active in this case, the other users will only see the previously published content.

6.6. Automatic database cleanups

The web app on arsnova.cards has been used for many students to submit their homework in web-based information systems and software engineering modules. The last activity for a majority of students was several years ago. This does not have a big impact on performance for the publicly visible area, but the amount of data in the backend is noticeable. In this case, it would make sense to introduce a system that automatically deletes inactive content after a certain amount of inactivity.

A new menu item should be introduced in the backend called“ Cleanup & Maintenance”, which allows the admin to define which content is going to be automatically deleted. Settings for these include:

- How long is a user inactive before their content is removed from the database?
- How many days before the user should be informed by e-mail that their data will be deleted if they do not log back in.

The already existing Meteor method will be used to delete the user. This deletes all private content and keeps the information relevant to the learning statistics. The admin has two additional controls to disable the automatic deletion or to trigger it manually. For both options, it is possible to show a statistic, which shows the number of affected contents.

6.7. Improved card set editor & navigation

The editor for editing the description of a card set, is currently located inside a modal. There does not exist a preview window. The user has to save his current adjustments to see the changes. Here it would be appropriate to use a new route, which shows the editor on the left side, and a live preview on the right side. Since the editor for the card content already offers such a layout, it would be appropriate to refactor it and make it available for both types.

There is right now a problem with the card set navigation, as has been seen in the previous UX tests from the software engineering module of WS 20/21, with it being a bit too cluttered and unorganized. One suggestion would be to move the bottom navigation to the side of the screen, as is already the case with the card navigation. Users already seem to recognize what functionality is involved by looking at the icons. For mobile devices, we could only offer the icons unless the user wants to expand the entire navigation. As for the desktop view, both icons and the descriptions of the buttons can be kept, as there is enough free space on both the left and right sides at a card set route 6.3. The labels and information dropdown menus also need a redesign so that they no longer look like buttons. Labels may only look like buttons if they have a function to click on: In the case of the label, it is the Error Reporting Status. This only affects the detailed view of a file. In the theme pool, every label must look like a button, because they use a filter function. For the dropdown menus, it would be recommended to remove them completely from the header

and move them to the new sidebar navigation. Instead of a dropdown, a modal with all the necessary information will be displayed.

Figure 6.3.: The new placement for the reworked card set navigation on desktop

6.8. SonarQube

To ensure the code quality of the web app, the staging branch is checked by a specially managed SonarQube installation [SA21]. Only the content of the import and server folder is checked to exclude code from third-party developers for the games/backgrounds or Markdeep from the scan. To allow developers to quickly check if their changes improve the SonarQube scores, a Dockerfile [Inc21b] is also provided to set up a local installation SonarQube for scanning the local .cards application.

However, to make the values match the THM project infrastructure installation, users still need to update the plugins in the SonarQube Marketplace and remove any errors that were reported as false positives. SonarQube does not have support for Blaze HTML templates and therefore may mark them as broken.

```
<template name="parentElement">
  <ul>
    {{> childElement}}
  </ul>
</template>

<template name="childElement">
  <li>Child</li>
</template>
```

Listing 6.3: Template setup will throw “Missing container tag” false positive

According to the April 15 report, 88 bugs exist. 74 of them are related to missing tags and headers for tables. These are only a limitation for screen reader users, but should still be fixed in the future so that people with a visual impairment can navigate the app more optimally.

There are 47 code smells with a debt of 5 hours. Most of the smells are repetitive errors, which is why the debt time could be even shorter than what's being displayed. Due to the lack of automated tests, the coverage is at 0.0%.

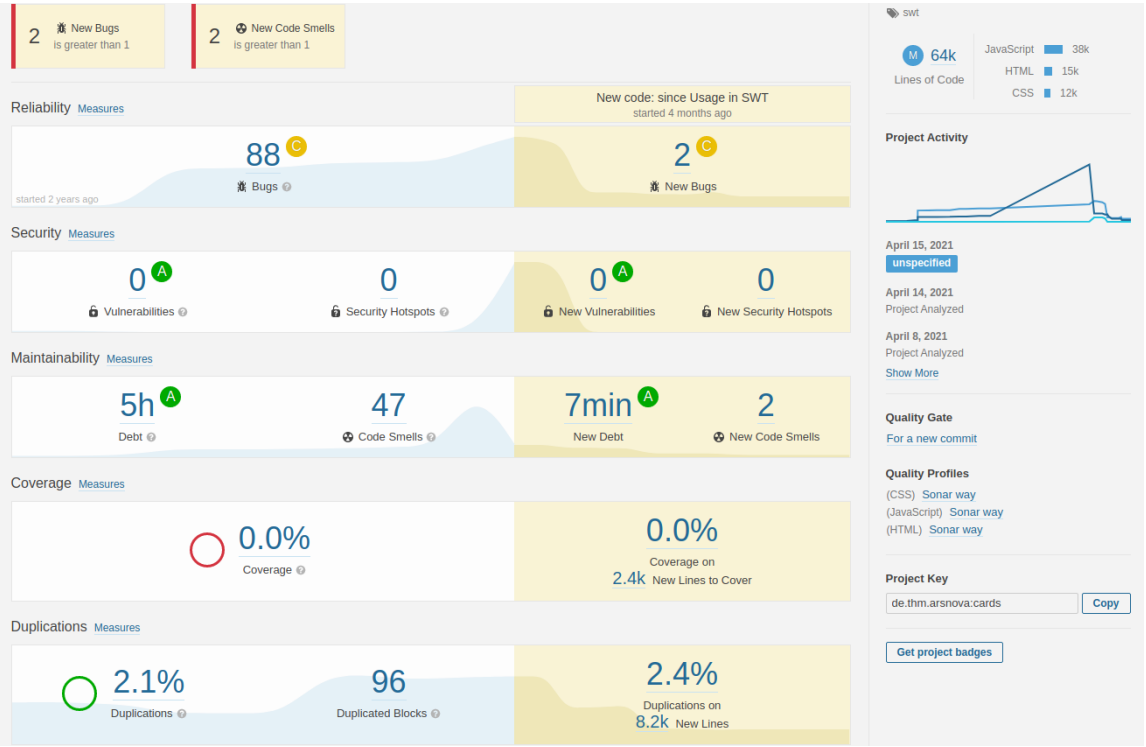


Figure 6.4.: .cards SonarQube stats as of april 15th 2021

6.9. ESLint

Since version 3.0 of .cards, in addition to the use of SonarQube, code analysis by JSCS [Dul16] and JSHint [Ant20] has been added to the CI of the .cards repository to ensure good code quality. If one of these tools fails, then the changes cannot be deployed to production. The currently active rules turn out to be not ideal. They do not distinguish between the use of spaces or tabs, and many checks are disabled because they are not fully compatible with ECMAScript. Furthermore, support for JSCS has been discontinued for years [Ole16], and the old project has been merged with ESLint [Fou21a].

To make development more comfortable in the future, it is recommended to switch to ESLint. Advantages over the current solution would be:

- The integration is better in IDE's like Webstorm [Jet21]. Fewer steps have to be done to use the linter compared to JSCS & JSHINT.
- There is the possibility to convert already existing code automatically, which makes a switch much easier.
- By using widely used rules such as Google guidelines, [LLC20], code could be made cleaner.
- New features like dynamic imports in FlowRouter, which are of great relevance for later optimization, are detected correctly. The old code analysis sees these as errors if they are not at the beginning of a file.

Since .cards is a larger project, it will also take a somewhat large amount of time to convert the entire project to ESLint. That's why it would be recommended to use both linter during the transition period. The app will be converted to ESLint in partial steps: The current folder structure would make this easy and clear, as the transition could be made possible in the following order:

1. /client
2. /imports/api
3. /imports/config
4. /imports/startup
5. /imports/ui
6. imports/util
7. server

After converting the content of these folders to ESLint, the old JSCS & JSHint installation can be removed. It would not be advisable to get rid of all three linter and only use SonarQube. Some rules such as the text indentation seem to have stricter rules in ESLint.

6.10. Automated tests

Initially, there used to be end-to-end tests implemented with chimpy [Xol20]. These had to be removed after later Meteor versions, due to dependency issues with newer Node.js releases. New tests were developed for .cards from an earlier systematic software test module, which are not yet in production. These include integration and end-to-end tests. The current development status can be found in a branch named "sst", which is more than 900 commits behind the current version. E2E testing requires the Robot Framework [Fou21b] and a running .cards instance with backdoor logins. The integration tests use a Docker container [Inc21b], which includes Postman Newman [Inc21d]. For both tests, only the basic framework exists and individual tests still need to be written for the application.

There are two ways to integrate tests into the app: Either complete the state from the sst branch or go back to chimpy and use the rewrite chimp 4.0 [Luk21].

The existing old chimpy tests, which are from previous web-based information systems and software engineering modules, should not be reused for end-to-end testing due to their inefficiency. These mostly consist of testing individual functionalities, such as logging in or creating a card set, which can all be covered by a larger test at the same time. The advantage of larger test runs is that to bypass the logon/logoff process and the resetting of the database, which saves a huge amount of time.

For the integration tests, it is sufficient to copy the state from the sst branch and apply it to all Meteor methods. The focus here is to improve the error handling of the methods so that it automatically checks if a specific action throws a correlated error.

6.11. Performance

Although the web app's range of functions is already mature enough for it to be used productively, there is still a great deal of potential for optimization, which is particularly noticeable on mobile devices with a slow internet connection. This is reflected above all in the Lighthouse tests [LLC21], where the loading of the first view is rated as very negative.

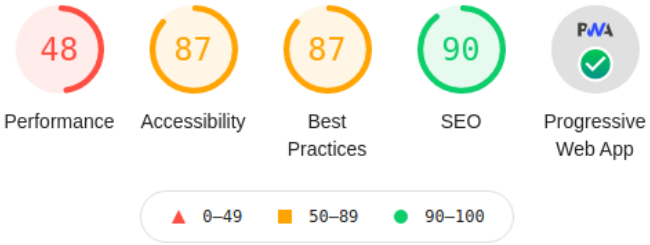


Figure 6.5.: Lighthouse desktop results

Whether a score of 90% to 100% can be achieved with the current framework is questionable. Some changes within the code could result in a faster page load for the client:

- **Topic pool and filter:** The content for each pool view is controlled by subscriptions. When the route is called, the server sends all the required MongoDB database content to the client, which stores it in a MiniMongo DB [Sof21a]. Calling the content then only takes place on the client. As soon as much content exists, e.g. over 1000 entries, the change to this view can take several seconds. The problem does not seem to be the communication between client and server, but the filters themself. Due to the current structure of the filters, the client has to go through the database at least 7 to 10 times, depending on the enabled filter options, before it finishes rendering. One solution for the current client implementation is that the filter selection list is generated only once the filter is clicked, but this would lead to a loading time for opening said filters. A second solution would be that instead of a subscription, only a meteor method is called, as is the case with the list of bonus participants. But, this would have the problem that it would increase the server load and the filter would have to be completely reworked as the client now lacks the information to dynamically generate the content of the dropdown menus.
- **Dynamic Imports**: By using the FlowRouter package [Ltd17], it is now possible to define for each route which JavaScript files are sent to the client. As of now, this feature is not used properly: The majority of files are still included in `imports/ui/main.js`. These would have to move into the correct route files. It could be that further refactoring within the imported files is needed to make this work.

7. Conclusion

The goal of this thesis has been to extend the learning platform .cards in such a way that the learning evaluation is taken over more automatically by the system and the lecturer can see the learning behavior of his students. This goal has been reached in that aspect, the production already uses the first implementation, but there is still a lot of room for improvements. Concepts have already been developed, which still have to be integrated into the web app. One of these concepts that is currently being worked on is a general revision of the Leitner algorithm to retain the information of all previous learning bonus phases, as well as a listing of the most difficult flashcards. Due to the high complexity of the algorithm, it is also imperative that automated tests are created. It has often happened that parts of the application did not work properly for longer periods. After all the new concepts from this thesis have been implemented, we can then begin to fully integrate the Woźniak algorithm into the system based on the Leitner implementation. At this moment it exists in a simple form and does not support repeated learning or the awarding of bonus points.

Another aspect that needs improvement is the creation process of new content. The .cards web app still needs many " Quality of Life" improvements that make the management of the content much easier. A major point that needs increased attention is the sorting of content inside a table of contents. Because these are only sorted by the title and first side of a card, it is very time-consuming to change the order in larger card sets and repetitories. Plus, editing the content itself is sometimes cumbersome: It is necessary to switch between different routes to edit several cards at the same time.

Last but not least, the performance must be improved, especially in the mobile area. Devices with a slow internet connection take a long time to load the web app. Even smaller actions, such as switching between individual cards, can vary greatly in time depending on the size of the content. To what extent this can be optimized depends of course on the Meteor framework, as well as any additional packages used on .cards.

Bibliography

[Ale21] Deveria Alexis. *caniuse*. 2021. URL: https://caniuse.com (visited on 04/29/2021) (cit. on p. 50).

[Ant20] Kovalyov Anton. *JSHint, A Static Code Analysis Tool for JavaScript*. 2020. URL: https://www.npmjs.com/package/jshint (visited on 04/29/2021) (cit. on p. 79).

[Bet18] Ho Betty. *App Retargeting: How it Works and Why You Need It*. 2018. URL: https://www.criteo.com/blog/how-app-retargeting-works/ (visited on 04/29/2021) (cit. on p. 63).

[Che20] The Meteor Chef. *Bert*. Version 2.2.1. 2020. URL: https://github.com/themeteorchef/bert (visited on 04/29/2021) (cit. on p. 44).

[Che21] TU Chemnitz. *BEOLINGUS - Ein Service der TU Chemnitz unterstützt von IBS und MIOTU/Mio2*. 2021. URL: https://dict.tu-chemnitz.de (visited on 04/29/2021) (cit. on p. 20).

[Chr17] Mather Chris. *Iron.Router*. 2017. URL: https://github.com/iron-meteor/iron-router (visited on 04/29/2021) (cit. on p. 61).

[Cod21] CodePen. *CodePen*. 2021. URL: https://codepen.io (visited on 04/29/2021) (cit. on p. 50).

[CR14] Mather Chris and Walker Ry. *Meteor handlebars server*. Version 1.2.0. 2014. URL: https://github.com/cmather/meteor-handlebars-server (visited on 04/29/2021) (cit. on p. 40).

[Dam21] Elmes Damien. *Anki*. Version 2.1.43. 2021. URL: https://apps.ankiweb.net (visited on 04/29/2021) (cit. on p. 21).

[Dav21] Deutsch David. *Owl Carousel 2*. 2021. URL: https://owlcarousel2.github.io/OwlCarousel2 (visited on 04/29/2021) (cit. on p. 51).

[Dul16] Marat Dulin. *JSCS*. 2016. URL: https://www.npmjs.com/package/jscs (visited on 04/29/2021) (cit. on p. 79).

[Fc21a] JS Foundation and other contributors. *Moment.js*. 2021. URL: https://momentjs.com (visited on 04/29/2021) (cit. on p. 31).

[Fc21b] OpenJS Foundation and jQuery contributors. *jQuery UI - Draggable*. 2021. URL: https://jqueryui.com/draggable/#sortable (visited on 04/29/2021) (cit. on p. 73).

[Fou21a] JS Foundation. *ESLint*. 2021. URL: https://www.npmjs.com/package/eslint (visited on 04/29/2021) (cit. on p. 79).

[Fou21b] Robot Framework Foundation. *ROBOT FRAME WORK*. 2021. URL: https://robotframework.org/ (visited on 04/29/2021) (cit. on p. 80).

[Gar08] Wolf Gary. *Want to Remember Everything You'll Ever Learn? Surrender to This Algorithm*. 2008. URL: https://www.wired.com/2008/04/ff-wozniak/ (visited on 04/29/2021) (cit. on p. 3).

[Gmb21] DeepL GmbH. *DeepL Übersetzer*. 2021. URL: https://www.deepl.com (visited on 04/29/2021) (cit. on p. 20).

[Inc13] Twitter Inc. *Bootstrap 3*. Version 3.3.7. 2013. URL: https://getbootstrap.com/docs/3.3/ (visited on 04/29/2021) (cit. on pp. 8, 40, 44, 62).

[Inc21a] BlueButton Inc. *BigBlueButton*. 2021. URL: https://bbb.arsnova.eu (visited on 04/29/2021) (cit. on p. 63).

[Inc21b] Docker Inc. *Docker*. 2021. URL: https://www.docker.com (visited on 04/29/2021) (cit. on pp. 77, 80).

[Inc21c] F5 Inc. *NGINX*. 2021. URL: https://www.nginx.com (visited on 04/29/2021) (cit. on p. 75).

[Inc21d] Postman Inc. *POSTMAN Learning Center - Running collections on the command line with Newman*. 2021. URL: https://learning.postman.com/docs/running-collections/using-newman-cli/command-line-integration-with-newman (visited on 04/29/2021) (cit. on p. 80).

[Inc21e] Quizlet Inc. *Quizlet*. 2021. URL: https://quizlet.com (visited on 04/29/2021) (cit. on p. 23).

[Jer+21] Ashkenas Jeremy, Gonggrijp Julian, DocumentCloud, Investigative Reporters, and Editors. *Underscore.js*. 2021. URL: https://underscorejs.org (visited on 04/29/2021) (cit. on p. 31).

[Jet21] JetBrains. *Webstorm - Eslint*. 2021. URL: https://www.jetbrains.com/help/webstorm/eslint.html (visited on 04/29/2021) (cit. on p. 79).

[Joh18] Wilkos John. *Pomodoro Timer*. 2018. URL: https://codepen.io/kevyman/pen/ZGNBBN (visited on 04/29/2021) (cit. on p. 47).

[Jon+21] Bunke Jonathan, Bramabrata Reinhard Ganesha, Münch Dominik, Gode Marcel, Tchiabe Ngouabe Loic Maxwell, Mursall Maik, Kerkmann Daniel, Alayoub Mohammad, Spengler Daniel, Berg Janek, Grünwald Fabian, and Jensen Benedikt. *UI/UX-Testberichte der App "arsnova.cards"*. Tech. rep. MNI, 2021 (cit. on p. 63).

[jon21] Jos de jong. *Math.js*. 2021. URL: https://mathjs.org (visited on 04/29/2021) (cit. on p. 31).

[Lim16] TAPevents Asia Limited. *tap-i18n*. 2016. URL: https://github.com/TAPevents/tap-i18n (visited on 04/29/2021) (cit. on p. 41).

[LLC20] Google LLC. *eslint-config-google*. 2020. URL: https://github.com/google/eslint-config-google (visited on 04/29/2021) (cit. on p. 79).

[LLC21] Google LLC. *Lighthouse*. 2021. URL: https://developers.google.com/web/tools/lighthouse (visited on 04/29/2021) (cit. on p. 81).

[Lok21] Dhakar Lokesh. *LIGHTBOX*. 2021. URL: https://lokeshdhakar.com/projects/lightbox2 (visited on 04/29/2021) (cit. on p. 56).

[Ltd17] MeteorHacks Pvt Ltd. *FlowRouter*. Version 2.12.1. 2017. URL: https://github.com/kadirahq/flow-router (visited on 04/29/2021) (cit. on pp. 40, 46, 47, 62, 75, 81).

[Luc17] Lauretta Luca. *MultiDatesPicker for jQuery UI*. 2017. URL: https://dubrox.github.io/Multiple-Dates-Picker-for-jQuery-UI/ (visited on 04/29/2021) (cit. on p. 10).

[Luk21] Gandecki Lukasz. *chimp*. 2021. URL: https://github.com/xolvio/chimp (visited on 04/29/2021) (cit. on p. 80).

[Mar21] Heiderich Mario. *DOMPurify*. 2021. URL: https://github.com/cure53/DOMPurify (visited on 04/29/2021) (cit. on p. 56).

[Mor21] McGuire Morgan. *Markdeep*. Version 1.13. 2021. URL: https://casual-effects.com/markdeep/ (visited on 04/29/2021) (cit. on pp. 3, 56).

[Moz21a] Mozilla. *Fullscreen API*. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fullscreen_API (visited on 04/29/2021) (cit. on p. 58).

[Moz21b] Mozilla. *Using Web Workers*. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (visited on 04/29/2021) (cit. on p. 72).

[Ole16] Gaidarenko Oleg. *JSCS — end of the line*. 2016. URL: https://medium.com/@markelog/jscs-end-of-the-line-bc9bf0b3fdb2#.x0o1xqkk9 (visited on 04/29/2021) (cit. on p. 79).

[Pet21] Bientsman Peter. *The Mnemosyne Project*. Version 2.7.3. 2021. URL: https://mnemosyne-proj.org (visited on 04/29/2021) (cit. on p. 17).

[SA21] SonarSource S.A. *.cards SonarQube Quality Gate*. 2021. URL: https://scm.thm.de/sonar/dashboard?id=de.thm.arsnova%3Acards (visited on 04/29/2021) (cit. on p. 77).

[Sof16] Meteor Software. *User Accounts*. Version 1.14.2. 2016. URL: https://github.com/meteor-useraccounts (visited on 04/29/2021) (cit. on pp. 40, 41).

[Sof21a] Meteor Software. *Documentation on how to use Meteor's database collections*. 2021. URL: https://docs.meteor.com/api/collections.html (visited on 04/29/2021) (cit. on pp. 72, 81).

[Sof21b] Meteor Software. *Meteor*. 2021. URL: https://www.meteor.com (visited on 04/29/2021) (cit. on p. 3).

[Sof21c] Meteor Software. *server-render*. 2021. URL: https://docs.meteor.com/packages/server-render.html (visited on 04/29/2021) (cit. on p. 60).

[Stu21] Studybees. *SPACED REPETITION – KARTEIKARTEN MIT SYSTEM*. 2021. URL: https://studybees.de/magazin/spaced-repetition-karteikarten-mit-system/ (visited on 04/29/2021) (cit. on p. 3).

[Tea19] The Wekan Team. *Meteor-Accounts-CAS*. 2019. URL: https://github.com/wekan/meteor-accounts-cas (visited on 04/29/2021) (cit. on p. 40).

[Tea21a] ARSnova Team. *arsnova-click-v2*. 2021. URL: https://git.thm.de/arsnova/arsnova-click-v2-frontend (visited on 04/29/2021) (cit. on p. 26).

[Tea21b] ARSnova Team. *ARSnova.cards*. Version 3.14. ARSnova. 2021. URL: https://git.thm.de/arsnova/cards (visited on 04/29/2021) (cit. on p. 1).

[Tea21c] ARSnova Team. *frag.jetzt*. 2021. URL: https://frag.jetzt (visited on 04/29/2021) (cit. on p. 52).

[Tea21d] The MathJax Team. *MathJax*. 2021. URL: https://www.mathjax.org (visited on 04/29/2021) (cit. on p. 56).

[Tea21e] UEQ Team. *User Experience Questionnaire*. 2021. URL: https://www.ueq-online.org (visited on 04/29/2021) (cit. on pp. 63, 68).

[Tim15] Gu Timothy. *knuth-shuffle-seeded*. Version 1.0.6. 2015. URL: https://www.npmjs.com/package/knuth-shuffle-seeded (visited on 04/29/2021) (cit. on p. 45).

[TL21] Edwards Tristan and Monte Limon. *sweetalert2*. Version 10.16.6. 2021. URL: https://sweetalert2.github.io (visited on 04/29/2021) (cit. on p. 48).

[Uwe05] Frommann Uwe. *Die Methode „Lautes Denken“*. 2005. URL: https://www.e-teaching.org/didaktik/qualitaet/usability/Lautes%20Denken_e-teaching_org.pdf (visited on 04/29/2021) (cit. on p. 63).

[Web18] WebAIM. *WAVE Web Accessibility Evaluation Tool*. 2018. URL: https://wave.webaim.org (visited on 04/29/2021) (cit. on p. 65).

[WP21] SuperMemo World and Woźniak Piotr. *SuperMemo*. 2021. URL: https://www.supermemo.com (visited on 04/29/2021) (cit. on p. 19).

[Xol20] Xolv.io. *chimpy*. 2020. URL: https://github.com/TheBrainFamily/chimpy (visited on 04/29/2021) (cit. on p. 80).

[Yeh21] Katz Yehuda. *Handlebars.js*. Version 4.7.7. 2021. URL: https://handlebarsjs.com (visited on 04/29/2021) (cit. on p. 40).

List of Figures

List of Tables

Listings

Appendices

This appendix includes diagrams, descriptions of the MongoDB documents, and statistics. In addition, a translation table has been included to help programmers with the further development of the web app.

A. PlantUML diagrams

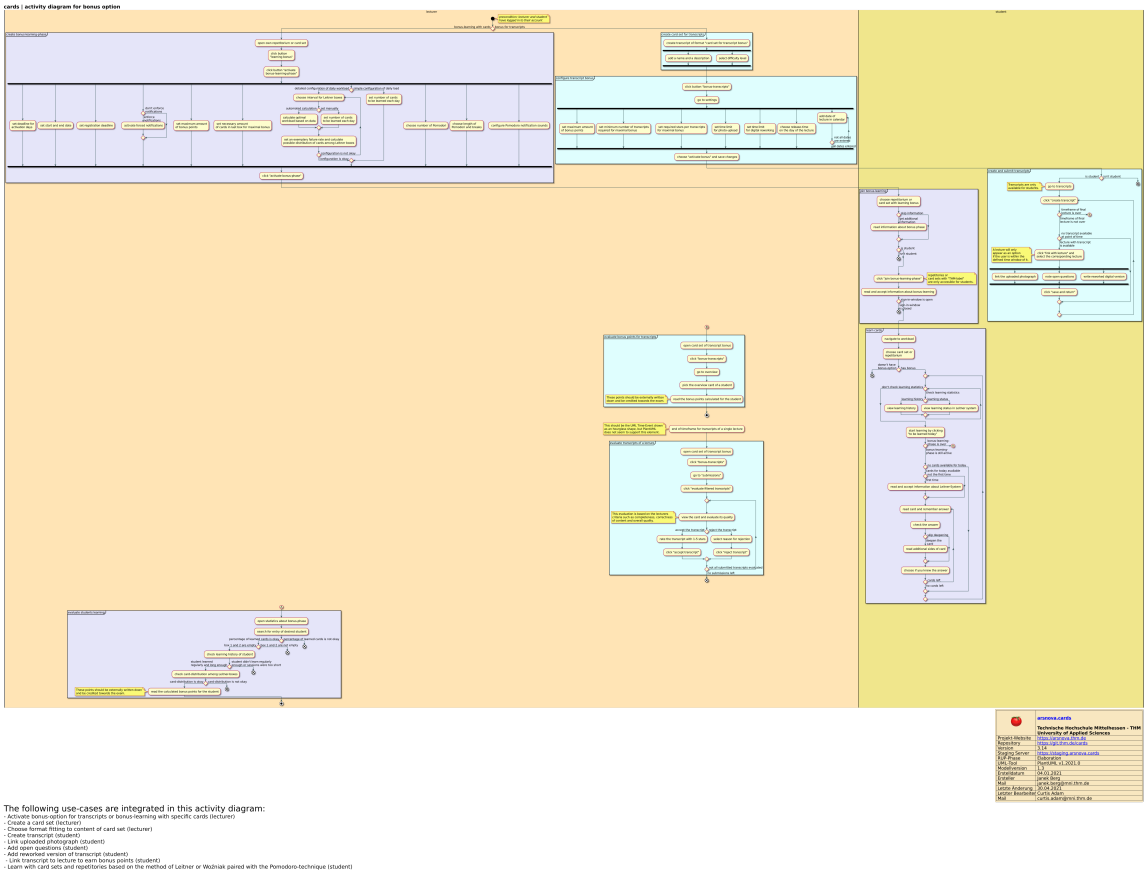


Figure 1.: PlantUML: Activity diagram for bonus option

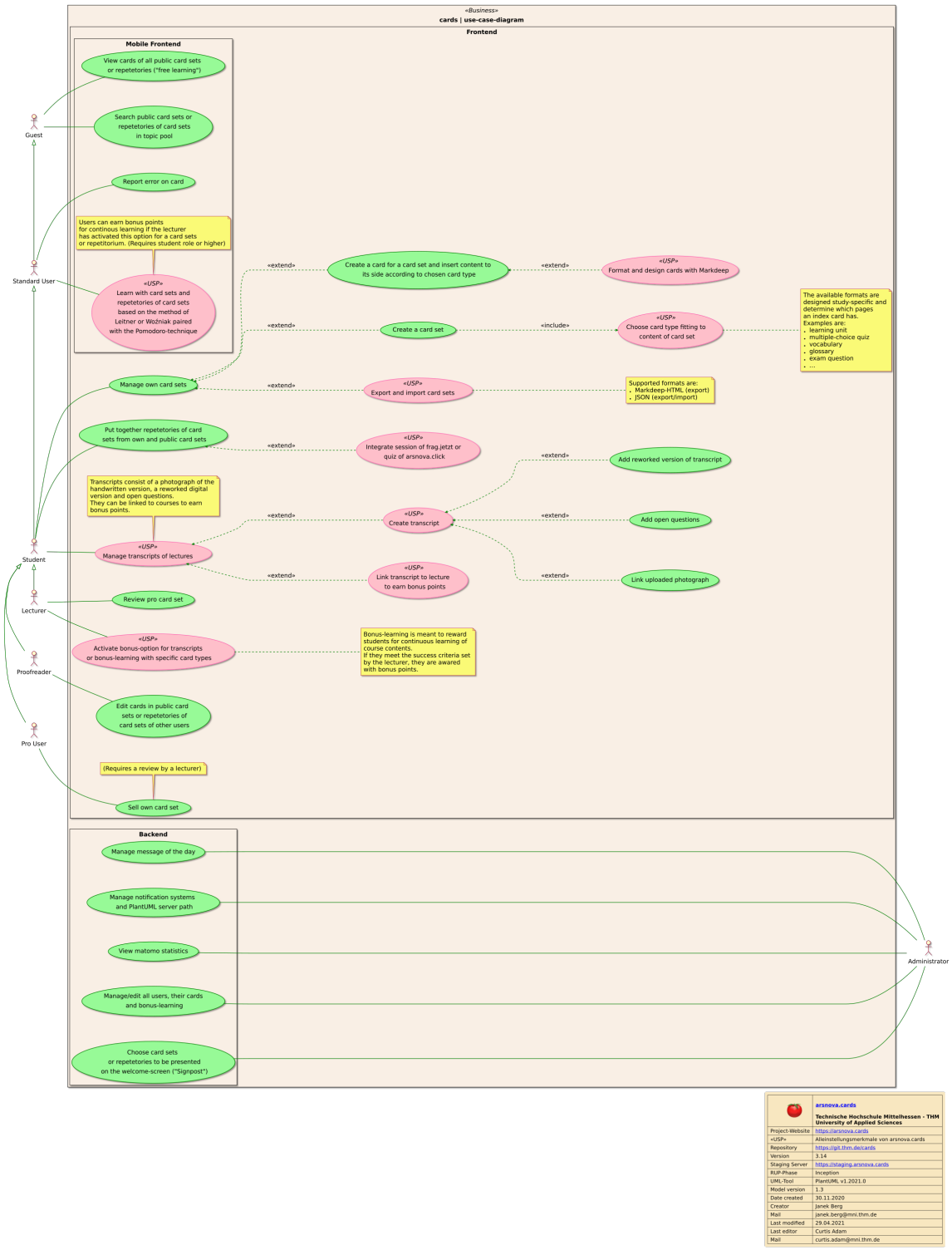


Figure 2.: PlantUML: Use cases diagram

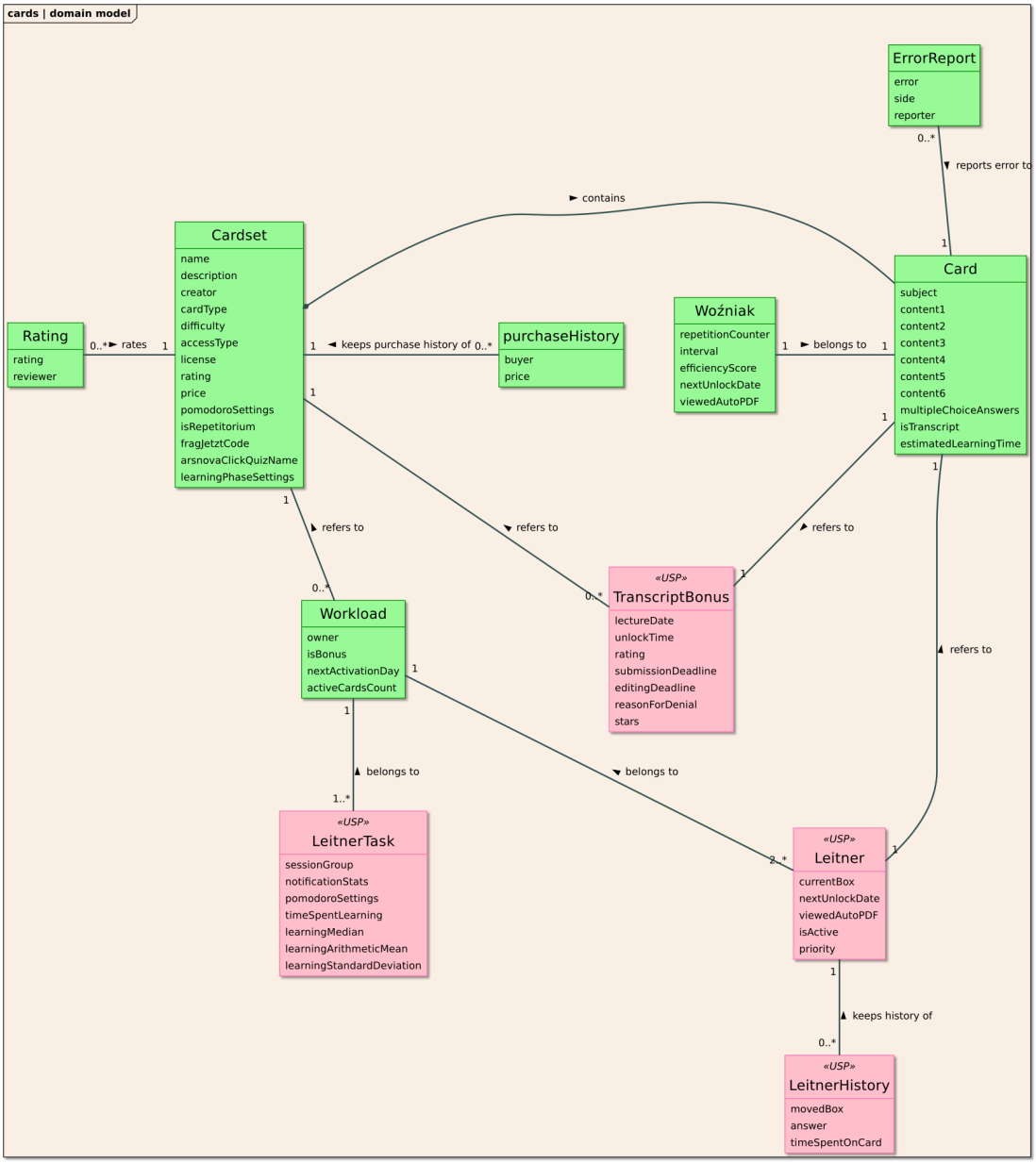


Figure 3.: PlantUML: Domain model

B. Open Hub stats

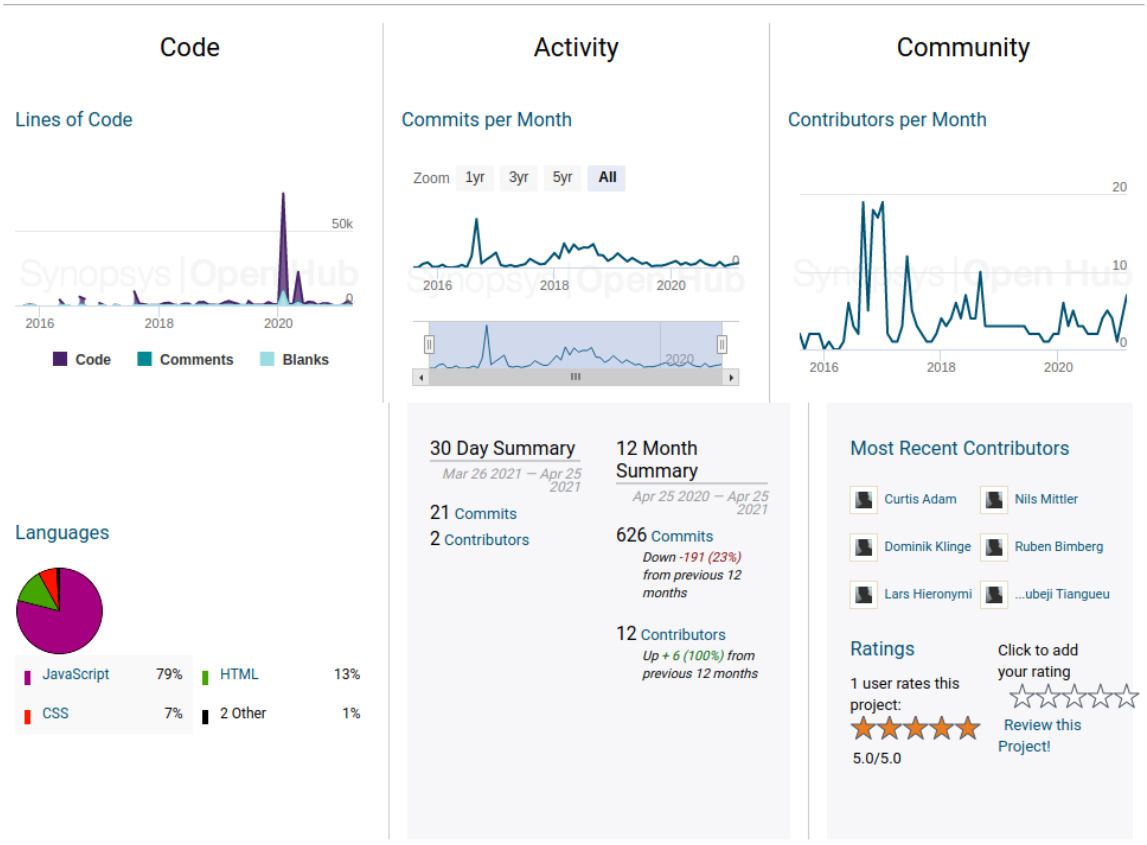


Figure 4.: .cards Open Hub stats

C. MongoDB document descriptions of .cards v3.14

This section contains a listing of all MongoDB documents currently in use on .cards. It describes what the documents are used for and what they contain.

C.A. AdminSettings

Saves all settings that can be found in the backend area under the category “Settings”. As an example, the sending of test notifications is listed here.

Field	Type	Description
_id	STRING	Primary key
name	STRING	Name for the settings file
target	STRING	User ID to send the message to
testCardsetID	STRING	Cardset for the test data
testUserID	STRING	User for the test data

Table 2.: MongoDB adminSettings document description

C.B. Cards

Contains all information about a card. This document is used for normal cards and transcripts.

Field	Type	Description
_id	STRING	Primary key
cardset_id	STRING	Foreign key of the card set
subject	STRING	The title of the card
centerTextElement	[BOOLEAN]	Centers the text vertically for each side
alignType	[INTEGER]	Sets the text alignment for each side
date	DATE	The creation timestamp
dateUpdate	DATE	The last modification timestamp
lastEditor	STRING	The ID of the user who edited the card the last time
backgroundStyle	INTEGER	Display the background with or without lines
owner	STRING	ID of the owner / creator
originalAuthorName	OBJECT	Used to display the original creator for imported cards
cardType	INTEGER	The card type selected through the card set
learningTime	OBJECT	Estimate of how long a user might need to learn the card (Used in the future for Leitner
answers	OBJECT	Contains the multiple-choice content
front	STRING	Content element Nr. 1
back	STRING	Content element Nr. 2
hint	STRING	Content element Nr. 3
lecture	STRING	Content element Nr. 4
top	STRING	Content element Nr. 5
bottom	STRING	Content element Nr. 6

Table 4.: MongoDB cards document description

C.C. Cardsets

Contains all information about a card set. This document is used card sets and repetitories.

Field	Type	Description
_id	STRING	Primary key
name	STRING	The title of the card set
description	STRING	The description of the card set
date	DATE	The creation timestamp
dateUpdate	DATE	The last modification timestamp
lastEditor	DATE	The ID of the user who edited the card the last time
owner	STRING	ID of the owner / creator
originalAuthorName	OBJECT	Used to display the original creator for imported card set
cardType	INTEGER	The card type that all cards should use
visible	BOOLEAN	Turns the card set invisible if a review is pending
ratings	BOOLEAN	Do ratings exist for this card set?
kind	STRING	The Access category (PRIVATE, FREE, EDU, PRO)
price	DOUBLE	Used for PRO card sets
reviewed	BOOLEAN	Did a lecturer review the PRO card set?
reviewer	STRING	The user ID of the reviewer
request	BOOLEAN	Is a review request pending?
quantity	INTEGER	The number of cards that belong to this card set
license	[STRING]	Active licenses
userDeleted	BOOLEAN	Was the user deleted from the database?

Table 6.: MongoDB card set set document description part 1

Field	Type	Description
learningActive	BOOLEAN	Is a learning phase bonus in progress?
maxCards	INTEGER	The maximum number of cards a Leitner activation date can have
learningStart	DATE	The date when the learning phase bonus began
learningEnd	DATE	The date when the learning phase bonus ends
registrationPeriod	DATE	The date before users can no longer join the learning phase bonus
learningInterval	[INTEGER]	The intervals of the Leitner boxes
wordcloud	BOOLEAN	Should the card set be displayed on the landing page word cloud?
shuffled	BOOLEAN	Is the card set a repetitorium?
cardGroups	[STRING]	A list of ID's of card sets for the repetitorium
raterCount	INTEGER	Number of unique users who have rated this card set
rating	INTEGER	The overall rating that is given to the card set
difficulty	INTEGER	The difficulty to be used for the cards
noDifficulty	INTEGER	Does the card set have a difficulty setting?
workload	OBJECT	General statistics about the active learning phase bonus
sortType	INTEGER	How is the table of contents sorted?
gotWorkload	BOOLEAN	Does Leitner workload data exist?
lecturerAuthorized	BOOLEAN	Is this card set recommended by lecturers?
useCases	OBJECT	Should the card set appear in the welcome screen and which priority does it have?
arsnovaClick	OBJECT	Link the to arsnova.click session and should the repetitorium override the card set settings?
fragJetzt	OBJECT	Link to the frag.jetzt session and should the repetitorium override the card set settings?
transcriptBonus	OBJECT	Includes settings such as lecture dates, maximum bonus, and deadlines for transcript bonus card types

Table 8.: MongoDB card set set document description part 2

C.D. ErrorReporting

Stores all the error reports that can be assigned to a card.

Field	Type	Description
_id	STRING	Primary key
card_id	STRING	Foreign key of the card
cardset_id	STRING	Foreign key of the card set
user_id	STRING	Foreign key of reporter
createdAt	DATE	Timestamp of when the error report was sent
status	INTEGER	Is the error resolved or unresolved?
cardSide	INTEGER	The card side that the error is located at
error	OBJECT	The type of error that is being reported

Table 10.: MongoDB errorReporting document description

C.E. Leitner

Stores information about the state of the individual cards of users who are in a Leitner learning phase. This information gets lost once the user leaves the learning phase.

Field	Type	Description
_id	STRING	Primary key
card_id	STRING	Foreign key of the card
cardset_id	STRING	Foreign key of the card set that the learning phase belongs to
original_cardset_id	STRING	Foreign key of the original card set if the learning phase belongs to a repetitorium
user_id	STRING	Foreign key of the user
box	INTEGER	The box in which the card is located
active	BOOLEAN	Does the user have to learn this card before the next deadline is reached?
nextDate	DATE	The date on which the card can be used again to be assigned to an activation date
currentDate	DATE	The date on which the card got activated. Only relevant for still active cards and the deadline
priority	INTEGER	When should the card be used again compared to others in the same box?
viewedPDF	BOOLEAN	Did the card automatically open the PDF at least once and did the user close it?

Table 12.: MongoDB leitner document description

C.F. LeitnerHistory

Stores information about the answer history during a learning phase. This information will be stored even if the user leaves a learning phase.

Field	Type	Description
_id	STRING	Primary key
card_id	STRING	Foreign key of the card
cardset_id	STRING	Foreign key of the card set that the learning phase belongs to
task_id	STRING	Foreign key of the Leitner activation day that this answer belongs to
original_cardset_id	STRING	Foreign key of the original card set if the learning phase belongs to a repetitorium
user_id	STRING	Foreign key of the user
box	INTEGER	The box to which the card got moved to
skipped	INTEGER	How many times did the user skip this answer before answering it?
mcAnswers	OBJECT	The selected answers for a multiple-choice question and the correct answers of the card
answer	INTEGER	Was the answer correct or wrong?
timestamps	OBJECT	The amount of time the user took to answer the card split between viewing the question, seeing the answer, and moving on to the next card
missedDeadline	BOOLEAN	Was this card moved cause of a missed deadline? (In this case, the answer will be marked as wrong)

Table 14.: MongoDB leitnerHistory document description

C.G. LeitnerTasks

Stores information about a user's activation date during a learning phase. This information will stay inside the database if the user leaves the learning phase.

Field	Type	Description
_id	STRING	Primary key
card_id	STRING	Foreign key of the card
cardset_id	STRING	Foreign key of the card set that the learning phase belongs to
user_id	STRING	Foreign key of the user
session	INTEGER	Identifier to group activation dates together between different learning phases on the same card set
isBonus	BOOLEAN	Is the activation date part of a bonus or is it a private learning phase?
missedDeadline	BOOLEAN	Did the user miss the deadline or did he manage to complete all active cards?
resetDeadlineMode	INTEGER	How did the deadline move the unanswered cards?
wrongAnswerMode	INTEGER	How did the app move the cards if the answer was incorrect?
notifications	OBJECT	Did the user enable e-mail/push notifications, did the server sent them out successfully, and to which address?
createdAt	DATE	The timestamp of when the cronjob created the activation date
pomodoroTimer	OBJECT	The Pomodoro timer settings during the creation of the activation date
strictWorkloadTimer	BOOLEAN	Is the strict workload timer active for this date
timer	OBJECT	How long did the user spend learning during this date?
learningStatistics	OBJECT	Statistics about working and answering times

Table 16.: MongoDB leitnerTask document description

C.H. MessageOfTheDay

Used to store the information of the message of the day that can be display on the landing page and frontend.

Field	Type	Description
_id	STRING	Primary key
subject	STRING	The title of the MOTD
content	STRING	The content of the MOTD
dateCreated	DATE	The timestamp of when the MOTD was created
dateUpdated	DATE	The last modification timestamp
locationType	INTEGER	Where should the MOTD be displayed? (Landing page, frontend, both)
expirationDate	DATE	The timestamp of when the MOTD will be deactivated
publishDate	DATE	The timestamp of when the MOTD will be published

Table 18.: MongoDB messageOfTheDay document description

C.I. Paid

Used to store which card sets a user purchased.

Field	Type	Description
_id	STRING	Primary key
cardset_id	STRING	Foreign key of the purchased card set
user_id	STRING	Foreign key of the user who purchased the card set
date	DATE	The timestamp at which the purchase was made
amount	DOUBLE	How much money was spent to purchase the card set?

Table 20.: MongoDB paid document description

C.J. Ratings

Information about the individual ratings given to card sets.

Field	Type	Description
_id	STRING	Primary key
cardset_id	STRING	Foreign key of the rated card set
user_id	STRING	Foreign key of the user who rated the card set
rating	INTEGER	The rating that was given by the user

Table 22.: MongoDB ratings document description

C.K. TranscriptBonus

Stores information about individual transcripts of users who participate at a transcript bonus.

Field	Type	Description
_id	STRING	Primary key
card_id	STRING	Foreign key of the transcript
cardset_id	STRING	Foreign key of the card set with a transcript bonus
user_id	STRING	Foreign key of the user who submitted the transcript
date	DATE	The date of the lecture
deadline	INTEGER	How many hours until the user can't submit his transcript to the lecture?
deadlineEditing	INTEGER	How many hours until the user can't edit submitted his transcript?
lectureEnd	STRING	The time at which the lecture does end
rating	STRING	The rating that was given by the lecturer
reason	[STRING]	The reason why the transcript got denied
stars	INTEGER	The calculated stars by the system based on other card set settings

Table 24.: MongoDB transcriptBonus document description

C.L. Users

The user account of the web app. This gets autogenerated by the Meteor login services.

Field	Type	Description
_id	STRING	Primary key
createdAt	DATE	Timestamp of when the account was created
services	OBJECT	Information about the login service being used to create this account
profile	OBJECT	Information that's made visible to the public such as names and e-mail address
roles	[STRING]	The assigned user roles (FIRSTLOGIN, BLOCKED, FREE, EDU, LECTURER, PRO, EDITOR, ADMIN)
status	OBJECT	Is the user online and when was his last login?
visible	BOOLEAN	Is the profile information visible to the public?
lastOnAt	DATE	Timestamp of when the user was online at the last time
selectedColorTheme	STRING	Which theme did the user select?
mailNotification	BOOLEAN	Did the user activate e-mail notifications?
webNotification	BOOLEAN	Did the user activate push notifications?
balance	DOUBLE	The user's current credit
count	OBJECT	Counter of owned card sets, active workloads, transcripts, and bonus transcripts
fullscreen	OBJECT	The preferred fullscreen settings for individual routes
motds	[STRING]	The ID's of already confirmed MOTD's

Table 26.: MongoDB users document description

C.M. WebPushSubscriptions

Used for the push notification service.

Field	Type	Description
_id	STRING	Primary key
user_id	STRING	Foreign key of the user the subscription belongs to
subscriptions	OBJECT	Contains the endpoint and authentification keys

Table 28.: MongoDB webPushSubscriptions document description

C.N. Workload

Stores info about global statistics of the user's learning phase. This information gets lost if the user starts a new learning phase or leaves the currently active one.

Field	Type	Description
_id	STRING	Primary key
cardset_id	STRING	Foreign key of the card set that the learning phase belongs to
user_id	STRING	Foreign key of the user
leitner	OBJECT	Stores information such as when the user joined, if the learning phase is a bonus, how many cards are active at the moment, and when the next activation date will occur once the user completed his active workload

Table 30.: MongoDB workload document description

C.O. Woźniak

Stores information about the state of the individual cards of users who are in a Woźniak learning phase. This information gets lost once the user leaves the learning phase.

Field	Type	Description
_id	STRING	Primary key
card_id	STRING	Foreign key of the card
cardset_id	STRING	Foreign key of the card set that the learning phase belongs to
original_cardset_id	STRING	Foreign key of the original card set if the learning phase belongs to a repetitorium
user_id	STRING	Foreign key of the user
ef	DOUBLE	The efficiency score
interval	INTEGER	The interval at which the card will be presented
reps	INTEGER	The repetition counter
nextDate	DATE	Timestamp at which the card first can be presented again
viewedPDF	BOOLEAN	Did the card automatically open the PDF at least once and did the user close it?

Table 32.: MongoDB wozniak document description

D. Translation table

English	German	.cards 3.0 (german)	Code	Database
Card	Karte	Karte	card	cards
Card set	Kartei	Kartensatz	cardset	cardsets
Repetitorium	Repetitorium	Gemischter Kartensatz	repetitorium, cardset.shuffled	cardsets
Transcript	Lose Mitschrift		transcritp	cards
Transcript bonus	Gekoppelte Mitschrift		transcript	transcriptBonus
Leitner	Leitner	Lernkartei	leitner, box	leitner
Woźniak	Woźniak	Memo	wozniak, memo	wozniak
Learning log	Lernverlauf		learningLog	leitnerHistory
Learning history	Lernprotokoll		learningHistory	leitnerHistory
Activation date	Freischaltungstag	Termin, Aufgabe	task	leitnerTasks
Learning workload	Lernpensum	Lernpensum	workload	workload
Card type	Karteityp	Kartentyp	cardType	
User	Account	Benutzer	user	users
Card set index	Karteiverzeichnis		cardsetIndex	
Word cloud	Wortkwolke		wordcloud	wordcloud
Ratings	Bewertung	Bewertung	rating	ratings
Presentation	Kartenschau		presentation	
Table of contents	Kartenverzeichnis	Liste	cardIndex	cards
Pool	Themenpool	Pool	filter (Includes pool, my cardsets, workload, transcripts, all cardsets)	cards, cardsets, workload
Notifications	Benachrichtigungen	Benachrichtigungen	notifications	notifications
Learning status	Lernstand		graph, learningStatus	
Pomodoro	Pomodoro		pomodoro	
Signpost	Hauptziele		useCases	

Table 33.: .cards translation table