

A WEB-BASED PLANNING TOOL FOR GERMAN SCHOOL TEACHERS

Web-basierte Planungsprozesse für Lehrende an deutschen Schulen

GEORG BERECH

georg.berecz@student.hpi.de

A Master's Thesis for attainment of the academic degree

Master of Science in IT-Systems Engineering

SUPERVISORS

Prof. Dr. Christoph Meinel

Dipl.-Inf. (FH) Jan Renz

CHAIR

Internet Technologies and Systems

Hasso Plattner Institute

University of Potsdam

Potsdam, the 08. December 2018

Georg Berecz:

A Web-Based Planning Tool for German School Teachers

Web-basierte Planungsprozesse für Lehrende an deutschen Schulen

SUPERVISORS:

Prof. Dr. Christoph Meinel

Dipl.-Inf. (FH) Jan Renz

CHAIR:

Internet Technologies and Systems

Hasso Plattner Institute

University of Potsdam

SUBMITTED ON:

08. December 2018

ABSTRACT

German school teachers only spent a part of their work time in lessons. The other part consists of preparation and organizational tasks.

All of Germany's teachers are obliged to implement a specific core curriculum, which differs in every state, and which is complemented by a school internal curriculum. To incorporate these requirements for every class of their school year, they create learning matter distribution plans to plan ahead. This includes much manual labor: Teachers will calculate the total amount of forthcoming lessons by counting the number of weeks and incorporating public holidays, vacation times and school internal events. Finally, the topic blocks can be distributed. The reuse of these plans is a common way to reduce the effort. Nonetheless, individual class profiles and unforeseen events during a school year, demand adjustments to the plan.

This thesis demonstrates that teachers currently rely on a very individual process to compile a school year plan: Most often table structures in Microsoft Word or pen and paper are used, even if software-based lesson management solutions are available in their schools. An examination of most popular school software showed that all lack convenience and support for this kind of task.

Thus, a concept for a web-based planning tool is proposed that aims to reduce teachers' manual labor and allows them to dynamically react to changes of events while retaining an overview of the current status. Furthermore, the availability of this information in the cloud could be used to create more synergies.

An implementation of the concept was integrated into the Schul-Cloud. The following evaluation and qualitative user research show that teachers do not only perceive the usability to be outstanding but that there is high potential in the general approach.

ZUSAMMENFASSUNG

Die Arbeitszeit von Lehrenden an deutschen Schulen besteht lediglich zu einem Teil aus eigentlicher Lehrtätigkeit. Der andere Teil wird für organisatorische Aufgaben und die Unterrichtsvorbereitung benötigt.

Der bundeslandspezifische Rahmenlehrplan sowie das schulinterne Curriculum spielen dabei eine gewichtige Rolle in der Unterrichtsplanung. Um die Anforderungen für jede Klasse eines Schuljahres umzusetzen, erstellen Lehrende Stoffverteilungspläne. Dies erfordert eine

Menge manuelle Arbeit: Meist ermittelt das Lehrpersonal die Anzahl der verfügbaren Schulstunden einer Klasse durch klassisches Zählen, wobei Feiertage, Feriendaten sowie schulinterne Veranstaltungen beachtet werden. Schlussendlich kann die ermittelte Gesamtzahl den einzelnen Themengebieten zugeordnet werden.

Um den Aufwand zu reduzieren, verwenden Lehrende ihre Stoffverteilungspläne meist wieder. Nichtsdestotrotz sorgen individuelle Klassenprofile und unvorsehbare Ereignisse dafür, dass auch während des Schuljahres Änderungen vorgenommen werden müssen.

Diese Arbeit zeigt auf, dass jeder Lehrende für die Erstellung eines groben Schuljahresplans einen eigenen Prozess implementiert. Meist werden dabei Tabellenstrukturen in Microsoft Word oder, ganz klassisch, Papier und Stift verwendet - auch wenn eine Lernmanagementsoftware an der Schule vorhanden sein sollte. Ein Grund dafür könnte sein, dass die Untersuchung dieser existierenden Applikationen zeigt, dass kein System eine dedizierte Unterstützung für solche Anforderungen bietet.

Aufgrund dessen wird in dieser Arbeit ein Konzept für ein webbasiertes Planungswerkzeug vorgeschlagen, das die manuelle Arbeit minimieren, dynamische Anpassungen während des Schuljahres ermöglichen und dabei stets einen Überblick über den aktuellen Stand bieten soll. Das Vorhandensein der Informationen in der Cloud könnte außerdem das Bilden von weiteren Synergien ermöglichen.

Eine Implementierung des Konzepts wurde in die Schul-Cloud-Plattform integriert. Eine qualitative Nutzerstudie konnte zeigen, dass Lehrende der Lösung nicht nur eine gute Benutzerfreundlichkeit zusprechen, sondern auch, dass das generelle Konzept hohes Potenzial bietet.

ACKNOWLEDGMENTS

Wow! What a journey it has been - not only the Master's thesis but this whole stage of my life. This piece of work will conclude my study time, and this is something I can hardly realize right now. No matter what, this time of my life will never be forgotten. Too many great memories and friendships are connected with it.

In any way, there are many people that made it possible for me to be where I am. First of all, I want to thank Jan for supervision and being passionate about the topic. You have been positive about the work at all times, and every meeting with you left me with a good feeling about the project. Thank you for this!

In that regard, the whole Schul-Cloud team is to thank. Everyone was ready to help at all times. Thanks folks!

In general, I want to thank the HPI. The study conditions that are provided to all of us here are excellent, and I feel incredibly privileged and grateful for having been able to study here.

Next, I want to thank my proofreaders Stephan, Marv, Felix, Thomas, and Christian. I know that without your valuable feedback this thesis could not have been what it is now. I feel deeply grateful. Have a drink on me!

Thanks to you, Lena, for the support and being by my side. I know the last months have not been easy on you. I highly appreciate that you never complained and always showed understanding.

Last but not least, I want to thank my family - especially my mum and my grandparents. I know that all this, would not have been possible without you. Your unconditional support has always accompanied me in my whole life. I cannot express in words, how grateful and privileged I feel for this.

Well then... Onto the next adventures!

CONTENTS

1	INTRODUCTION	1
1.1	The Schul-Cloud	2
1.2	School-Internal-Curriculum	4
1.3	Research Questions	6
2	RELATED WORK	9
2.1	PLATON	9
2.2	School Management Solutions	10
2.3	Schul-Cloud Architecture	12
3	REQUIREMENTS	17
3.1	User Research	17
3.2	Deducted Insights	20
3.3	Software-based Planner Proposal	21
3.3.1	Content Focus	21
3.3.2	Technical Focus	25
4	CONCEPTS	27
4.1	Modularization	27
4.2	Components	28
4.3	Styling	30
5	IMPLEMENTATION	39
5.1	Planner Library	39
5.1.1	Implementation Philosophy	39
5.1.2	Components	43
5.2	Integration into Schul-Cloud	58
5.2.1	Changes to Frontend Pipeline	58
5.2.2	Schul-Cloud-Client	62
5.2.3	Schul-Cloud-Server	66
6	EVALUATION	71
7	FUTURE WORK AND CONCLUSION	77
A	APPENDIX	83
A.1	Type Interfaces	83
A.2	Evaluation Test Form	89
	BIBLIOGRAPHY	91

ACRONYMS

API	Application Programming Interface
JS	JavaScript
REST	Representational State Transfer
HTML	Hypertext Markup Language
SCSS	Sassy Cascading Style Sheets
SASS	Syntactically Awesome Style Sheets
CSS	Cascading Style Sheets
TTI	Time To Interactive
DOM	Document Object Model
SQL	Structured Query Language
SCHIC	School-Internal-Curriculum
CRUD	create, read, update and delete

INTRODUCTION

The unstoppable digitalization of the world comes with significant challenges for all parts of society. Education is one of them. On the one hand, due to the increasing automatization of simple work, the amount of less skilled labor shrinks. On the other hand, the number of positions that require highly qualified employees grows.

Already, the lack of a qualified workforce has a significant effect on the economic performance in Germany. The Cologne Institute for Economic Research estimates that currently 440,000 positions cannot be filled, which results in slower economic growth of up to 0.9% [6]. According to the research institute Prognos, this trend will continue to rise to three million unfilled positions by the year 2030 [16].

The education system could cushion these trends. Primary and secondary schools have to foster the student's competencies and skills that are the very foundation for a successful work life later on. Improving the quality and adjusting the way knowledge is taught based on new scientific findings is, therefore, a crucial task.

As shown in an article from the Bertelsmann-Stiftung, there is empirical evidence that digital media is, in fact, conducive to learning results [11]. Across all types of schools, teachers believe that the digitalization holds excellent opportunities to improve the quality of their teachings [3].

However, most educational institutions in Germany fell behind the present digital reality. While current generations grow up in a digitalized world being digital natives, many schools lack fundamental requirements. According to the Bertelsmann Bildungsmonitor from 2017, only 37% of the teachers were happy with the quality of the school's WiFi. Moreover, 58% complained about missing IT support and 65 % wished for further training [18].

It is hardly surprising that the 2013 International Computer and Information Literacy Study showed that only one-third of German teachers use computers at least once per week, in contrast to 61.5% in the international comparison [5, p. 223]. These are worrisome numbers that show that Germany lags behind other nations.

To tackle these problems, the department for education and science enacted the Digitalpakt in October 2016 with a total volume of five billion euros. It is supposed to start in 2018 and strives to make the schools ready for a digitalized world. The funds are meant to build up and improve the digital infrastructure, provide broadband inter-

net connections for every school and educate the teaching staff. The Schul-Cloud project's origin lies in this context.

1.1 THE SCHUL-CLOUD

The Schul-Cloud is a Germany-wide pilot project supported by the federal ministry of education and research, the MINT-EC association¹ and the Hasso-Plattner-Institute. It began in 2016 with the goal to offer schools a single interface for web-based educational content and applications for students, teaching staff and parents. The pilot project started nationwide with 25 schools and was extended to the number of about 300 MINT-EC partner schools in May 2018.

While the Schul-Cloud is aimed towards providing a solution for the whole German school system, it is not the first education cloud on the market. Due to the German education federalism, school-related issues are a matter of the individual states. Therefore, some states already self-developed or invested in other education cloud system.

GOALS The platform was created in order to fulfill a specific set of goals that are described in the following:

- *Cross-school and cross-state synergies* - Being a Germany-wide platform, cross-school and cross-state synergies can be created. Instead of developing multiple systems with a similar feature set, the development of one platform could decrease the overall expenses. Moreover, all features that are developed are available for every school without extra costs.
- *Data and Privacy* - The compliance with the strict data and privacy law is a prominent focus point in the development. It is supposed to be ensured by incorporating "Privacy by Design" and keeping close consultation with data protection commissioners.
- *Digital Education* - As formerly mentioned, schools fell behind the rapidly evolving digital reality. Due to this constant change, it is even more important that schools be able to provide digital education. The Schul-Cloud wants to enable schools and teachers to catch up in this area by offering the possibility to incorporate digital media in their respective subjects easily.
- *Quality of learning material* - The digitalization led to the development of new, innovative learning materials. Being able to make use of such content, can improve the education quality signifi-

¹ The Verein mathematisch-naturwissenschaftlicher Excellence-Center an Schulen e. V., short MINT-EC is a nonprofit association with the goal to promote education in mathematics and natural sciences in schools.

cantly. In order to do so, the platform integrates an education store for high-quality learning materials which are easy to find and include into lessons.

Third-party developers of professional teaching and learning content are promoted to offer their resources, as the Schul-Cloud acts as a single interface for a multitude of schools. Also, the teaching staff is encouraged to share its material with fellow teachers from other institutions.

- *Off-campus learning* - Students nowadays - being digital natives - grow up with smartphones and mobile internet. According to BITKOM, 67% of students of age 10 to 11 owned a smartphone in 2017 [2]. The number grows to 88% for 12 to 13 years old. The Schul-Cloud platform is able to provide students with learning materials and information 24/7 - even off campus at home, no matter the end device.
- *Cost savings* - Schools are supposed to profit by not having to build up their own server infrastructure. Thus, they can save funds from their sparse monetary budget for other areas. Moreover, the need for maintenance operations, like security updates can be dropped - a task that is nowadays often fulfilled by dedicated teachers on top of their regular work. Due to the Schul-Cloud being a web application, it solely requires a working and fast digital infrastructure. A significant number of schools lack this fundamental requirement. However, this is about to change. The Digitalpakt with its fund of five billion euros is explicitly destined to help the schools with improving their digital infrastructure in the near future. The School-Cloud, thus, is designed for this future, taking a built-up infrastructure as a given.
- *Administration* - Teachers can have an easy overview of their courses. That includes providing digital learning materials for their students, homework or group work that can be managed via the Schul-Cloud.

USE CASES While the previous paragraph describes the general goals of the platform, this paragraph aims to explain the destined main use cases of the Schul-Cloud [14].

- *File sharing* - Teachers and students should be enabled to share documents and digital information bidirectional. Hence, teachers should be able to provide files to students and students should be able to upload material for teachers. That can include learning materials or submitting homework.
- *Collaboration* - By providing groups, collective work on assignments and projects is offered and can be easily managed.

- *Differentiation* - Teachers are encouraged to offer individual learning content. In that way, the student's different knowledge levels can be taken into account when providing tasks. The high-performing and interested students could be given extra-material, while others more struggling with the assignments could receive additional help and explanations.

As the goals and use cases show, the Schul-Cloud focused so far mainly on improving and extending the interaction possibilities between teachers and students. Other use cases like school administration related tasks are not addressed yet. However, these kinds of tasks are privileged to profit from digital tooling as well. While teachers are employed to teach students, the reality proves to provide a large number of tasks that are not directly connected to this activity. Tasks like lesson preparation, consultation hours or field days take a significant time of teachers' daily work. According to Hardwig and Mußmann, the actual lecture time takes only about 30% of the total work time for teachers at high schools [10, p. 99]. Moreover, the studies suggest that the share of non-lecture related workload has increased in recent years.

1.2 SCHOOL-INTERNAL-CURRICULUM

An example of additional overhead that teachers have to deal with is the School-Internal-Curriculum (**SCHIC**). It is an obligatory curriculum that had to be implemented by teachers and schools in Berlin and Brandenburg for their respective classes from grade one to ten. The State Institute for School and Media Berlin/Brandenburg² developed the concept on behalf of the Senate administration of Berlin and the ministry of education in Brandenburg. It was introduced in 2016 and aims to play an essential part in the quality development and quality assurance in schools. The **SCHIC** wants to ensure the qualification of students in relevant skills by bringing all stakeholders of the regular school life together (which also includes external cooperation partners). Moreover, it turns its attention to improving interdisciplinary learning by establishing a skill-based learning concept and integrating media education for all subjects.

The curriculum's structure is made up of three parts [7]:

- *Part A* - School Specific Definition
This part aims to include the school's individual characteristics, which includes the respective profile, external partners and topical focus points. It is meant to describe the school in its contextual environment and create a frame for the following parts.

² <https://lisum.berlin-brandenburg.de/lisum/>, Accessed: 06.12.2018

- *Part B - Interdisciplinary Definitions*
Regarding the emphasis on interdisciplinary learning, media and language education, as well as topics like intercultural education, democracy cultivation or preventing violence have to be covered across all subjects. Furthermore, collecting and documenting this information should support the creation of synergies between different subjects.
- *Part C - Subject Related Definitions*
Finally, the individual subject concerning requirements and plans have to be gathered. They should include references to part A and B, in order to make sure that the subject references the individual school profile and contributes to the overall interdisciplinary learning goals.

It is developed by including the whole teaching staff. However, the approach on creating the individual parts can be different: Part A/B can be developed based on Part C or vice versa. In general, the development of the curriculum can be seen as a year-long process that has to be followed and monitored over time. It starts by creating a baseline of the current situation. Afterward, responsibilities have to be assigned, and an action plan has to be created. Each subject department creates a **SCHIC** individually, which again consists of definitions on the topic level of a subject as can be seen in [Figure 13](#). This process needs multiple meetings and agreements on different levels.

In average, the final document, consisting of all the content from the subject departments and general information, is made up of several hundreds of pages.

Research in regard to this thesis showed that the implementation of the **SCHIC** suffered from many problems. First, the creation of such a curriculum is a very complicated and individual procedure. Thus, the initial commitment and effort were tremendous. The work had to be done in addition to regular work of the teaching staff. Moreover, the justification for this extra effort was not completely communicated/accepted/understood.

Schools had much freedom for the implementation, which led to frustration because it was unclear what was required and how the process could be started. Missing guideline material, in the beginning, resulted in additional effort on top.

Due to missing guidelines, the curriculum looks different in every school and teachers started to build up their own ecosystem to share guides and templates⁴.

Interviews with officials showed that it is expected that the **SCHIC** is locked away after its creation, even though it is wished that the plans

³ This example shows the topic "Me and my school" for a class of grade one and two in the subject German.

⁴ E.g., http://www.mieriesuperklasse.de/neuer_rahmenlehrplan_berlin/ and <http://schulinternes-curriculum.de/>, Accessed: 13.11.2018

Jahrgangsstufe: J1/2		Fach: Deutsch		Unterrichtsvorhaben: Ich und meine Schule		zeitlicher Rahmen: ca. 15 Schulstunden	
<div>Aspekte zur Sprachbildung</div> <ul style="list-style-type: none">Wortschatz Klassenzimmer, SchulhofRitualisiertes Sprechen (Begrüßung, Datum, Stundenplan, ...)		<div>B</div>	<div>Methoden</div> <ul style="list-style-type: none">Gruppen-, Einzel-, Partnerarbeit, Plenum		<div>Arbeitstechniken/fachimmanent</div> <ul style="list-style-type: none">ausschneiden, klebenausmalen, gestalten, schreiben		<div>A</div>
<div>Basiscurriculum Medienbildung</div> <ul style="list-style-type: none">Ich und meine Schule (Buch gestalten)			<div>Jahrgangsstufe 1</div> <div>Sprechen und Zuhören</div> <div>2.1</div> <div>A: einzelne Informationen mitteilen</div> <div>B: über Dinge aus ihrer Lebenswelt erzählen und informieren</div> <div>C: Informationen für ihre Erzählung auswählen und nutzen</div> <div>A: deutlich sprechen</div> <div>2.2</div> <div>A: Gesprächskonventionen in vertrauten Situationen berücksichtigen</div> <div>B: eigene Beiträge zu einem Thema einbringen</div>		<div>Jahrgangsstufe 2</div> <div>Sprechen und Zuhören</div> <div>A: degt.</div> <div>B: degt.</div> <div>C: Informationen für ihre Erzählung auswählen und nutzen</div> <div>A: degt.</div> <div>B: beim Sprechen auf Lautstärke u. Tempo achten</div> <div>C: vorgegebene Redemittel für ihren Vortrag nutzen</div>		
<div>Übergreifende Themen</div> <ul style="list-style-type: none">Gemeinschaft und ZusammenhaltMobilitätsbildungErhaltung der GesundheitIch-IdentitätRegeln kennenlernen und übenSoziales Lernen: Paternsystem			<div>Schreiben</div> <div>2.5</div> <div>A: Wörter deutlich sprechen</div> <div>B: Laute benennen</div> <div>2.6</div> <div>A: Wörter zu einem vorgegebenen Inhalt nennen</div> <div>B: Wörter u. kurze Sätze zu vorgegebenen Inhalt aufschreiben</div>		<div>Thema/Inhalte/Kompetenzentwicklung</div> <ul style="list-style-type: none">SchulralieIch und meine Schule (Buch)KennenlernspieleÜber sich und Familie erzählenInterviews mit Personen der Schule durchführenBeobachtungsaufgaben		
<div>Externes Kooperationsangebot/ außerschulische Lernorte</div> <ul style="list-style-type: none">—			<div>Fächerverbindende Schwerpunkte</div> <div>SU: Themenfeld Kind</div> <div>BK: Zeichnungen (Wachsmaler, Buntstifte)</div> <div>Sport: Bewegungsralie durchs Schulgelände</div> <div>Musik: Begrüßungslieder, laut/leise Themen</div>		<div>Leistungsdokumentation und -bewertung</div> <ul style="list-style-type: none">„Ich und meine Schule“ (Buch erstellen)		
					<div>Ganztäg</div> <ul style="list-style-type: none">Lern- und Lebensumfeld Schule und GanztägSpieleRituale und Regeln anbahnenTagessabläufe (Rhythmisierung) kennenlernen und lebensich orientieren		
					<div>Schulkultur</div> <ul style="list-style-type: none">Bildung einer KlassengemeinschaftIdentifikation mit der Schulgemeinschaft durch Feste (Einschulungsfeier)		

Figure 1: Exemplary curriculum implementation of part C

"should live." However, this is not realistic, since the curriculums are not checked by officials.

1.3 RESEARCH QUESTIONS

Section 1.1 described that the Schul-Cloud does not address use cases concerning the organizational overhead yet. However, reducing this workload could enable teaching staff to use the freed up resources to focus on improving the quality of the teachings instead. Moreover, useful additional functionality could improve the acceptance and adoption of the Schul-Cloud by deepening the integration of it in the daily teacher's and school's work.

As described in Section 1.2 the school internal curriculum posed a significant organizational challenge for teachers. It could be investigated, how the Schul-Cloud could help with the implementation of such a curriculum. However, even though other states have similar initiatives, the exact requirements are restricted to schools in Berlin and Brandenburg and the Schul-Cloud, being a nationwide project, should target use cases concerning all schools. Furthermore, the general implementation of the SCHIC is finished. Taking into account the background story, it can be expected that teachers are not motivated to touch on the topic again.

Nonetheless, there seems to be much potential in exploring a more general idea of supporting teachers in the planning of their classes' curriculums, which leads to research question Q1. In that regard, this thesis will investigate, how teachers organize themselves nowadays.

Moreover, it will be researched, if an extension of the Schul-Cloud's feature set could incorporate software-based support and how that could be implemented.

Section 1.1 explained that the Schul-Cloud is a pilot project. It is just available for a small subset of Germany's schools yet. Solely integrating planning functionality for teachers into the Schul-Cloud would, therefore, limit the pool of users. Additionally, other state-specific or commercial cloud projects try to solve similar problems and could potentially also profit from such implementation. Thus, this thesis will explore in question Q2, how functionality can be seamlessly integrated into the Schul-Cloud, while also being integrable into other applications or being able to stand on its own.

Q1: *How can we support teachers' education planning in the Schul-Cloud?*

Q2: *How can the functionality be optimized on graphical adaptability?*

The thesis will begin with an exploration of related work in Chapter 2. Chapter 3 describes the insights gathered by the user research and the derived requirements for a concept proposal. Following is Chapter 4 which will explain the technical concepts developed to realize adaptable functionality. Afterward, the implementation of the planner library is documented. Moreover, Chapter 5 includes the integration process into the Schul-Cloud. Chapter 6 presents the results of a user study that was performed to evaluate the developed planning tool. Finally, Chapter 7 gives an outlook and recommendations on the next steps of the research.

RELATED WORK

In order to investigate how the Schul-Cloud could help with the planning of terms and how functionality could be integrated, this section is meant to examine the current state. By exploring related work that is concerned with similar fields of interest, areas worth investigating in the following sections are to be identified.

First, PLATON a web-based planning tool is introduced. Furthermore, other popular classroom management solutions like iServ or Google classrooms are examined. Lastly, the Schul-Cloud's software architecture is analyzed and presented.

2.1 PLATON

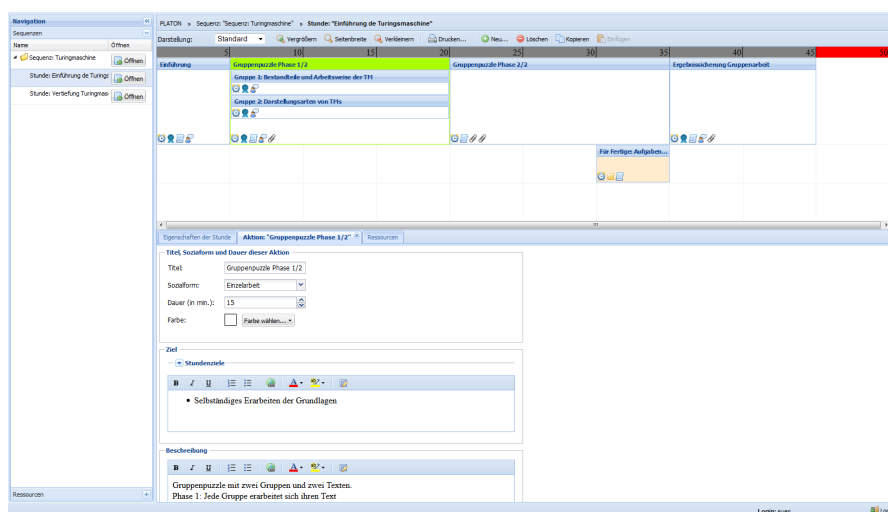


Figure 2: The PLATON lesson planning platform

PLATON is a lesson planning platform developed and described by Strickroth in his dissertation [19]. In his work, he underlines the importance of the development of a lesson plan as a prerequisite for effective education, especially for prospective teachers. Due to it being a complex and demanding task, the preparation is often realized with a prepared lesson draft. It always contains the sequence of a lesson, but most often also the reasoning and context of decisions.

The central part of the dissertation investigates whether a computer-based system can support the teaching staff in the preparations of these drafts.

Although there are existing solutions, Strickroth describes that many of them do not even meet the basic requirements. For example, despite that adaptive feedback was identified as a particularly important feature to help with the complexity of lesson planning, no existing system implements it.

In order to fill this gap, a web-based planning tool named PLATON was developed, which is shown in [Figure 2](#). The main features are a graphical, time-based visualization approach in combination with analysis capabilities and automatic feedback.

An empirical two-step evaluation of about 100 participants showed that the approach offers good usability. The software is suitable for the planning of lessons. The time-based approach reminds of otherwise unthought aspects and automatically generated hints provide improvement opportunities.

However, the valuation of the usefulness of the tool was decreasing for more experienced teaching staff. Strickroth explains that PLATON could potentially only offer additional value for specific periods of the teaching education.

2.2 SCHOOL MANAGEMENT SOLUTIONS

Besides the Schul-Cloud, there are other popular software-based school and classroom management solutions, that are used in Germany and other countries of the world. This section will explore their feature sets to identify whether these products incorporate school year planning support.

GOOGLE CLASSROOM Google Classroom is an integrated learning platform that uses and connects known Google services with the school context. According to a nationally-representative survey in the US from 2017, Google GSuite and Google Classroom are used most frequently by 68% of respondents. That makes Google Classroom the most popular classroom management solution in the US [8, p. 4].

The feature set allows two-way sharing of documents, presentations, and spreadsheets via Google Drive. Teachers can post announcements into an information stream that can be commented on by students as displayed in [Figure 3](#). Moreover, Google Mail is integrated what enables teachers to communicate with parents to inform them about events or send them summaries about current assignments. Additionally, appointments and deadlines can be managed with Google Calendar. Also, assignments can be graded via the platform and returned to the student for further revision.

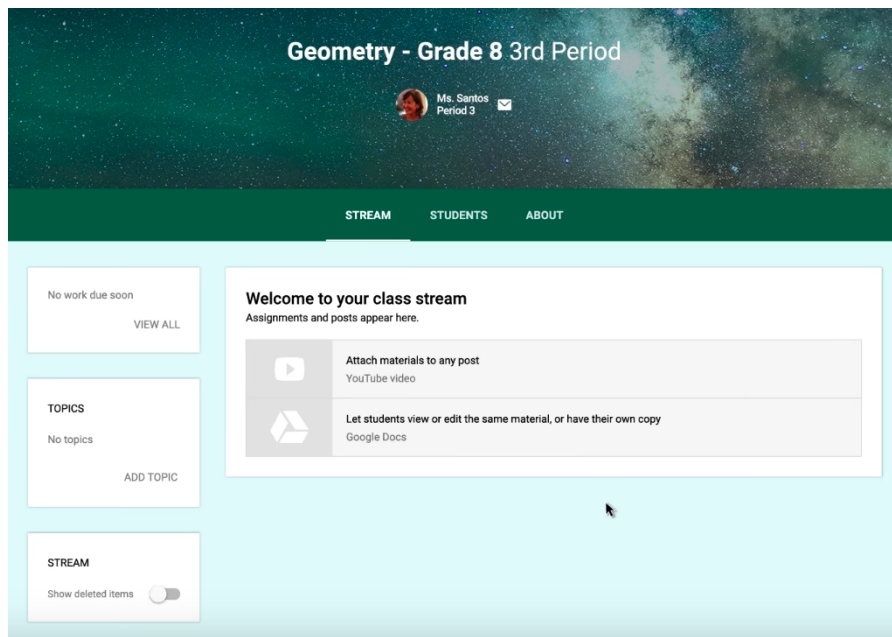


Figure 3: Google Classroom dashboard

Even though Google introduced the function to copy and reuse classwork in August 2018, dedicated support for school year planning is not given.

APPLE CLASSROOM Apple Classroom is an iPad application for teachers that supports setting up, controlling and monitoring iPads in school classes. The application gives teachers control over the student's iPads. They can force open the same app on every device, lock features, mute devices or navigate students to specific content. Moreover, documents (like worksheets, text, presentations) can be shared in both ways. However, there is no support for school management related tasks.

ISERV iServ is a dedicated, commercial school server system made in Germany that enables building up a school network. It is especially adapted in Niedersachsen but provides its service to a total of about 2000 schools in Germany.

The feature set offers more than just a classroom management solution. The server includes a web portal for teachers and students with an adjustable set of modules. Depending on the configuration there is support for functionality as file sharing, chats, bulletin boards, calendars, facility management or a school-internal mail service.

Due to the possibility to upload material, create classes and document events in a calendar, it is technically possible to use it for school year planning. Nonetheless, the functionality is not dedicated for this use case.

ITSLEARNING The learning and community platform itslearning is a commercial cloud platform. It is designed for lower and higher education facilities and especially popular in its founding country Norway. However, among others also schools and universities from the US, Sweden, and Germany use the software. In Bremen, for example, it was decided that till 2018 all 170 schools should be able to use itslearning after an earlier pilot project proved to be successful [12]. As other learning platforms, it offers a system to communicate with course participants, enables lecturers to create assignments and share material, define a course schedule or grading submissions. As for iServ, it is technically possible to use itslearning for school year planning. However, there is no dedicated support. Thus, the research showed that users use workarounds to achieve such behavior. For example by creating regular courses as a master course and by copying material from it to instances of such courses¹.

2.3 SCHUL-CLOUD ARCHITECTURE

The Schul-Cloud application resembles a classical client-server architecture consisting of two notable entities: the Schul-Cloud client serving the web frontend and the Schul-Cloud server as the backend offering an Application Programming Interface (API) for the former one [14, p. 23]. Both parts have their respective repository². Following a micro-service architecture³, the server itself is subdivided in multiple independent micro-services following the feathers.js web framework. It also interacts with other external services as can be seen in Figure 4. These external micro-services resemble a notification, content and calendar service.

TECHNICAL REQUIREMENTS When the Schul-Cloud's architecture was designed, some particular goals were kept in mind: First of all, despite the future growth with possibly hundreds of schools, the app's architecture should include simplicity by design. The application is a university project: It is used for research and teaching. Thus, new functionality is contributed by many developers and from a wide variety of sources including seminars, bachelor's or master's thesis, researchers, part-time working students or full-time employees. In order to enable these different developers to work efficiently despite having different levels of familiarity with the system and possessing varying amounts of development experience, the architecture needs to offer simple entry points. No matter the location, where function-

¹ <https://www.youtube.com/watch?v=9A046-Kru8E>, Accessed: 18.11.2018

² <https://github.com/schul-cloud/schulcloud-client> and <https://github.com/schul-cloud/schulcloud-server>

³ <https://martinfowler.com/articles/microservices.html>, Accessed: 06.12.2018

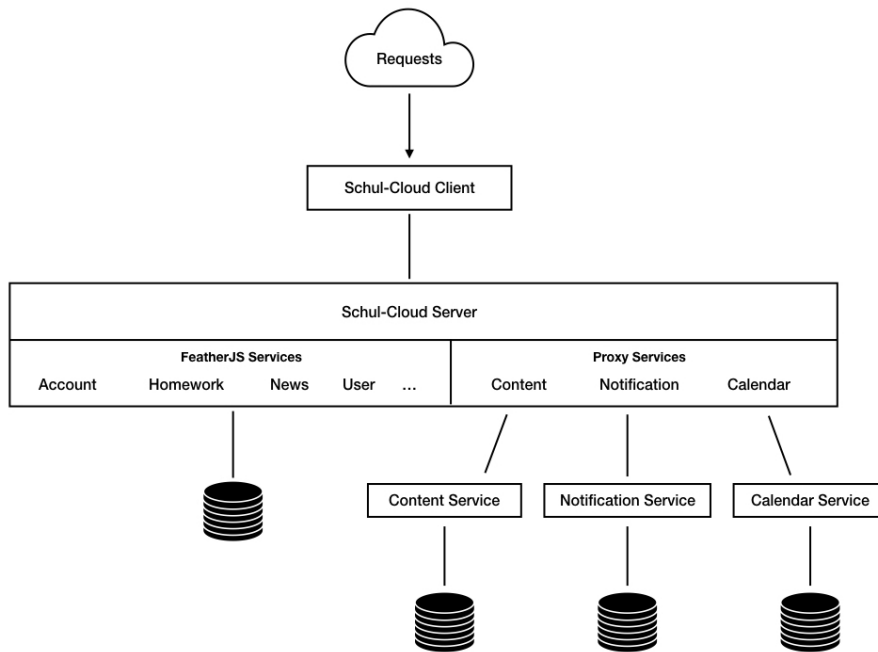


Figure 4: Overview of Schul-Cloud architecture

ality should be adjusted or added, the right place in the source code needs to be easy to find.

Another requirement was a horizontally scalable architecture. That is because new schools and, therefore, many users will join the project step by step. Hence, the architecture needs to be robust enough to offer potential scaling when required.

SCHUL-CLOUD CLIENT The client is responsible for rendering and delivering the content of the web application to the user's browser. A server written in node.js using the web framework `express`⁴ handles the incoming user requests. Depending on the requested path, route-specific controllers handle the further work as displayed in Figure 5. Based on the respective request and potential query parameters, they process the sent data and request dynamic data from the backend. As for that, they query the Schul-Cloud server via Representational State Transfer (REST) requests. The microservices' responds are then arranged and, finally, passed to the `handlebars.js`⁵ templating engine. It injects dynamic content into pre-defined templates based on the given data. It can render not only simple strings but takes care of generating Hypertext Markup Language (HTML) elements or rendering messages based on conditions. Finally, a response with the generated markup is sent to answer the browser's request.

Any dynamic scripting and styling are done by statically including script and link tags in the respective `handlebar.js` templates. Moreover,

⁴ <https://github.com/expressjs/express>

⁵ <https://github.com/wycats/handlebars.js/>

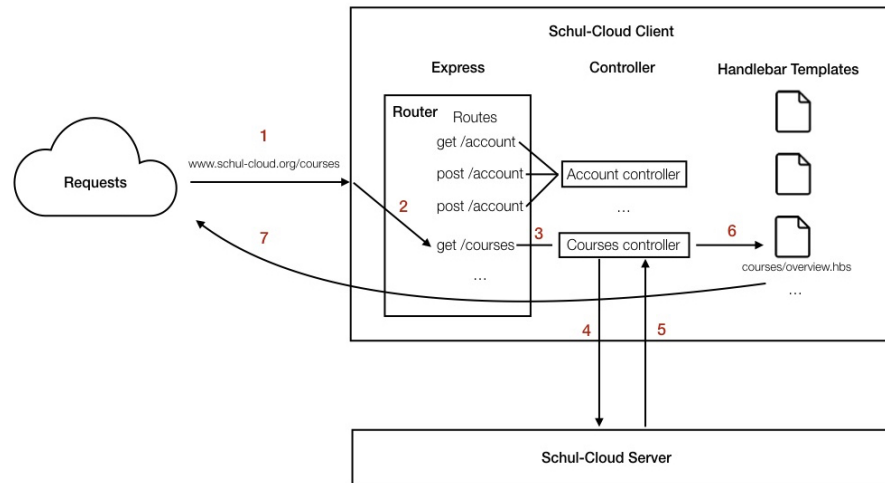


Figure 5: Schul-Cloud client architecture

since the Schul-Cloud cooperates with other state-specific initiatives like n-21 in Niedersachsen [4], it has to provide a dynamic theming approach.

Thus, the generation of the assets requires a build step. Hence, a pipeline was created with Gulp⁶. Besides compiling Sassy Cascading Style Sheets (SCSS) files based on the defined theme, it orchestrates all other steps of the build process:

- *Transpiling*

Transpiling has become a default build step for most web development teams just very recently. It arose with the release of Babel⁷. Before that, new JavaScript language features could often not be used by developers, because the different target browsers versions (especially older ones, that still needed support due to high adoption rates) would not handle the most recent syntax. Babel changed this by allowing developers to write next-generation JavaScript code without having to care for the browser support. That is done by defining the target browsers and letting Babel take care of transforming any unsupported new syntax into supported code.

- *Minification*

By default source code is meant to be readable by humans not machines. Therefore, development teams pose a high value on readable, well-documented code that can be easily understood. That includes meaningful variable names and code comments to support the understanding of the reader. When sending JavaScript files to the client, however, readability is not relevant. The only entity that is meant to read the source code is the browser. Moreover, every additional letter or space that needs

⁶ <https://gulpjs.com/>

⁷ <https://babeljs.io/>

to be sent to the client slows down the parsing and, hence, the Time To Interactive (TTI) for the user. Minification in the build step counteracts this problem by automatically stripping white spaces, reducing the length of variable names and removing comments from the source code. In that way, the code can stay readable for the developers, while being as fast as possible for the production setup.

The analysis shows that the Schul-Cloud is not a single page application: Rendering and UI logic is implemented on the server side. Nonetheless, there are single components written with Vue and React. In most cases, however, jQuery is used for scripting purposes.

SCHUL-CLOUD SERVER The Schul-Cloud server is implemented as a Node.js application with feathers.js⁸ as underlying micro-service web framework to organize the REST API layer. Following a micro-service architecture, the server has a wide range of different services (e.g., account, authentication, calendar, notification, role service) that are either implemented as local feathers.js services or are self-contained ones. Figure 4 visualizes that architecture.

Most of the services are connected to the no-SQL database MongoDB via mongoose⁹. The library offers a solution for schema-based modeling of application data while ensuring validation and providing automatic query building. It is integrated into feathers.js with the feathers-mongoose¹⁰ database adapter.

This architecture allows for the uncomplicated development of new services that offer simple create, read, update and delete (CRUD) operations on their resources.

⁸ <https://feathersjs.com>

⁹ <https://mongoosejs.com/>, Accessed: 06.12.2018

¹⁰ <https://github.com/feathersjs-ecosystem/feathers-mongoose>

REQUIREMENTS

This chapter will describe the set of requirements needed to support educators with their creation of learning matter distribution plans. At first, the content and insights gathered from the qualitative user research are presented. Using the condensed findings from these interviews, as well as the knowledge from the related work described in [Chapter 2](#), the deducted requirements are stated. Finally, a software system supporting the preparation and planning of school years is proposed.

3.1 USER RESEARCH

To collect the requirements for a software-based tool, user research was performed. By interviewing several teachers from different federal states, empathy for processes, expectations, and problems was developed. Condensed interview information and insights are described in the following.

INTERVIEWS A first interview was conducted with a high school teacher from Niedersachsen. As of the interview date, the interviewee lectures classes in sports and music from grade 6 to 11. The assignment of classes happens before the school year starts, based on the number of working hours a teacher has. This number differs depending on the state - in Niedersachsen, for example, it is 23.5 hours per week. The other 16.5 hours are meant to be spent on lecture preparation and wrap-up. After getting the list of assigned classes, she will plan her school year ahead.

As for that, the core curriculum by the state, as well as the school internal curriculum is taken into account. In her example, she is teaching in a school stressing musical education. Hence, some music classes have to include one mandatory chorus lesson per week, which is a requirement given by the school internal curriculum.

Taking this information into account, the general topics that have to be covered for a class, as well as the number of available hours, are used when creating the school year plan with an assignment of time periods per general topic. The number of available hours is determined by going through the calendar and summing up the available lessons. Due to her school using iServ - a platform described in [Section 2.2](#) - holidays, public holidays and long-term events (like a project

week/day) are often known before the school year starts.

Nonetheless, not everything is clear in advance and "surprises happen quite often." This does not mean that a class will not show up completely unexpected, but that events like the teachers' conference are only determined during the school year. In most cases, she is notified via iServ if there are unexpected unavailabilities concerning her classes. Despite the matter, depending on the subject she plans the topics with buffer times to make sure to stay on plan. However, if unplanned events occur, she mostly does not change the length of the time period for a topic but shortens the content. That could mean, for example, less exercising time for students. Moreover, it is a regular case that topics are adjusted based on the classes' individual understanding. Therefore, even though two classes of same grade and subject are taught in one school year, their plan can differ.

In general, the use of iServ is valued highly. Most school-related issues can be managed via the platform. That includes substitution teacher plans, reservation of rooms, a mail service, file sharing with other teachers or students, calendars for everyone, students or teachers, an address book, a printing system, and an exam schedule. Students are obliged to check the platform on a daily basis, which is why announcements and messages are binding for everyone.

Stoffverteilungsplan Oper „Hänsel und Gretel“

Unterrichtsstunde	Thema der Unterrichtsreihe	Thema der Stunde	Besonderheiten
1-18.10.	Notenlehre	- Testausgabe, Vergleich, - - Komposition alternatives Ende „Wattwurm“ - Kriteriengeleitete Bewertung	
2-21.10.		Chorstunde	
3-25.10.		Einführung in die Thematik	2
4-28.10.		Chorstunde, Vertiefung Lied „Hänsel und Gretel“	1
5-01.11.		Opernbegriff (S. 82, Nr. 1, 2)	2
6-04.11.		Steckbriefe	1
7-08.11.		„Hexenritt“ Standbilder	Examens-FUB? 1
8-11.11.		Chorstunde	1
9-15.11.		Ouvertüre. Szen.	2
10-18.11.			Entfall
11-22.11.		KA-Vorb., szenisches Spiel-differenziert	2
12-25.11.			KA 1
13-29.11.		Chorstunde, Waldszene	2
14-02.12.			WB
15-06.12.		Waldszene, Abendsegen	2
16-07.12.			PU 1
17-09.12.	Chorstunden		Aula reservieren?
18-13.12.			
19-16.12.			Generalprobe für Konzert am 19.12.

Figure 6: Exemplary learning matter distribution plan

However, iServ is not used for the learning matter distribution or detailed lesson planning (not taking into account the calendar, which is used to determine the availability of classes). As for that, she relies on word documents. Generally, there is one document per topic-

class-school year combination. It contains everything belonging to this topic. Thus, it is made up of the learning matter plan arranged as a table as shown in Figure 6, more or less detailed lesson drafts and related learning material like worksheets. However, it is to note that the lesson drafts are generally not as detailed as they had to be during the apprenticeship. In this time period, these plans have to be created as minute-wise sequence plans. In the regular teacher's life, this is, however, not realistic.

In case a new school year needs to be planned, the process for existing class-subject combinations is to just manually copy the folder from a most recent school year and paste it to the new school year. Consequently, only the dates of the rough schedule have to be adjusted.

While this is her process, she assumes that other teachers have similar habits, due to everyone having to complete the apprenticeship that teaches this approach. Nonetheless, the interviewee reckons that older teachers organize themselves less digitally. Often they have their worksheets still in physical folders.

Regarding the sharing of learning material between teachers, she describes that school-internal sharing is common. Especially, detailed lesson plans and worksheets are shared. Either the material is sent via school-internal mail offered by iServ, or, what can be the case with older teachers, a worksheet is printed and then handed to her in physical form. In that case, she will scan the document again and add it to her digital collection. Inter-school-sharing is, however, something that does not happen.

Another point brought up by the interviewee is that there is much uncertainty among the whole teaching staff regarding data and privacy, as well as licencing questions. Teachers are unsure about what they are allowed to save, share and show and what not. In case of doubt, teachers generally decide to not use the material, in order to not be at risk. This problem is perceived as a very pressing issue.

Two other interviews with teachers working in Niedersachsen, Berlin, and Brandenburg mostly confirmed the mentioned findings. Interviewees reported that the planning of the school year is often done in advance. However, one teacher reported that the teachers he works with, do the planning usually on paper with a calendar at hand. After the creation, teachers generally stick with their plan but make adjustments depending on the skill level of their respective classes. He stated that, typically, these plans are reused every year, only incorporating small changes. Moreover, it was mentioned that while some teachers plan everything ahead in very small details, other teachers have a more fluid and dynamic planning style.

The problem with uncertain events occurring was confirmed. It was stated that "one often has less time for a topic, but does not know about it" in advance. Another point brought up is the planning of

tests and exams. Due to regulations preventing that too many tests are written on the same day, scheduling conflicts occur, because one "does not know what the colleagues do." Another problem that was brought up was that substitution teachers often do not know about the classes they have to lecture. They get assigned in the morning and have no knowledge about the state of the currently covered topic. Therefore, they have to lecture something very different or let the students simply work on their own.

3.2 DEDUCTED INSIGHTS

Summarizing the findings from the research, the following central insights could be extracted:

- Teachers plan their school year roughly in the beginning. However, the granularity of these plans differs individually.
- Even if schools use platforms like iServ - the school year planning is still solely self-organized. That means that it is either done on paper or saved on personal computers. Information from the school management solution, necessary for the plan, are transferred manually.
- Early in their work life, teachers build up a personal learning matter collection that is based on their teaching preferences and style. Even though teachers iterate on their material, once there, it is the starting point for every year's planning. Hence, the amount of work decreases.
- Classes are different, and some might be quicker, others slower. Teachers use different materials and slightly adjust the content to make up for that.
- Sharing of learning material with fellow teachers solely happens inside schools, but generally not with teachers from other schools.
- Topic content is determined by the core curriculum of the state in connection to the school-internal curriculum.
- The detail level of lesson plans significantly decreases after the traineeship. That is mainly because during the traineeship the lesson plans are evaluated by other teaching staff and used for grading the prospective teachers.

These findings with respect to the related work described in [Chapter 2](#), show significant shortcomings of current solutions. Looking at the existing, modern classroom and learning management software, it becomes clear that it does not reflect the teacher's school year planning process. Even though it is theoretically possible to organize

the planning on most platforms, it is not convenient. Thus, teachers mostly rely on self-made systems. This can either mean that information is stored analogously in folders, or digitally on the personal computer as data files.

Moreover, the user research showed that the development of fine-grained lesson plans is not the result of the practical reality, but the assessment pressure in the teacher's education. Therefore, this granularity is solely reached during the teachers' apprenticeship period. The integration of a fine-grained planning helper like PLATON as described in [Section 2.2](#) is, therefore, not a use case for the Schul-Cloud.

3.3 SOFTWARE-BASED PLANNER PROPOSAL

Based on the insights and shortcomings of current solutions, a proposal for a software-based learning matter planner is presented in the following.

3.3.1 *Content Focus*

The qualitative research showed that classes are organized in topic blocks. After the creation, blocks are reused by teachers in subsequent years when getting reassigned to a similar class. However, depending on the achieved learning outcome of a topic block, a teacher might add, edit or delete parts of it for the next class. To map this requirement to a suitable model, two terms are proposed: Topic Templates and Topic Instances.

- *Topic Template* - Describes a generic model for a certain topic block (e.g., genetics for grade 9 in biology). The topic template serves as a boilerplate for actual instances of a topic. It can incorporate multiple lesson units that determine the estimated length of the block. Moreover, it can include different learning materials that are associated with the topic.
- *Topic Instance* - Represents an actual implementation of a topic template (e.g., genetics in the biology class 8a in school year 2018/19). The template serves as the parent of the entity. An instance contains all the data from a template by default but can be modified to reflect the individual needs of a class or the schedule without affecting the template.

To reflect the planning and adjustment of topics in the beginning as well as throughout the school year, a graphical time-based approach is proposed. The evaluation of Strickroth's PLATON system showed that a graphical representation of time was well received by study participants. It was evaluated as more visually present, leading to

a better sense of time. Moreover, drag and drop adjustability was praised as being an easy way to modify the content [19, p.253]. Incorporating these results into the proposal, the following draft views were created:

The form is titled 'Topic template view draft' and contains the following sections:

- Fach:** A dropdown menu with 'Biologie' selected.
- Jahrgang:** A dropdown menu with '8' selected.
- Name:** A text input field containing 'Reproduktion'.
- Anzahl Wochen:** A text input field containing '6'.
- Unterrichtseinheiten pro Woche:** A text input field containing '2'.
- Inhalte:** A large text area for general topic-related notes.
- Unterrichtseinheiten:** A table with 5 rows and 2 columns. The first row is filled with '1. Woche' and 'Einführung Thema'. The other rows are empty.

1. Woche	Einführung Thema
2. Woche	
3. Woche	
4. Woche	
5. Woche	
- Leistungserfassung:** A row of three dropdown menus: 'Mündlich', '5. Woche', and 'Vortrag über X'. There is a small 'x' icon to the right of the last dropdown and a '+' button below it.
- Materialien:** A dashed box containing three file icons: 'PDF' (labeled 'Homework 1.pdf'), 'DOCX' (labeled 'Arbeitsblatt 1.docx'), and 'JPG' (labeled 'Zelle1.jpg'). There is a '+' button to the right of the icons.
- Erstellen:** A button at the bottom right of the form.

Figure 7: Topic template view draft

TOPIC TEMPLATE CREATION Figure 7 depicts the design draft for the topic template creation form. Each template corresponds to a certain subject and grade level. It has a name and length that is determined by the number of weeks and lesson units per week. Moreover, general topic related notes can be added in a text area. Additionally, reflecting the user research on the organization of learning matter as shown in Figure 6, each individual lesson unit can be given another short description. The definition of examinations with specific week and type is another entity of a template. Finally, the topic-related material can be added.

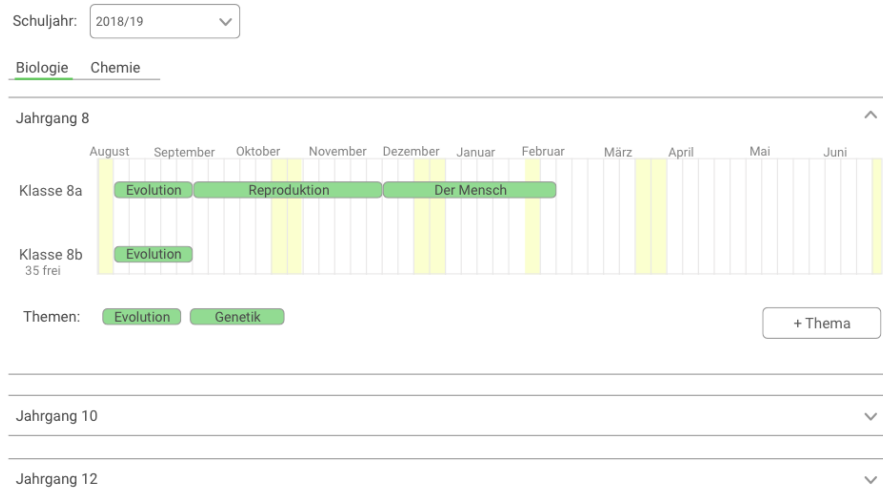


Figure 8: Class configuration view draft

CLASS CONFIGURATION Topic templates are used in the class configuration view. It is the central view to plan the school year ahead. The draft, shown in Figure 8, visualizes the grouping of topic instances based on the school year, subject and grade.

By dragging and dropping a topic template element into a class lane, topic instances can be created. After placing an instance, it can still be rearranged in the row by dragging it to a new position. Consequently, other topic instances will flow to a free position. Furthermore, the length of the topic can be adjusted by dragging the left or right border of the topic element. Thus, teachers can adjust the length of a topic easily if needed.

Holidays or other events (e.g., a project week) that influence the available lesson count are highlighted. Hence, the teacher's plannings can incorporate these circumstances from the start. Besides, the remaining number of free weeks is calculated and printed. This calculation is based on the already configured instances.

Editing and deleting instances, as well as templates is realized via tooltips: When hovering a topic element for a particular time, an edit and delete button is spawned.

In case a topic template is missing for a subject, teachers can get redirected to the topic template view by clicking on the button.

CALENDAR OVERVIEW When the configuration of a school year is done, an overview of all assigned classes of the respective teacher should be available in the form of the draft visualized in Figure 9.

The calendar view gives teachers an orientation for the current state of the school year. It offers three different granularities: a school year, three months and two-weeks view. Not shown in the figure, but also part of the view is a notification system (e.g., to make aware of new

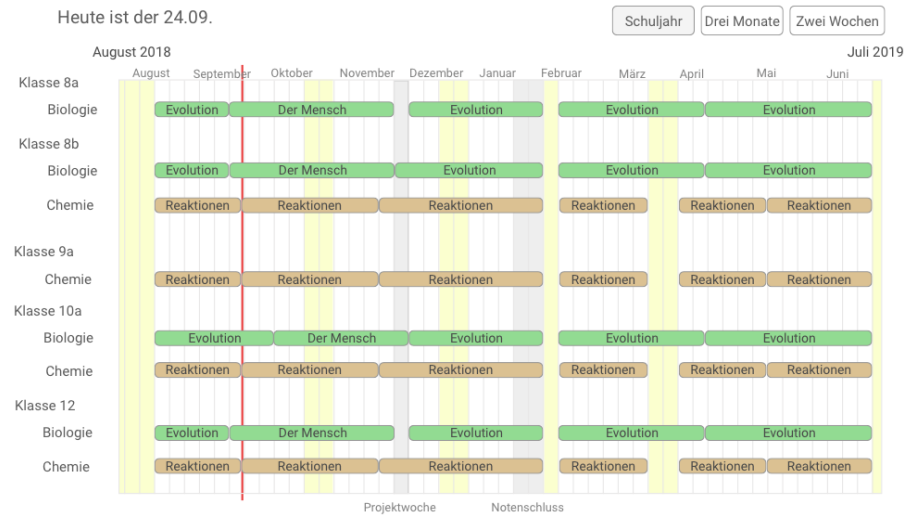


Figure 9: Calendar overview draft

topics starting soon or of upcoming holidays), as well as the visualization of forthcoming examinations configured in the topic templates.

User Journey

A typical user journey for the planner application in the context of the Schul-Cloud could be defined as follows:

1. Two weeks before the new school year starts, a teacher gets his classes assigned by the school administration. When accessing the planner application, he sees the calendar overview (Figure 9). It holds a notification that informs the teacher of the new assignment of classes. The calendar shows the new school year already, including holidays and already determined other events, but is otherwise empty.
2. The teacher decides to start the planning of the learning matter for his assigned classes. He switches to the class configuration view (Figure 8). Due to him having used the planner application in the year before, he already has topic templates for most of his grade level-subject combinations. For those, he can just reuse the topic blocks from last year. Thus, he creates topic instances by dragging and dropping topic template elements into the school year. Also, he adjusts the default length of some topic instances to make them fit with this year's holiday schedule.
3. For one grade level-subject combination, he does not have any templates defined. Hence, he checks the core curriculum by the state and the school internal curriculum to determine the topics that have to be discussed in this subject for the grade level. Next,

he switches to the topic template form (Figure 7) and creates new topic templates according to the stated requirements. He adds general notes about the topic block, learning matter for the individual lessons and material like worksheets. By saving it, he can use the template now to create instances for his class, as he did for already existing templates. If he should get assigned to this grade level/subject combination again in a following year, his definitions can be reused.

4. During the school year, when all the configuration is finished, the teacher mainly uses the calendar view to receive information about upcoming topics and examinations, as well as to get orientation for the current state of his classes. If uncertain changes, like an event, reduce the number of available lessons for a class, a notification is displayed. Afterward, the teacher adjusts his plannings by changing the length of upcoming topics in the class configuration view to incorporate the availability of less time than originally scheduled.
5. When a topic is completed in a class, the teacher revisits the according template and optimizes the learning matter plan. He removes material from the template that did not work well with the students and adds a new worksheet he got from a colleague. Therefore, he iteratively improves the topic for later years.

3.3.2 *Technical Focus*

The proposed software-based planning support for teachers could not only be a part of the Schul-Cloud but be a valuable functionality for other platforms as well. As the related work showed, current solutions do not have dedicated support for school year planning. Moreover, it stands to reason that such feature set could also be a part of a stand-alone platform.

Thus, the implemented functionality is supposed to offer high configurability and adjustability to allow for seamless integration with other platforms. That requires thorough documentation and interface descriptions.

Moreover, it needs to be easy to install. One uncomplicated way to offer the functionality and have clear interfaces would be to offer the views as a library. Versioning and installing could be managed via the node package manager in an uncomplex way.

In general, the implementation needs to be as light-weight as possible. It has to be accessible on mobile, tablets and regular computers. Schools in Germany still tend to suffer from bad internet infrastructure and do not necessarily have the best hardware. As for that, JavaScript is still the most expensive resource to send as described by

Osmani [15]. Hence, to stay fast only necessary JavaScript should be shipped.

CONCEPTS

The development of websites and web applications has always been a field of vast change. The rapid increase in data transfer rates, more powerful hardware, the browser competition and the rise of the mobile internet have led to ever-changing requirements. Thus, frameworks, libraries, best practices, and patterns advance fast. In consequence, the term *JavaScript fatigue* was populated in many blogs and conferences in early 2016 describing the phenomenon [9].

Based on the current state of web development, the incorporation of the existing architecture of the Schul-Cloud specified in [Chapter 2](#) and the technical requirements depicted in [Chapter 3](#), conceptual decisions were made that will be described in this chapter.

4.1 MODULARIZATION

As it was mentioned in [Section 3.3.2](#), the concept implementation should offer the flexibility to be integrable into other platforms with unknown architectures and styles. E.g., platform A might be a multi-page, where buttons are always on the bottom left and platform B a single-page application with buttons on the bottom right.

Due to this inherent ambiguity of the users' requirements, there had to be a technical concept for allowing a necessary amount of configurability, which would allow library users to integrate the functions despite their different environments.

One way configurability can be achieved is by offering small functional entities that can be composed based on the user's demands.

For example, the planner library could expose small entities like a button or a topic label component. By composing these elements into more complex views, the library user can decide on himself whether he places the button on the left or the right. This offers the highest flexibility for the price of increased complexity.

Library users need to understand the small functional entities very well, in order to be able to compose them into something bigger. Moreover, functionality will not just work out of the box: there is a significant time-effort to compose the library functions correctly.

Another option is to offer larger chunks of functionality that work without complex composition. To achieve flexibility nonetheless, a significant number of configurability options have to be offered, and this configuration interface has to be thoroughly documented.

E.g., the library could just export a topic template view component but offer an option interface with the attribute "buttonOrientation" to give the library user a choice to place buttons on the right or the left. Nonetheless, requirements can change during the development process. Hence, there is always a chance that new specifications still cannot be implemented by the available options - despite a complex configuration interface. Furthermore, if too many options are offered, the complexity of the interface rises to a level that makes this approach inefficient.

The library is supposed to fulfill the accumulated requirements from [Chapter 3](#) and to offer very particular functionality. Flexibility is, thus, only needed in some particular areas including styling as described in the subsequent subsection. Therefore, in order to reduce the level of abstractions, it was decided that the general structure of a view should not be configurable. First of all, the configurability of structure adds the need for more complex abstractions. Secondly, it is assumed that the need for a specific structure is much smaller than one for styles and in the case that such fundamental properties as the structure need to be adjusted, the question arises, whether a complete custom implementation is not to favor. That is because using a library that requires complex configuration will add a whole lot of implicitness in the source code, which makes it less understandable and harder to learn for other developers.

Consequently, in this particular case, small functional entities are the worse option, and it was decided to offer functional chunks of high granularity that, nonetheless, provide an interface for very detailed styling. This allows for integration in different applications with their respective corporate identity.

4.2 COMPONENTS

The decision to go for larger chunks of functionality that would work out-of-the-box lead to the question on how to implement the library functionality. It should be possible to integrate the library into the in [Section 2.3](#) described architecture of the Schul-Cloud seamlessly. No additional build steps should be necessary, and it should fit into the controller-view pattern that is implemented. Nonetheless, the library should work in different environments as well: it has to be flexible enough to be includable into single and multi-page applications.

React, a 2013 released library, is as of now the most popular library for creating user interfaces and the popularity is still growing. The main idea is to create encapsulated, composable components that include markup and scripting, and manage their state themselves. Whenever the state changes, a component renders itself and all its

children again.

A summary of the main concepts is presented in the following:

1. *Virtual DOM* - Re-rendering the whole subtree whenever a property changes might be beneficial for programmers because it leads to predictable behavior. However, it is not very performant. Operations on the Document Object Model (DOM), especially adding and removing nodes, are expensive. Therefore, re-rendering on every state change would not be a viable solution.

In order to provide the developers with the comfort of subtree re-rendering whenever the state changes nonetheless, React works with the so-called Virtual DOM. It is an internal representation of the actual DOM. Whenever components try to re-render, changes are made to the virtual DOM first. By batching updates and smart diffing algorithms, the expensive operations on the actual DOM can be optimized, which makes this approach feasible in the first place.

2. *Unidirectional data flow* - The state in a React application is supposed to strictly flow from top to bottom. That makes the behavior and consequences of state changes more comprehensible and easier to understand. To trigger state changes, callbacks (e.g., in case of events) can be passed to executed by child components and, thus, a stateful parent component can be notified that decides how to adjust its state consequently.
3. *Declarative programming* - Currently still the most spread way of writing applications is an imperative one. This programming paradigm makes programmers explicitly write down a sequence of steps to transform one application state into another one. It is all about the *How*. Declarative programming differs in that regard. Programmers define the *What* - the result they want to produce. It is not necessary to care about how this result is achieved. An example of a declarative language is Structured Query Language (SQL). Queries are declarative because they do not resemble an exact execution plan. They solely describe *what* is needed.

The advantage of this programming paradigm is that programs are often simpler to understand, develop and modify according to Apt [1]. Following these benefits, React follows a similar concept: Properties on components are set declaratively.

Being a pure JavaScript library, it can be integrated into every frontend application. Moreover, React is already used in some parts of the Schul-Cloud. Therefore, it was decided to rely on React as an underlying framework for implementing the component library.

As for every library, documenting the interfaces is an essential part of helping the users adopt the functionality. In order to automatically

guarantee exact and up-to-date definitions, it was decided to make use of static type checking.

The most mature solution to date is Typescript, which is backed by the findings of the State of JavaScript survey in 2018. It showed that the satisfaction and adoption rate of Typescript is the highest among developers [17].

The open-source programming language acts as a JavaScript superset and compiles to valid ECMAScript. It is developed by Microsoft and by providing an option to export the interface definitions, it allows for an automatic generation of documentation.

4.3 STYLING

Views and components that get exported by the library have to have very flexible styling because it is required to fit them seamlessly into different web applications that might have varying design guidelines. Nonetheless, the components should have a default styling to minimize the time users have to spend to integrate the components in their application. Not offering any default styles would lead to a massive configuration effort for users of the library before actually being able to use it and increase the integration time.

To further decrease the time, the overall amount of configurability has to be chosen carefully, because it comes with the price of more abstractions that require understanding and, therefore, increases the integration effort. All in all, the goal is to provide a maximum amount of reasonable configurability while retaining a minimal integration time.

FIRST APPROACH: THREE-STEP CSS ADJUSTMENT HIERARCHY

Concept The first idea that was followed to satisfy these requirements was to offer a three-step adjustment hierarchy that allows users to overwrite default CSS properties.

1. General Styles: The general properties of the styling such as primary and secondary colors, the font family or default font sizes.
2. Base Component Styles: The general properties of components such as text area, input or button elements.
3. Specific Component Styles: The properties of specific components in a view, e.g., the save button.

All of these hierarchy steps are optional and offer default values that allow for a smooth integration of the library. Nonetheless, flexibility is given by the simple rule that specific properties overwrite more

general ones. Hence, the library styling can be adjusted only slightly or replaced entirely depending on the requirements of the users. Another requirement is that the technology should not feel like foreign matter: Users of the library should not be locked into specific pre-processors like Syntactically Awesome Style Sheets ([SASS](#)), LESS or libraries like styled-components¹ or JSS². It cannot be assumed that the embedding application uses the respective technology, nor that programmers possess knowledge of it, which is why additional adoption barriers would be created. Concluding the mentioned reasons, the choice was to use Cascading Style Sheets ([CSS](#)), because it is the least common denominator.

Implementation According to the concept, the users of the library should be able to pass their custom [CSS](#) definitions to the view components. Thus, every view component of the library exposes a style property that can be set as in line 25 of the provided code example 1. This property has to implement an exact object shape documented by a Typescript interface.

The code example shows the shape of a definition for the three-step [CSS](#) adjustment concept. Line two to six state the definition of general styles like the primary color or the default font-family, which are properties on an object. Afterward, more specific styles for basic components like an input can be specified. As can be seen in line 8, the interface requires implementing a function, where the earlier implemented general styles are passed as a function parameter. In that case, the font-family for the input component does not have to be duplicated, but the definition from earlier is reused in line 12. The function's return value has to be a string that is made up of valid [CSS](#) code. Lastly, the styling for specific components is to be specified. Line 17 and further show the styling definition for the minus button. As for the base components, this style definition is implemented via a function that has to return valid [CSS](#).

¹ <https://github.com/styled-components/styled-components>

² <https://github.com/cssinjs/jss>

```

1  const styles: ViewStylesInputType<ComponentStylesInputType> = {
2    general: {
3      primaryColor: "#b10438",
4      "font-family":
5        '"PT Sans", "Helvetica Neue", Arial, sans-serif',
6      ...
7    },
8    baseComponents: {
9      input: generalStyle => '
10      display: inline-block;
11      padding: .5rem .75rem;
12      ...
13      font-family: ${generalStyle["font-family"]};
14      ',
15      ...
16    },
17    specificComponents: {
18      minusButton: generalStyle => '
19      border: 1px solid ${generalStyle.primaryColor};
20      background-color: white;
21      ...
22      '
23    }
24  };
25  ...
26  <LibraryView styles={styles} />

```

Listing 1: Style definition for the three-step CSS adjustment concept

For the internal library implementation of that concept, it was decided to create a style provider. This component would take care of the user-defined styles by merging them with the library base styles and pass the result to the individual view components. In that way, style-related concerns could be separated from the view components, and code duplication could be avoided.

A shortened implementation and usage example of the style provider can be seen in code example 2. Line 2 to 15 show the internals of the style Provider: It merges the input styles with the three adjustment levels from the user with the default ones from the library. Afterward, it passes them to its child components by calling the children property with the merging result. This pattern is called a "render prop."³ The following lines 18 to 31 show, how a wrapper component uses the style provider to merge and pass the styles to the view component. It can be seen that the view is, thus, completely decoupled from the styling concerns. Going from here, one could think about reduc-

³ A "render prop" is a technique used for sharing code in React applications. The concept expects that a property (e.g., the children property) is implemented as a function that returns a component. <https://reactjs.org/docs/render-props.html>

ing the boilerplate even more by using a higher-order component⁴ to establish the connection between view component and style provider.

```

1  // Style Provider
2  export default function StyleProvider(props: PropsType) {
3      const { inputStyles, specificComponentDefaultStyles, children } = props;
4      const defaultStyle = {
5          general: defaultGeneralStyle,
6          baseComponents: defaultBaseComponentsStyle,
7          specificComponents: specificComponentDefaultStyles
8      };
9      const mergedStyles = mergeStyles(
10         defaultStyle,
11         inputStyles
12     );
13
14     return children(mergedStyles);
15 }
16 ...
17 // Usage of Style Provider
18 function ViewWithStyles(props: PropsType) {
19     const { styles, ...otherProps } = props;
20
21     return (
22         <StyleProvider
23             inputStyles={styles}
24             specificComponentDefaultStyles={defaultSpecificStyle}
25         >
26             {(styles => (
27                 <View styles={styles} {...otherProps} />
28             ))}
29         </StyleProvider>
30     );
31 }

```

Listing 2: Style provider implementation and usage

Evaluation The approach fulfilled the initially posed requirements. However, these requirements changed after it became clear that the Schul-Cloud plans on migrating to material design - a design language developed by Google that became very popular after its release in 2014. The component guidelines heavily rely on a simple flat de-

⁴ A higher-order component is an advanced React pattern for code reuse. It is a function that expects a component as a parameter and returns a new component. <https://reactjs.org/docs/higher-order-components.html>

sign in combination with animations that simulate physical behavior, in order to copy the characteristics of real materials.

These definite plans showed the lack of the current concept: It would not have been possible to achieve full, seamless integration with other, standard material UI components. The solution solely offers a way to style components dynamically. It does, however, not allow to include JavaScript, which is often used to handle animations and custom behavior - something a style library that implements material design makes heavy use of.

Therefore, the concept had to be iterated.

SECOND APPROACH: COMPONENT-BASED INTERFACE

Concept In order to overcome the shortcomings of the previous concept, a component-based approach was chosen. That means that an interface is offered, where components (e.g., an input or button component) that are used by the whole library can be replaced with custom variants of these components. By providing library defaults, the views will still work out of the box. However, more flexibility to overcome the new requirements is granted.

Moreover, a component-based interface integrates perfectly into the React component architecture. Interface definitions make sure that custom components implement well-defined specifications and get the information they need. Furthermore, accurate API documentation is offered, which is generated from Typescript definitions and, therefore, always up-to-date. Library integrators can use this information to comply with the requirements. Additionally, there is a component library playground provided by Storybook⁵ that helps to understand the implications of properties.

Apart from the component interface, the possibility to define general styling is still given as in the first approach. These styles are used by components that are not exposed by the interface.

⁵ <https://storybook.js.org/>

```

1  import ComponentProvider from '../provider/componentProvider';
2  ...
3  render() {
4    return (
5      ...
6      <FlexContainer>
7        <ComponentProvider.Label
8          caption="Unterrichtseinheiten"
9          type="small"
10         />
11        <ComponentProvider.TextFieldTable
12          rows={rows}
13          onChange={rows =>
14            this.onFormChange(rows.map(row => row.value), 'subjectUnits')
15          }
16        />
17      </FlexContainer>
18    );
19  }
20  ...

```

Listing 3: Usage of component provider

Implementation In order to separate concerns effectively, a component and style provider component were introduced. The component provider can be imported from view components and provides the interface to render basic components. Line 7 and 11 of code example 3 show how a label and a text field table are included via the provider.

It is to note that the provider implementation guarantees type-safety by implementing exact type interfaces. Line three to seven of code example 4 shows an exemplary type definition for a label component. With these definitions, Typescript can make sure that included components comply with the interface.

In order to enable library users to substitute library defaults with their custom components, a method to submit a map of components is exported (line 35). Whenever a component is to be rendered, it takes care of either selecting the default or user-defined custom version of a component as can be seen in line 25.

```

1  import BaseLabel from '../base/Label';
2  ...
3  type LabelPropsType = {
4    caption: string;
5    type?: 'small' | 'medium' | 'large';
6    className?: string;
7  };
8  type ComponentMapType = Readonly<{
9    label: ComponentType<LabelPropsType>;
10   ...
11 }>;
12
13 class ComponentProvider {
14   readonly defaultComponentMap: ComponentMapType = {
15     label: BaseLabel,
16     ...
17   };
18   customComponentMap: Partial<ComponentMapType> = {};
19
20   setupComponentMap(customComponents) {
21     this.customComponentMap = customComponents;
22   }
23
24   getElement<K extends keyof ComponentMapType>(id: K) {
25     return this.customComponentMap[id] || this.defaultComponentMap[id];
26   }
27
28   get Label() {
29     return this.getElement('label')!;
30   }
31   ...
32 }
33
34 const componentProvider = new ComponentProvider();
35 export default componentProvider;
36
37 export function setupComponentMap(customComponents) {
38   componentProvider.setupComponentMap(customComponents);
39 }

```

Listing 4: Partial implementation of component provider

Besides the component provider, a style provider lets the user define default styling for properties like font family or primary color. This is offered via a similar interface as in the described three-step adjustment concept.

Evaluation The field test showed that the cumulated requirements were fulfilled with this solution. While the integration of library views

Fach Jahrgang
Biologie Klasse 8a

Name

Anzahl der Wochen Einheiten pro Woche

Inhalt

Unterrichtseinheiten

1. Woche 1. Einheit	
1. Woche 2. Einheit	
2. Woche 1. Einheit	
2. Woche 2. Einheit	
3. Woche 1. Einheit	
3. Woche 2. Einheit	

Fach Jahrgang
Biologie Klasse 8a

Name

Anzahl der Wochen Einheiten pro Woche
4 **2**

Inhalt

Unterrichtseinheiten

1. Woche 1. Einheit	<input type="text"/>
1. Woche 2. Einheit	<input type="text"/>

(a) View with default styles
(b) View with material styles

Figure 10: Comparison of view styling

would still work out-of-the-box with the predefined styling of the base components, it was possible to adjust the styling and behavior of the components via the provided interface in the desired way. An example of a library view in default and material design can be seen in [Figure 10](#).

IMPLEMENTATION

Following the set of requirements depicted in [Chapter 3](#) and the derived concepts of [Chapter 4](#), this chapter describes the implementation of the planner library¹ and its integration into the Schul-Cloud. At first, the general philosophy behind the technical realization of the concept proposal is described. This includes the documentation of used libraries as well as the reasoning behind these decisions. After that, the library's view components are specified. This includes the property interfaces, but also interesting implementation details. After this, the integration process of the library into the Schul-Cloud is presented. Hence, this subsection contains the initial changes made to the client's build setup, followed by the documentation of the adjustments in the frontend. At last, the extensions to the backend are stated.

5.1 PLANNER LIBRARY

This section depicts the planner library's implementation details. In the beginning, the code's general philosophy is described: It is shown, how React and Typescript were used to create a well-documented component architecture. Moreover, the emphasis on a small bundle size is explained, and the reasoning behind the usage of third-party dependencies is clarified.

5.1.1 *Implementation Philosophy*

REACT Being around for about five years now and having originated a large community, a substantial number of best practices for the development of React applications have been developed.

Following this collected knowledge, the goal was to implement the library with the awareness of the current state of the art patterns.

A good React architecture breaks the functionality into small composable component entities that can not only be reused but keep the implementation clean and understandable.

This philosophy was followed as shown in [Figure 11](#). The example depicts the composability of components in the calendar view. The complex YearlyCalendar consists of smaller entities like the ClassRows

¹ <https://github.com/schul-cloud/planner>, Accessed: 07.12.2018

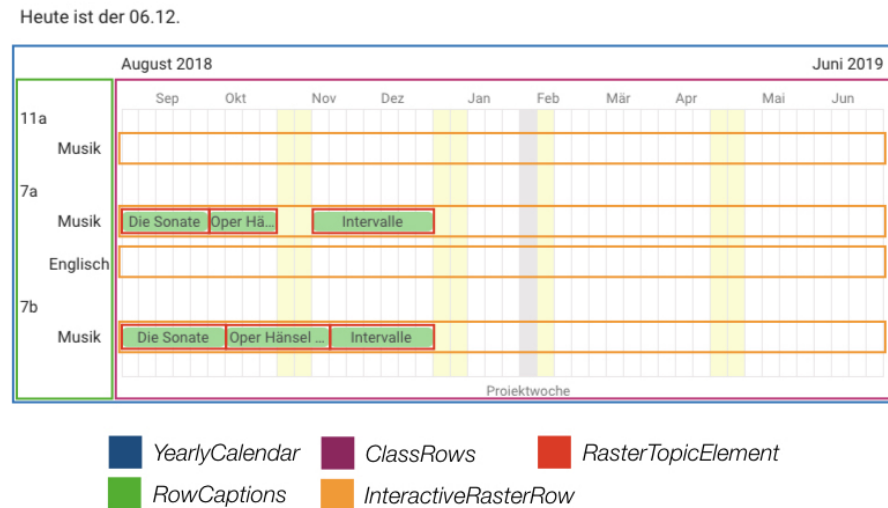


Figure 11: Usage of component composability

and RowCaptions component. By further breaking down the class rows into single raster rows, which finally include topic elements, the complexity of the components can be reduced. Additionally, the RasterTopicElement can be reused in other views that need these elements.

Thus, a clear, composable component hierarchy is achieved.

TYPESCRIPT To make sure that interfaces are generally self-documented, Typescript was used from the very start of the implementation. During the development, it was consequently updated, whenever a new version would have been released. As of the date of the writing, Typescript 3.1.1 is used.

Of course, solely using Typescript is not a guarantee for good typing. However, by defining strict Typescript compiler options like *strictNullChecks*, *noImplicitAny*, and *strict*, a proper typing is forced to a certain degree, in order to compile the library. These options ensure that all the code, including the one from third-party libraries, has declared type interfaces. Especially types for open source libraries are very helpful and can often be included without much effort. In most cases, either library authors offer typings by themselves, or the community creates and shares definitions². In that way, Typescript can provide much confidence for the development process, but also help unfamiliar people understanding the code base. Moreover, it allows for increased development speed when using a code editor like Visual Studio Code³ that has excellent support for Typescript and uses the typings for smart code intelligence.

2 <https://github.com/DefinitelyTyped/DefinitelyTyped>

3 Visual Studio Code is a free code editor released by Microsoft. It offers excellent support for writing Typescript code by providing code intelligence that uses insights from the defined types.



Figure 12: Exemplary Typescript error

In general, static type systems can prevent many bugs before they happen. Be it wrong data types, misspelling variable names or referring to wrong object properties as visualized in [Figure 12](#).

On the other hand, there are also disadvantages, when deciding to use a static type system like Typescript. First of all, there is the overhead of having to annotate types. Furthermore, the amount of code is increased, and additional overhead when reading the file is generated. Developers may feel like they have to write code in a specific way to solely please the type system, which can be a major annoyance. Additionally, in case that the type system's error messages are not clear enough to give the programmers a clear action item to pursue, they can feel helpless. Moreover, type systems are not perfect. Especially in combination with unclear error messages, they can make the developers lose confidence resulting in not knowing whether an error originates from an actual error or the type systems incapacibilities.

It is to say, however, that Typescript is a very mature language. Backed by Microsoft, it is updated frequently, and the community's wishes play a significant part in its development. Thus, the popularity and adoption rate has lead to a very stable system that keeps its promises. Especially in the context of this project, it was decided that the advantages outweigh the disadvantages.

DEPENDENCIES AND BUNDLE SIZE One focus point for the development was to minimize the bundle size of the library. As described in [Chapter 1](#) many schools in Germany, especially in rural areas, suffer from the lack of proper network infrastructure. Given that environment, it is clear that the reduction of asset sizes has to be an important optimization priority for the development of the Schul-Cloud. For the implementation of the planner library, this means that especially the use of third-party libraries has to be carefully considered since external dependencies are a frequent reason for large bundle sizes.

Following this priority, only three external packages (excluding the general React dependency) were added.

- *styled-components* - React has changed the way, how web development is done. Before it became popular, markup and scripting were generally strictly separated. This was mainly done be-

cause it simply felt right. [HTML](#), JavaScript ([JS](#)) and [CSS](#) were kept in separated files, and also popular pre-React-era libraries like Angular did not change this. The React team challenged this assumption. By separating concerns on a component level, they showed that bringing business logic and markup together into one file is a valid alternative. The main reason being is that it brings closer to what is logically connected.

Styled components, a library released in 2016 spun this thought further by not only mixing markup and scripting but also bringing styling to the component level. This method is often referred to as [CSS in JS](#).

```

1 import styled from 'styled-components';
2
3 const Text = styled.div`
4   color: ${({ isActive }) => isActive ? 'green' : 'red'};
5   text-align: center;
6 `;
7
8 export const Component = (props) => (
9   <Text isActive={props.isActive}>Hello World</Text>
10 );

```

Listing 5: Styled components example

By writing pure [CSS](#) in template literals, the power of JavaScript can be used to conditionally style components as done in code example 5. The obvious advantage is that essentially all logic concerning a component entity - from markup and styling to logic - is actually in one place. Moreover, developers do not have to care for class names anymore, because they are automatically generated. This also prevents dead [CSS](#) code, which almost always becomes a problem for large projects, because removing a component will directly lead to removing its corresponding [CSS](#).

In order to embrace the React component system in the best way, it was decided to make use of styled components instead of relying on a classic styling approach. The library has a size of approximately 16 KB⁴, which is considered an acceptable amount.

- *@tippy.js/react* - Tippy.js is a library that offers customizable JavaScript tooltips and popovers. With its small size of 14 KB⁵, it made a custom implementation redundant.

⁴ The size is stated for the minified and gzipped version of the library.

⁵ See [4](#)

- *react-dnd* - As described in [Section 3.3.1](#), drag and drop functionality needed to be implemented for the library. Traditionally, that is a challenging domain in the browser, due to the varying support of the different vendors. It is doubtful that a self-made implementation could have been stable enough to fulfill the requirements in an acceptable time frame. It was decided that the more viable solution be, to focus on using an established and popular library like react-dnd. The total size of about 24 KB⁶ is reasonable.
- *lodash* - Lodash is a JavaScript utility library. It offers among others functions for memoization and throttling. For the planner application, only a small subset of the library's functions are used. In total, lodash contributes about 8 KB⁷ to the bundle size.

Thus, In total external dependencies have a size of approximately 62KB. In a bad case scenario, where only a 3G⁸ connection is available, this is equivalent to about one second of download time.

5.1.2 Components

Following the requirements depicted in [Chapter 3](#) four general views had to be implemented:

- Class Configuration View
- Topic Template View
- Topic Instance View
- Calendar View

The following subsections will describe the implementation details of the respective component views. Moreover, there are extensive descriptions of the component's interfaces which includes the data that needs to be passed to the component, as well as the callback functions that are offered to receive the user input.

5.1.2.1 Class Configuration View

[Figure 13](#) shows the class configuration view. It is the central configuration hub for teachers to plan their school year. Here, they can configure the topics for their classes and subjects by adding, deleting, reordering and resizing topic blocks.

⁶ See [4](#)

⁷ See [4](#)

⁸ 3G has a download rate of approximately 384 kbit/s

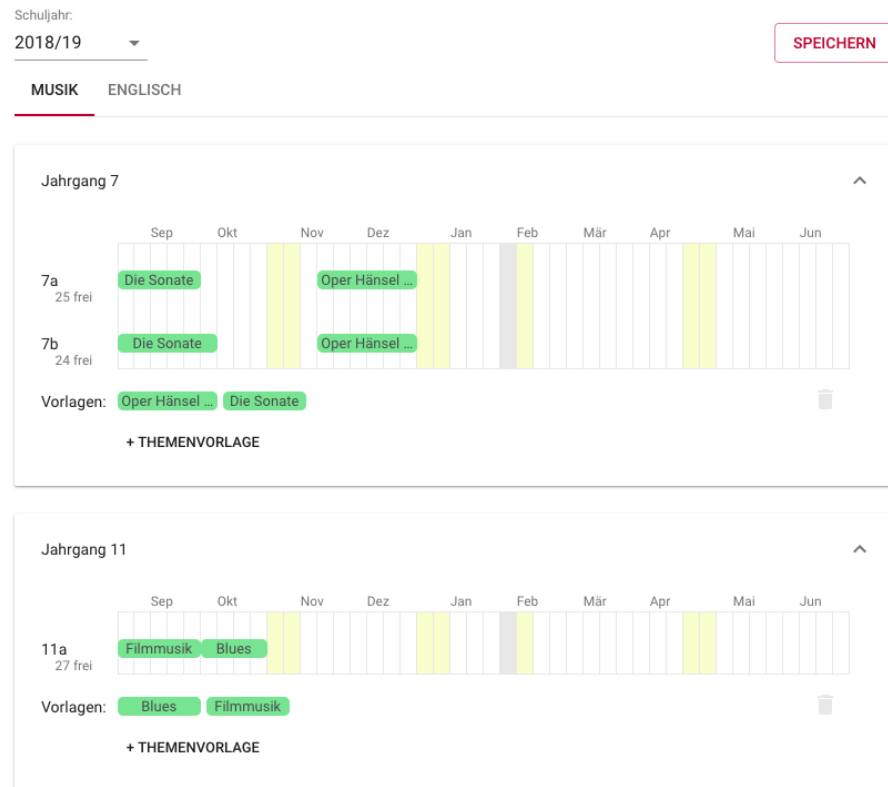


Figure 13: Interface of class configuration view

INTERFACE

Properties

- allClassTopics - AllClassInstancesType<TopicIndexType>⁹**
 A complex, nested JavaScript object that includes the initial topic instance data for the view. The complexity comes from including all the school years (e.g., 17/18, 18/19) with their respective subjects (e.g., biology and chemistry for 17/18). The individual subject again can have different class levels (e.g., eighth and 10th grade) and a class level can have multiple actual classes (e.g., 8a and 8b). Finally, the classes have their respective topic instances assigned (e.g., topic one and topic two for class 8a). In the end, topic instance data allows displaying the whole configuration of classes in an ordered way.
- allTopicTemplates - AllTopicTemplatesType¹⁰**
 The allTopicTemplates property includes all the topic templates a user has defined. Every template belongs to a certain subject (e.g., biology) and class level (e.g., 8th grade). Therefore, it is a nested JavaScript object that categorizes the templates by divid-

⁹ AllClassInstancesType<TopicIndexType> definition can be found in code example 8

¹⁰ The AllTopicTemplatesType definition can be found in code example 9

ing them by the subject id and class level id.

Thus, templates are displayed for the right subject and class level.

- `initialSchoolYearId` - string
Defines the initial school year id that is selected for display. If no id is given the first id found in the `allClassTopics` data is used. The idea is that the current school year is always selected by default. In general, it seems advisable to switch the school year, when the first half of the summer holidays is passed because the planning phase of teachers for the new school year begins earlier than the actual start date. Depending on the federal state, however, the date, when to switch school years differs, because holidays are varying.
- `schoolYearData` - { [schoolYearId: string]: SchoolYearType; }¹¹
An object map assigning the id of the school year to the start and end date. The two dates have to be given in UTC format to make sure that daylight saving time dates and times are still clearly defined. Even though the time format includes data up to milliseconds solely the year, month and day information is used.
The data is needed to determine the number of columns that have to be rendered as well as the right display of the corresponding months' captions.
- `eventData` - { name: string; color?: string; utcStartDate: number; utcEndDate: number; }[]
The last property that needs to be supplied is the event data. It is an array of objects that include information like start date, end date, a name, and an optionally associated color. These events incorporate official holidays as well as school internal happenings like a project week. They are important for the teachers' considerations when planning the school year ahead because they reduce their time pool. Hence, particular weeks are displayed in a different color - either the specified one or a default one - and are taken into consideration when calculating the remaining time.

Callbacks

- `onAddTemplate` - (selectedSubjectId: string, classLevelId: string) => void
The function is called, when the user clicks the "+ THEMENVORLAGE" button and supplies the context of the click by providing the subject id and class level id. It is expected that a redi-

¹¹ The `SchoolYearType` definition can be found in code example 10

rect to the Topic Template View takes place when this method is called.

- `onEditTemplate - (templateId: string) => void`
Whenever the edit icon of a template is clicked, this function is called with the respective id of the template.
- `onDeleteTemplate - (templateId: string) => void`
The `onDeleteTemplate` function gets executed in case the user clicks the trash icon of a template.
- `onEditInstance - (instanceId: string) => void`
This function is called whenever the edit icon of a topic instance is clicked. By providing the instance id, a redirect to the respective edit view of the topic instance could be initiated.
- `onSaveClassInstances - (instances: AllClassInstancesType< LocalTopicIndexType >) => void12`
In case the "Speichern" button on the top left gets pressed the `onSaveClassInstances` callback is executed with the current local state of all the class topic configuration, which includes all the changes made by the user.

IMPLEMENTATION DETAILS The class configuration component is without a doubt the most complex and fundamental view of the library. It is the very core that has to guarantee that users are able to configure their school year as desired with ease.

Thus, a priority was to implement a fluid and intuitive user experience. Drag and drop resemble a very accurate reflection of the actual physical process of moving something somewhere. Therefore, the arrangement, creation, and adjustment of topic elements were entirely designed with drag and drop.

The details of the implementation will be explained in the following. Moreover, it should be documented, how the insertion of topic elements was handled. Afterward, the caveats of the resize functionality will be described in more detail. At last, measures taken to reach a reasonable performance are documented.

Drag and Drop As described in [Section 5.1.1](#), `react-dnd` was used to realize the drag and drop feature. The library offers a set of higher-order components to augment components with necessary functionality while keeping the rest of the implementation decoupled.

For the planner library, the ES7 decorator syntax, which is already supported by Typescript, was used. Although the language feature is still just a proposal in the TC-39 process, it is already classified as stage two, which means the committee expects the proposal to be

¹² The `AllClassInstancesType< LocalTopicIndexType >` definition can be found in code example [11](#)

"eventually included in the standard."¹³

Consequently, the `@DragSource`, `@DropTarget`, and `@DragDropContext` decorators were used to augment library components giving them the ability to be dragged or to receive drop events. A `DragDropContext` is a surrounding environment that always has to be present in the component tree above a drag source or drop target. As

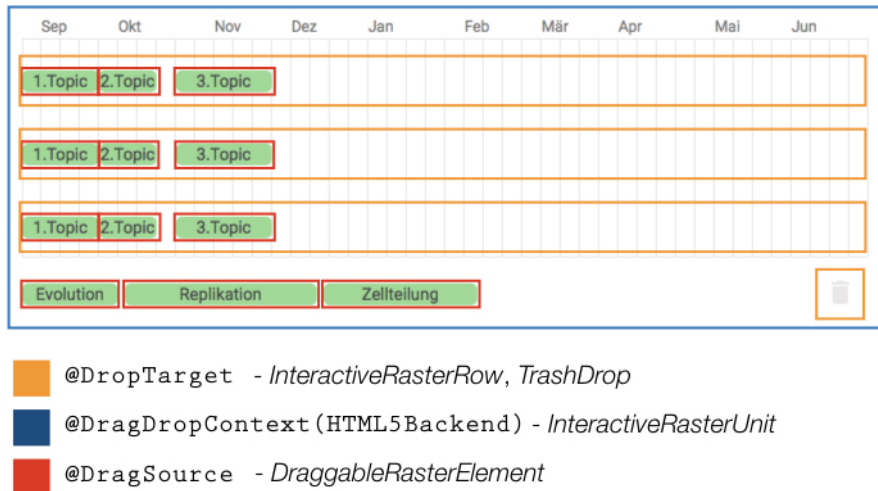


Figure 14: Visualization of the drag and drop components

Figure 14 shows, every `InteractiveRasterUnit` component functions as a separate drag and drop context. Further down the component tree, there are `DraggableRasterElement` components. These entities can have two different types, as they can be topic templates or topic instances. Depending on this information the drop targets have to behave differently.

These targets are `InteractiveRasterRows` and a `TrashDrop` component. Therefore, if a `DraggableRasterElement` gets dropped on them, they get notified and can react accordingly. However, before a topic gets actually dropped, there are other events the components respond to. If a drag operation is initially started, they will react by changing their background color. Hence, they communicate to the user that they can interact with the drag source. Furthermore, if the element gets hovered over them, they will change their background to green to signal that the drop operation would be successful now.

Instant feedback on where an element would be inserted if it were dropped onto the `InteractiveRasterRow` is essential to users.

As described in Section 4.2, React features a uni-directional data flow. That means that the state flows from top to bottom and it should solely determine how components are rendered. This is, why only

¹³ <https://tc39.github.io/process-document/>, Accessed: 04.12.2018

state changes trigger a re-rendering of the component tree.

Thus, in order to display a preview on how a topic would be inserted, the state has to be updated. However, the state of the current topic instances cannot be changed because if the user aborts the operation, there would be no way to restore the old state.

To realize this functionality nonetheless, the process of soft-creating and soft-relocating an element was introduced. Whenever a drag process starts, the state of topic instances is not determined by the global state of the class configuration view anymore, but it is replaced with a temporary state, which is held by the `InteractiveRasterUnit` and represents a copy of the global state in the beginning. Therefore, when the user drags an element (e.g., a new topic) over an `InteractiveRasterRows`, the `DropTarget` will execute a callback to soft-insert the element. This will, however, only update the temporary state, which will trigger a re-render and display the change consequently.

Now, when the user drops the element, the source of truth will switch back to the global state. However, there are two options:

First, the user aborted the process. In that case, nothing has to be done. By switching back to the global state, the old condition is restored.

Second, the user dropped the element successfully. If so, the `InteractiveRasterUnit` gets notified from the drop target and will execute a callback to replace the existing global state with the temporary state. Hence, the change is persisted, and the drag and drop operation is handled.

Insertion The last paragraph described how the general drag and drop functionality of the class configuration view was realized. A directly related nontrivial implementation detail is the question on how to decide where to insert an element.

This question is trivial when space is unoccupied at the given position. If other elements are in the way, the answer gets more complicated. The easiest way were to allow insertion on free space solely. However, this is not a very user-friendly solution. Often multiple micro-operations, like moving other elements aside, were necessary before an element can be inserted on the desired position.

Instead, it was decided to try to insert an element always as close to the specified position as possible and move other elements out of the way if necessary. In order to make the algorithm less complicated, it was decided to just relocate elements to the right.

Thus, if an element is to be inserted and it is located over another element, the first step is to determine an actual insertion position. This is because it might not be possible to directly insert the new element at the specified position due to other existing elements.

Therefore, it has to be determined whether the new instance needs to be inserted before or after the element that is currently occupying

the requested position. If the insert position is in the first 50% of the existing element's length, the actual insert position is kept as it is. If the position is part of the right 50%, the actual insertion position is set to be directly after the existing element.

Now that an actual insertion position is determined, the element is tried to be inserted. Hence, all existing elements that are in the way are moved to the right until they find enough space to be inserted again.

This process can conflict with other elements in the row that are located even more to the right. In that case, they are moved as well until space for them is found. If no space is left, which happens if the row is too full to make space for another element, the operation is aborted, and the new element is not inserted.

As can be seen, there is significant complexity in the insertion process of topic elements. It remains to be seen if this approach's induced behavior is comprehensible enough for the users of the library. Although working in general, it is assumed that this approach still holds optimization potential.

Resize The resizing mechanic of topic instances in the rows of a course is another measure taken to ensure that users can adjust and configure their school year quickly and intuitively.

The main complexity of this feature does not lie in the drag operation per se, but the interaction and adjustment of other elements in the row. In general, two cases have to be considered: a topic tries getting smaller, or it tries getting larger.

The first case is trivial since no problems can arise here. The only thing to consider is to make sure that the length of the element is at least one unit.

The second case needs further investigation. If the element tries to expand into free space, the case is trivial as well. It only has to be assured that the total length of the row is not broken. However, if another topic element is in the way, the process gets more complicated. An easy solution would be not to allow an element to expand into occupied space. However, that would make the user always having to perform two drag operations to adjust the length into occupied space: decrease the length of the neighbor element and increase the length of the target element.

To reduce the required operations to one, it was decided to automatically decrease the length of a neighbor when an element tries to expand into its territory. Nevertheless, the new length of the neighbor can never be less than one tile.

Certainly, it is to say that this could not be the best solution. In the end, user testing is required to identify the most optimal implementation.

Performance As described, state changes in components result in the re-rendering of all their children. Although, React uses the virtual **DOM** to reduce expensive operations on the real **DOM** and as described in [Section 4.2](#), too many render calls in a short time can impact the performance. The drag and drop creation and relocation, as well as the resizing of topic elements, can lead to a problematic amount of render calls, due to constant event firing. Therefore, some basic performance optimization had to be implemented.

Regarding the resizing of topic elements, when dragging an element, mouse move events are constantly emitted. However, not every movement results in the necessity to adjust the length of the topic element. To prevent re-rendering in this case, the resize handler functions (`handleElementSizeChangeLeft` and `handleElementSizeChangeRight`), were wrapped with the `memoizeArguments` function. This method memoizes the arguments of a function and only executes it if the arguments change in comparison to the last function call.

A second optimization made, was the usage of `PureComponent` in the `InteractiveRasterRow` and `RasterUnitContainer` components.

Usually, React components extend `React.Component`, but it also possible to extend `React.PureComponent`. The difference is that a pure component will conduct a shallow comparison of props and state in the `shouldComponentUpdate` method and will return `false` if the properties turn out to be equal. This function can tell React if a component and its children need to be re-rendered. By default, the method returns `true`, which tells React to call `render`. Typically, this is fine, because rendering in React does not mean that elements are actually unmounted and remounted.

Yet, in this case, preventing a render call high up in the component tree yields measurable performance improvements because it will prevent subsequent render calls for all child components. When an element is resized, it will result in a global component state, which triggers a re-render of the whole class configuration view, even though only the respective `InteractiveRasterRow` with the resized component eventually needs to re-render. Cutting branches high up in the component tree by using pure components is, therefore, for this case a measurable performance optimization.

To prevent unnecessary calculations for the insertion of templates and relocation of instances, the `softRelocateTopicElement`, and `softInsertTopicElement` functions are solely invoked on changes. To do so, it is checked whether the insertion index changed for the relocation and whether the row, insertion index or width changed for the addition of a new instance.

Moreover, the `softRelocation` and `softInsertion` functions were throttled to a 150ms interval. This means that these methods can only be invoked every 150ms. In that way, too many subsequent calls that

would impact the performance are prevented.
By applying these measures, a fluid rendering could be achieved.

5.1.2.2 Topic Template View

The topic template view is the detailed configuration form for templates. It is used for creating new templates as well as editing existing templates. The differences between the two modes are small. Therefore, to reuse as much logic as possible and reduce code duplication, a NEW and EDIT mode is offered in the component interface.

Fach Jahrgang
Musik 7

Name
Die Sonate

Anzahl der Wochen Einheiten pro Woche
1 1

Inhalt

Unterrichtseinheiten

1. Woche 1. Einheit

Leistungserfassung

+

Materialien

Orchester Image2.jpg Entfernen

Dateien hochladen...

ERSTELLEN

Figure 15: Interface of topic template view in NEW mode

INTERFACE

Properties

- mode: 'NEW' | 'EDIT';
The mode property has to be supplied and can either be NEW or EDIT. The sole difference is the rendering of varying buttons. While for the NEW mode a single create button is rendered, the

EDIT mode renders a delete and a save button. That results in different requirements for the supplied properties: e.g., no on-Create callback has to be supplied when the view is in EDIT mode. The details are specified in code example 20 in the appendix.

- `initialValues?: FormValuesType`;¹⁴
InitialValues is an optional object that supplies the initial form data to the view. It is a partial object, which means it does not have to be complete (not all values are required to fill in). It consists of several string properties like subject or name and some array properties.
- `valueOptions: FormValuesOptionsType`;¹⁵
This object contains two arrays for the subject and grade level options that can be chosen by the user in the respective selector component. The single array elements contain id and a text. Hence, when a user selects a subject the name of it is shown, but in the background, the id is remembered to have been selected.

Callbacks

- `onCreate: (values: FormValuesType) => void`;
The provided function is executed when the "Erstellen" button is clicked. It passes the current form values for further actions and solely has to be defined in the NEW mode.
- `onSave: (values: FormValuesType) => void`;
This callback is executed, when the "Speichern" button is clicked in the EDIT mode. The function does not have to be supplied if the component is rendered in the NEW mode.
- `onDelete: () => void`;
The onDelete function is called when the "Löschen" button is clicked. It is solely required in the EDIT mode.
- `onFileClick: (file: FileType) => void`;¹⁶
This method is executed when a user clicks on a file of the File-Selector component. It passes a file property as a parameter.
- `onFileAdd: OnFileAddType`;¹⁷
This method is executed for every file a user tries to upload. Thus, when a user clicks on "Materialien hochladen" and selects two files in the following dialog, the callback will be executed two times. This allows the user of the library to implement the upload functionality according to his wishes.

¹⁴ The FormValuesType definition can be found in code example 12

¹⁵ The FormValuesOptionsType definition can be found in code example 13

¹⁶ The FileType definition can be found in code example 15

¹⁷ The OnFileAddType definition can be found in code example 14

Notably, the method provides three arguments: an object with a file and temporary id and the two methods `onComplete` and `onError`. These methods are supposed to be called with the temporary id of the file when the upload was successful, respectively failed.

This information is used by the `TopicTemplateView` to remove the loading state and, if the upload was successful, to update the state, in order to display the new file in the form.

- `onFileRemove: (file: FileType) => void;`
When a user clicks on the "Entfernen" label in [Figure 15](#), this callback is executed with the file information.

IMPLEMENTATION DETAILS The topic template view being just a form component, does not hold much complexity.

It has an internal state that holds a `currentValues` property, which is set to the `initialValues` property on startup and represents the single source of truth for the form components. Therefore, whenever changes in the form elements are made, they execute a callback that will eventually update this state. Then, the data values are propagated back to the individual components.

5.1.2.3 Topic Instance View

The topic instance view allows viewing and editing the properties of an actual topic instance that was derived from a topic template once. Due to this, the displayed information and components of these two views have high similarity. The differences are solely the inability to change the subject and the class value, and the display of the parent template for now.

It might have been an option to combine the topic instance and the topic template view. However, the template view already has some logical complexity, due to the two different modes. Adding another logical branch would complicate the code further. Moreover, there is a high probability that these views will diverge even further in the future. Hence, it was decided to create a separate view component.

INTERFACE

Properties

- `initialValues: Partial<FormValuesType>;`¹⁸
According to the same property described in the topic template

¹⁸ The `FormValuesType` definition can be found in [code example 16](#)

Fach	Jahrgang
Musik	7b
Name	Elter-Template
Oper Hänsel und Gretel	Oper Hänsel und Gretel
Anzahl der Wochen	Einheiten pro Woche
1	1
Inhalt	
- Hänsel und Gretel	

Unterrichtseinheiten	
1. Woche 1. Einheit	Einleitung

Leistungserfassung

Schriftlich ▼ ▼ Kontrolle ×

+

Materialien

Hänsel und Gretel Bild.jpeg Entfernen

Dateien hochladen...

LÖSCHEN SPEICHERN

Figure 16: Interface of topic instance view

view, it holds the initial form data information. However, it also includes a parent template property, which has a name and id.

Callbacks

- onSave: (values: CurrentFormStateType) => void;¹⁹
This method gets invoked in case the "Speichern" button gets pressed. It contains all the form data as the first function argument allowing for persisting the modified information.
- onDelete: () => void;
Executed callback function when the "Löschen" button gets clicked.
- onTemplateClick: (id: string) => void;
The onTemplateClick callback is called when the user clicks on the parent template. It includes the id of the template.
- onFileClick: (file: FileType) => void;
- onFileAdd: OnFileAddType;
- onFileRemove: (file: FileType) => void;
The onFileClick, onFileAdd and onFileRemove methods are im-

¹⁹ The CurrentFormStateType definition can be found in code example [17](#)

plemented and function as their counterpart in the topic template view. Thus, the interface description of [Section 5.1.2.2](#) is solely referenced here.

IMPLEMENTATION DETAILS As the topic instance view is related to the topic template view, the complexity of this view is equally low. Similar to the template form, the component holds an internal state that is initially set to the `initialValues` property's value. Whenever a form element registers a change, it will execute a callback that eventually adjusts the state.

5.1.2.4 Calendar View

The calendar view is meant to be the central information point of the planner functionality. To provide teachers with the information about current topics and upcoming important dates, it offers three previews of different granularity: a two-week ([Figure 19](#)), three-month ([Figure 18](#)) and whole school year ([Figure 17](#)) view.

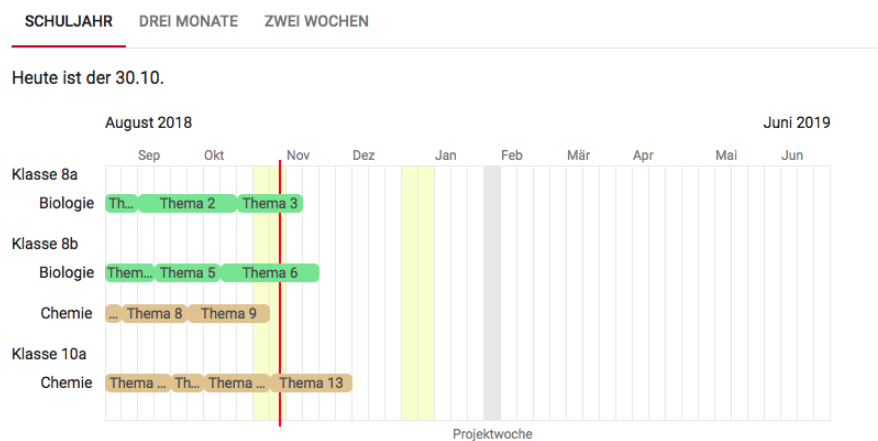


Figure 17: Interface of calendar view - school year

INTERFACE

Properties

- `schoolYear`: {`utcStartDate`: number; `utcEndDate`: number; }
The `schoolYear` property passes the start and end date of the described school year in UTC format to the component. As for other UTC dates, only the year, month and day information is

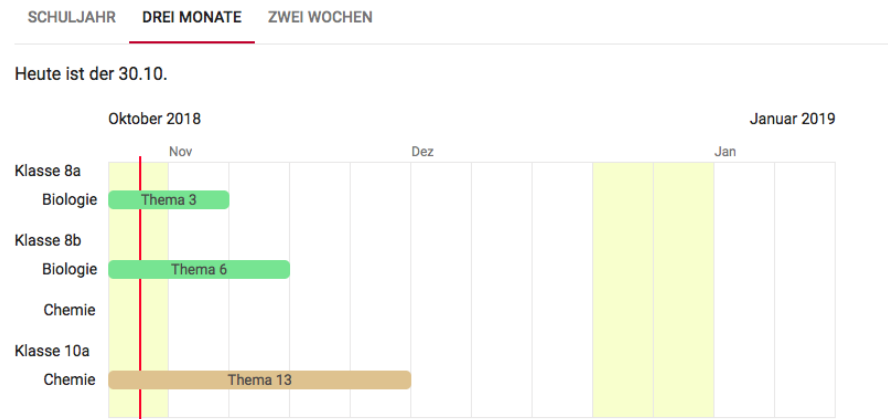


Figure 18: Interface of calendar view - three months

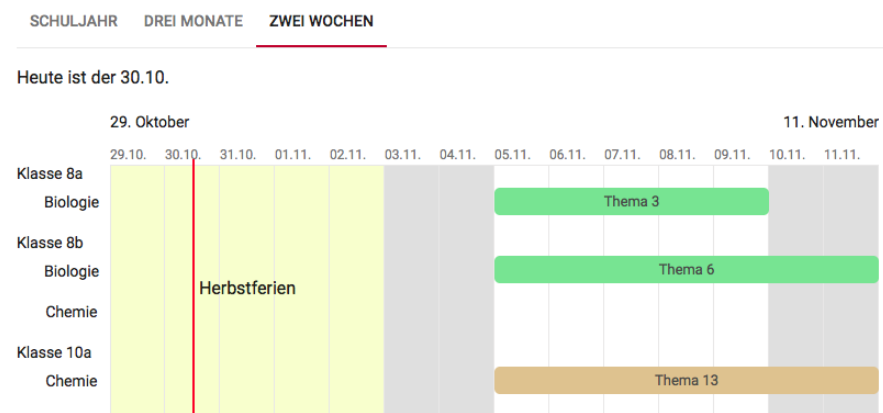


Figure 19: Interface of calendar view - two weeks

used ("Jan 01 1970 00:00:00 UTC" and "Jan 01 1970 23:59:59 UTC" are both treated as Jan 01 1970). This data is solely used for the whole school year preview mode of the calendar that can be seen in [Figure 17](#). Therefore, neither the three-month - nor the two-week preview use the property. Moreover, it is to note that the two dates are expected to always exactly depict the first school day and the last school day of a school year.

In the school year preview mode, the schoolYear data is eventually used to generate the correct number of raster columns, as well as to determine the right place for the upper month labels.

- utcToday: number;

UtcToday is a number describing the UTC date that depicts the current date. It is used to display the current date above the calendar component. Furthermore, it determines, where the red current day line is painted.

For the Three-Month-Mode and Two-Weeks-Mode the date is

used to determine the displayed time window as can be seen in [Figure 18](#) and [Figure 19](#).

- `classTopicsData`: `ClassTopicsDataType`;²⁰
The `classTopicsData` includes the main data that is displayed in the calendar view. The structure of the object determines the different classes and their respective subjects. The subjects again are made up of multiple topics that include UTC start and end date. This data is then used to create the topic labels with caption in the respective row.
- `holidaysData`: `EventType`;²¹
The property does, as the name suggests, include the holiday data described by the `EventType` interface. It resembles an array of objects that include a UTC start date, a UTC end date, an event name, and an optional color string. This information is used to highlight and display the holidays on the calendar.
- `otherEventsData`: `EventType`;
`OtherEventsData` resembles the same data type compared to the previously described `holidaysData` property. They are separated because the display of holidays and other events differs in some parts of the calendar view as can be seen in [Figure 17](#).

Callbacks

- `onTopicInstanceClick`: `(id: string) => void`;
The `onTopicInstanceClick` callback is executed if the user clicks on one of the topic labels. It could, for example, be used to redirect to the detailed view of one topic.

IMPLEMENTATION DETAILS The calendar view is a container component that renders one of the three different calendar types. Based on the current selection the `SchoolYear`, `ThreeMonths` or `TwoWeeks` calendar is rendered. While the school year and three months view both base upon the `WeekClassRows` component that functions similar to the rendering of the class configuration view, the two weeks calendar holds a higher complexity.

This is because, in contrast to the others, single days are displayed and not weeks. Thus, the names of holidays and events are rendered. Moreover, the weekend is displayed differently and depending on if a topic instance ends in a week or not, it either crosses the weekend columns or it ends before them, as can be seen in [Figure 19](#).

The key functionality that realizes this is the `getEventMaps` function, which generates three different data structures: a `columnColorMap`, `eventTypeMap`, and `eventArray`.

²⁰ The `ClassTopicsDataType` definition can be found in code example [18](#)

²¹ The `EventType` definition can be found in code example [19](#)

As the name suggests, the column color map is used to print the respective colors in the columns. As events and weekends can overlap the following hierarchy is used: First, the weekends are printed. Secondly, individual event colors are shown and lastly, the regular holidays are displayed.

The event type maps indices to event types like holidays or weekend. This data is later used to determine if the indices of topic instances have to be adjusted for the calendar display. Given, for example, a topic instance is defined to end on a Sunday but holidays start on Wednesday already, the instance should be displayed to end on Tuesday.

At last, the event array is used to generate the event labels in the calendar. It is to note though that the generation of this data is not as trivial as it may seem. If a holiday spans both displayed weeks, there have to be two labels: one for the first and one for the second week. The array data, thus, has to include two entities for the same holiday.

5.2 INTEGRATION INTO SCHUL-CLOUD

To embed the previously described library views into the existing Schul-Cloud application, changes and extensions to the client and server-side of the platform were implemented. Furthermore, an adjustment to the general build setup was introduced.

This section will document and explain these modifications in detail.

5.2.1 *Changes to Frontend Pipeline*

The initial baseline study of the Schul-Cloud-Client architecture uncovered a significant shortcoming of the build pipeline: No bundling was supported. This insufficiency posed a major inconvenience for the integration process.

BASELINE While the frontend pipeline of the Schul-Cloud client did support minification and transpiling as described in [Section 2.3](#), it did not provide bundling of JavaScript files. Consequently, it was not possible to require functionality from other js modules. All scripting for one site was, therefore, either implemented in a single file or shared via global variables. The first led to confusing js files with hundreds of lines of code, while the second polluted the global scope and introduced a whole lot of implicitness when these variables were used.

Another issue was that the missing bundling step required manual management and integration of third-party libraries via script tags. That has multiple disadvantages: Firstly, several libraries lie in the

repository as raw files bloating the repository size. Secondly, it makes the upgrading process for libraries tedious, because the update has to be done by manually replacing the library file in the repository. Moreover, the actual version number is not documented, which further complicates the update process. Last but not least, npm's²² automatic security audit that checks package dependencies for security vulnerabilities cannot be executed for dependencies that are added manually.

Lauinger et al. analyzed in their 2018 released study 133 k websites and were able to show that 37% of them contained security vulnerabilities, due to outdated JavaScript libraries. They concluded that thorough approaches for dependency management and third-party code inclusion are crucial as a countermeasure [13]. As described, the existing build architecture of the Schul-Cloud did not comply with these findings at all.

For the mentioned reasons, it was decided to resolve this inconvenience.

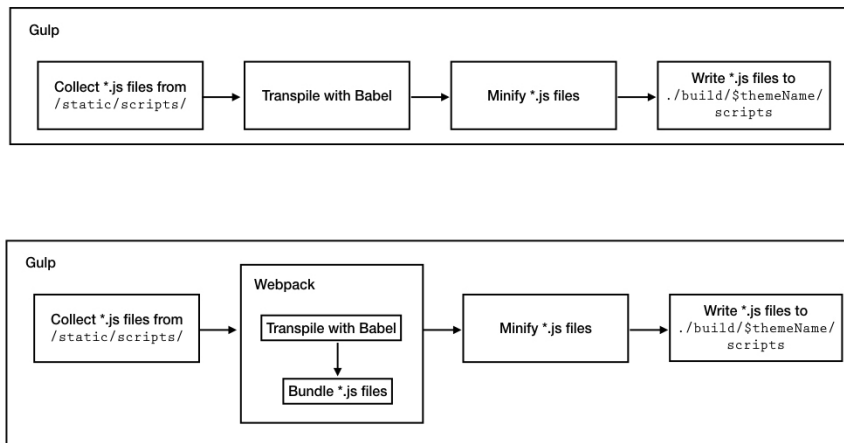


Figure 20: Comparison of build pipeline changes

OPTIMIZATION The static module bundler *webpack*²³ was added to the build pipeline as visualized in Figure 20 to improve the third-party dependency management and enable a more modular programming style. However, it also introduces some new difficulties.

In the development process, it is a wise optimization to solely rebuild files that were actually changed - especially for large projects. Before the adjustment, *gulp.watch* and *gulp-changed-smart* made sure of

²² NPM (Node Package Manager) is a packet manager for JavaScript <https://www.npmjs.com/>

²³ <https://webpack.js.org/>

that. While gulp executed a task, whenever a file in a defined path changed, the changed-smart plugin would take care of solely rebuilding files that actually changed. However, after including a bundling process, this is not sufficient anymore. Webpack builds a dependency tree to determine how to bundle files together. Given the example that `main.js` imports a function from `helper.js`: If `helper.js` changes, `main.js` needs to be rebuilt as well. This knowledge requires awareness of the dependency tree that gulp cannot have. The only available option is to rebuild everything, whenever some file changes. However, that results in a lot of potential unnecessary rebuilds. One build took about 20 seconds²⁴ to complete which is not a satisfying situation for an effective development process.

CHANGED-ENTRYPOINTS-WEBPACK-PLUGIN In order to optimize the above solution, the build time can be reduced by determining and rebuilding solely files that actually require a rebuild. This information can only be extracted in the webpack build process.

Thus, a webpack plugin was written that would follow this idea. By hooking into the build process and analyzing the files, rebuilds of unnecessary files can be prevented²⁵.

This was realized by introducing a caching mechanism for entry points²⁶. In a first build, the plugin tries to load an existing cache. If no caching information is available, a new cache is populated by iterating over all entry points and saving the file modification date of all their respective dependencies. Code example 6 shows this part: In line nine the iteration over every specified entry points is realized. Line 12 and 13 take care of saving the file modification date for every dependency of the current entry point. A schematic display of the process is depicted in Figure 21.

²⁴ Measured on a MacBook Air with 1,4 GHz Intel Core i5

²⁵ <https://github.com/georgberez/rebuild-changed-entrypoints-webpack-plugin>

²⁶ A file is labeled as an entry point when it serves as a root of a dependency tree.

```

1  async generateCache(entryPointInformation) {
2      let cacheJSON = {};
3
4      if (this.cache) {
5          cacheJSON = { ...this.cache };
6      }
7
8      await Promise.all(
9          entryPointInformation.map(
10             async ({ name, dependencySet }) => {
11                 let dependencyCache = {};
12                 await Promise.all(
13                     Array.from(dependencySet).map(async filePath => {
14                         dependencyCache[filePath] = await
15                             this.getAsyncMTimeForFilePath(
16                                 filePath
17                             );
18                     })
19                 );
20                 cacheJSON[name] = dependencyCache;
21             }
22         )
23     );
24
25     return cacheJSON;
26 }

```

Listing 6: Implementation of cache generation

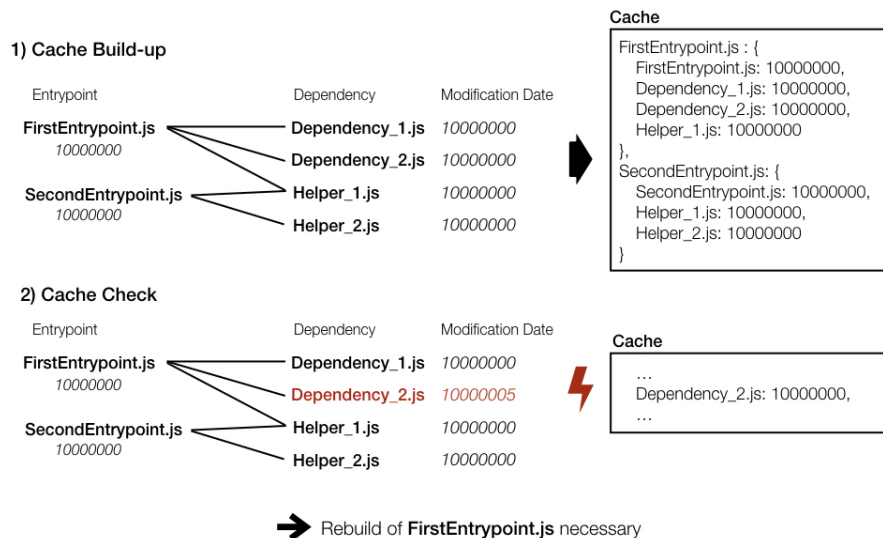


Figure 21: Description of cache functionality of Webpack plugin

Finally, the new cache information can be persisted as a JSON file to the hard drive.

Next time the cache is loaded the modification dates of files can be compared against the cached values. In case there are no changes, it is not necessary to rebuild the entry point. However, if at least one dependency changed for the respective entry point, it needs to be rebuilt, and the cache has to be updated, which is described in step two of [Figure 21](#).

By introducing the plugin the build time could be reduced to 1-2 seconds when modifying files for a single entry point²⁷.

5.2.2 Schul-Cloud-Client

This subsection documents the changes made to the Schul-Cloud client. At first, the integration concept for React components is described, followed by a paragraph concerning the styling and usage of custom components according to the concept developed in [Section 4.3](#). After that, bridge components are introduced which function as a wrapper for the library views. Lastly, a necessary change to the existing course creation view is specified.

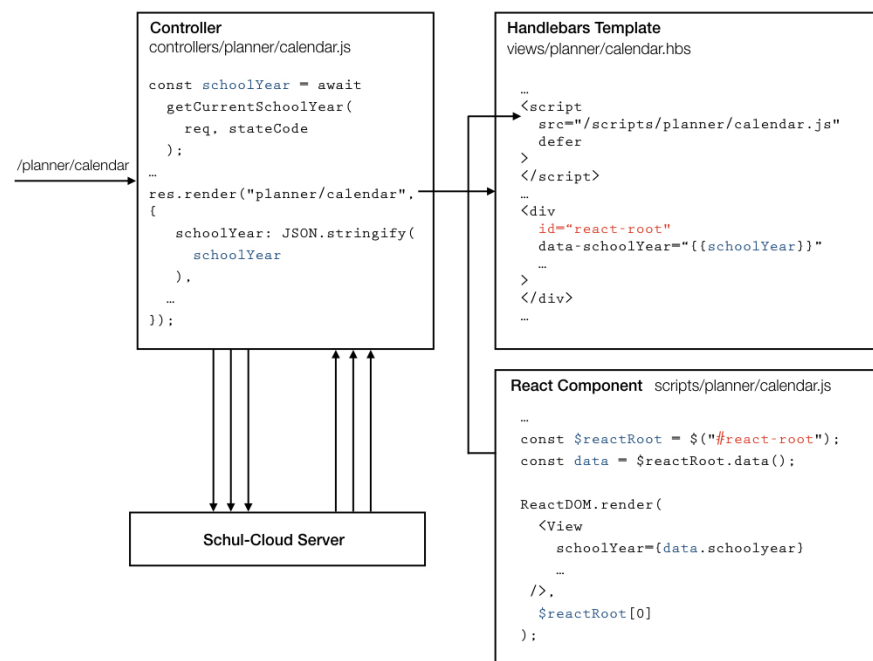


Figure 22: Rendering of React components in the Schul-Cloud

²⁷ Measured on a MacBook Air with 1,4 GHz Intel Core i5

INTEGRATION CONCEPT Considering the findings mentioned in [Section 2.3](#), the library was integrated into the Schul-Cloud respecting the existing multi-page application architecture with controllers and handlebar templates.

Following this concept, when a view that contains a React component from the library is requested, first of all, a respective controller is activated. In the example visualized in [Figure 22](#), a browser requests `/planner/calendar`. Consequently, it is the calendar controller that has to answer the request.

Therefore, it starts to aggregate the data needed to be supplied to the component. Besides other properties, a `schoolYear` property has to be passed as shown in [Section 5.1.2](#). Thus, the required data is asynchronously queried from the Schul-Cloud server, and the awaited response is transformed to satisfy the constraints of the calendar view component interface as described in [Section 5.1.2](#). Lastly, the controller passes the `schoolYear` object as well as other preprocessed data properties to the handlebars template.

It is to note that all objects have to be converted into strings. The reason can be seen in the handlebar template in [Figure 22](#). In contrast to the regular process, whereas the properties are used to generate markup directly, the data is solely supplied as custom data attributes to a div. According to the standard, these values have to be strings²⁸. An example of the resulting markup that is at last sent to the client is shown in [Figure 23](#).

The `calendar.hbs` file does also contain a script tag that references the

```
<div id="react-root" data-schoolyear="{"utcStartDate":1534723200000,"utcEndDate":1560902400000}"
data-utctoday="1543427160000" data-classtopicsdata="{"className":"10a","classes":[{"subjectId":
"5beb4c59f745b16c3630dd2a","subjectName":"Bildende Kunst/Kunst","topics":[{"id":
"5bfc92fda77e1945117f7e03","text":"Rembrandt","color":"#92DB92","utcStartDate":
1541376000000,"utcEndDate":1545609599999},{"id":"5bfc92fda77e1945117f7e02","text":
"Rembrandt","color":"#92DB92","utcStartDate":1534723200000,"utcEndDate":1540166399999}]},
{"subjectId":"5beb4c59f745b16c3630dd2b","subjectName":"Chemie","topics":[]}],"className":
"Blubsklasse 1/2","classes":[{"subjectId":"5beb4c59f745b16c3630dd2a","subjectName":"Bildende Kunst/
Kunst","topics":[]}]}]" data-holidaysdata="{"name":"Herbstferien","color":"#FBFFCF","utcStartDate":
1540166400000,"utcEndDate":1541203200000,"year":2018,"stateCode":"BB"},{"name":
"Weihnachtsferien","color":"#FBFFCF","utcStartDate":1545350400000,"utcEndDate":1546732800000,"year":
2018,"stateCode":"BB"},{"name":"Winterferien","color":"#FBFFCF","utcStartDate":
1549238400000,"utcEndDate":1549756800000,"year":2019,"stateCode":"BB"},{"name":
"Osterferien","color":"#FBFFCF","utcStartDate":1555286400000,"utcEndDate":1556323200000,"year":
2019,"stateCode":"BB"}}" data-othereventsdata="{"name":"Projektwoche","color":
"#e9e8e8","utcStartDate":1548633600000,"utcEndDate":1548979200000}">
```

Figure 23: Data attributes in the react-root div element

`calendar.js` file. Hence, when the browser received the document and finished parsing the markup, the `calendar.js` script is executed. As can be seen in [Figure 22](#), this has the effect that the information from the data attributes is read, and supplied to the React view component. At last, the React [DOM](#) is rendered into the div with the id `"react-root"`.

²⁸ <https://www.w3.org/TR/2011/WD-html5-20110525/elements.html#dom-dataset>, Accessed: 28.11.2018

Another way this could have been realized, is not to pass the properties to the React component via data attributes, but to let the component itself query the data on the initial render via AJAX requests. The former solution, however, was preferred for several reasons. First, it fits the existing architecture better, since most of the logic lies in the controller on the server side.

Secondly, it is assumed that the TTI would increase because more round trips were necessary. As in the implemented solution, the browser would request and receive the document from the Schul-Cloud client. Afterward, it would parse the markup and execute the scripts. However, only now, the component could initiate the fetching of the missing data, while for the first approach the data would have already been available because it is sent as part of the markup.

Additionally, in contrast to the Schul-Cloud client, which is in the same network as the Schul-Cloud server, the requests have to traverse the internet again, further slowing down the process. Only when the response finally arrives, the data can be fed to the React components, which will re-render and display the content.

Therefore, approach one seems to be the more effective way.

CUSTOM STYLING The development of the component library was started with regard to the in [Section 4.3](#) described styling concept that aims to provide the possibility of seamless integration. Thus, for the Schul-Cloud the library should be configured in a way so that views would respect the corporate identity and color scheme of the Schul-Cloud.

To realize that, the component provider interface for providing custom components was used. A total of twelve components were written, that fulfill the interface requirements and get passed to the component provider.

Due to the prospective change from the current Bootstrap²⁹ design to Google's material-UI, the custom components were already built in material design. For that purpose, it was relied on the react component library @material-ui³⁰ - the most popular React component library that implements the material-UI.

Consequently, 12 components were implemented³¹ that get passed to the `setupComponentMap` method of the `ComponentProvider` as shown in line 6 of code example 7. Moreover, the custom styles for font family and different text colors variants are supplied to `GeneralStylesProvider`. For convenience purpose, these two operations were aggregated in one method that gets exported.

²⁹ Bootstrap is a very popular frontend css framework that offers layouting patterns for UI elements. <https://getbootstrap.com>

³⁰ <https://material-ui.com>

³¹ This includes an `ExpansionPanel`, `Select`, `Tabs`, `Label`, `Button`, `IconButton`, `Text`, `TextField`, `TextArea`, `SelectorInput`, `TextFieldTable` and `TopicElement` component.

Hence, whenever a React component from the library is rendered, this method can be called before React starts to render, to make sure that the custom components are set up as required. This approach is visualized in line 19 and 20 of code example 7.

```

1  /////////////////////////////////////////////////// setupCustomComponents.js
2  import { Button , ... } from "../materialComponents";
3  import { setupComponentMap, setupCustomStyles } from "planner-core-lib";
4
5  export const setupMaterialComponents = () => {
6    setupComponentMap({
7      button: Button,
8      ...
9    });
10   setupCustomStyles({
11     "font-family": "Roboto, Helvetica, Arial, sans-serif",
12     ...
13   });
14 };
15
16 /////////////////////////////////////////////////// scripts/planner/calendar.js
17 import { setupMaterialComponents } from "../../helpers/planner";
18 ...
19 setupMaterialComponents();
20 ReactDOM.render(
21   ...
22 );

```

Listing 7: Setting up of the custom component map

BRIDGE COMPONENTS The documentation of the component interfaces in [Section 5.1.2](#) showed that all planner library views offer different callback functions. These enable developers to respond to events that can be emitted from components when the user interacts with them (e.g., a click on the delete or save button).

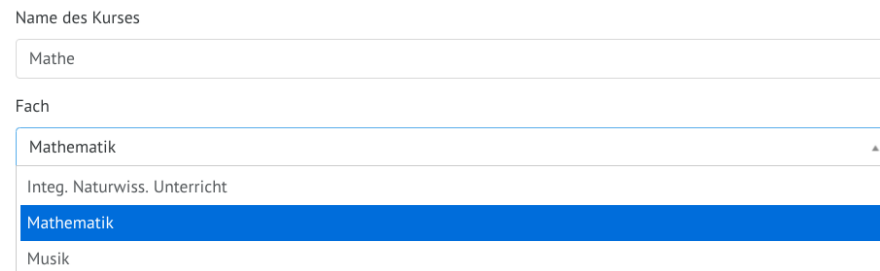
For the integration of the library into the Schul-Cloud, every view component was wrapped by a bridge component that takes care of passing callback handlers and supplying the right properties.

In most cases, AJAX-requests that persist data, or redirects to other sites are triggered when a handler is executed. Technically, this could have been realized without bridge components. However, although for this initial implementation no error states were taken care of, in the future failed requests should be used to trigger state updates and render corresponding status messages. Furthermore, the bridge component for the Class Configuration View needs to persist its state, since it needs to trigger a re-render when the AJAX requests are suc-

cessful.

Due to this use-cases and to stay consistent with the usage of React components, it was decided to implement mentioned bridge components for all views.

SUBJECT TYPE FOR COURSES The Schul-Cloud data model did not offer a possibility to assign specific subject types to courses.



The screenshot shows a web form for creating a course. It has two main sections: 'Name des Kurses' and 'Fach'. The 'Name des Kurses' section has a text input field containing the word 'Mathe'. The 'Fach' section has a dropdown menu. The dropdown is open, showing a list of subjects: 'Mathematik', 'Integ. Naturwiss. Unterricht', 'Mathematik', and 'Musik'. The first 'Mathematik' entry is highlighted with a blue background.

Figure 24: New subject-type setting for courses

The ability to differentiate courses by subjects is, however, a major precondition for a meaningful use of the planner functionality. If no information were available, templates of all subjects would have to be displayed, leading to major confusion and bad user experience. While the changes to the data model will be discussed in the following [Section 5.2.3](#), these adjustments also had to be reflected in the existing course interface of the Schul-Cloud. Thus, a new, optional setting was added to the course creation and edit view, which is shown in [Figure 24](#).

5.2.3 Schul-Cloud-Server

As described in [Section 2.3](#), the Schul-Cloud server consists of multiple microservices. The incomplete, but for this context relevant visualization of the data model and service relations can be seen in [Figure 25](#). Services and connections marked in black were already existent.

In order to be able to persist and pass the required data to the library's planner views, the architecture, however, had to be extended. A total of four new services - the holidays, topic instance, topic template, and subject type service - was introduced. Moreover, the course model was extended. [Figure 25](#) marks these adjustments by highlighting new services and connections in red.

These extensions shall be described in more detail subsequently:

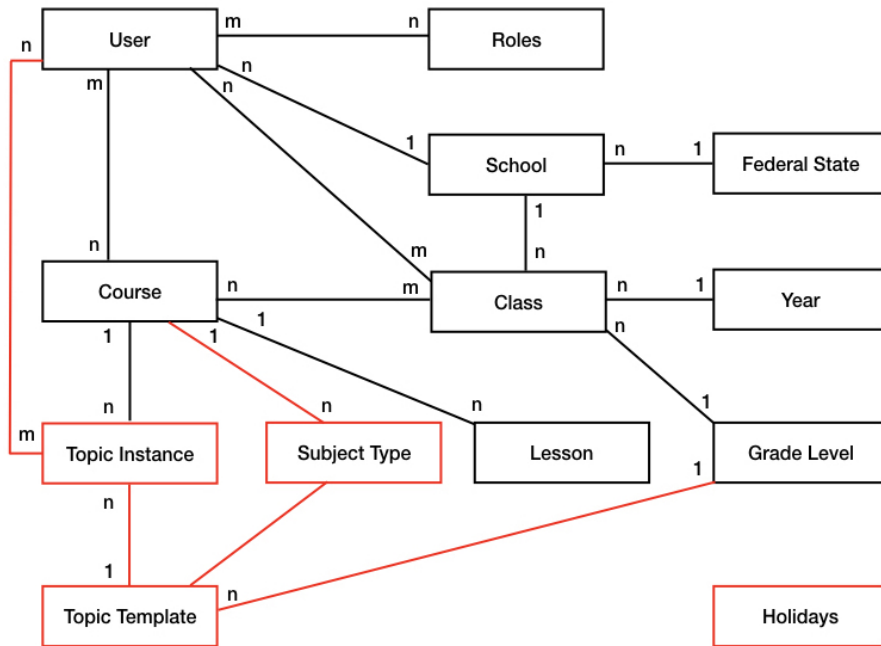


Figure 25: Excerpt of the server's relevant model relations with conducted adjustments

HOLIDAYS SERVICE As stated earlier in this chapter, information about state's holidays is necessary for the calendar view and class configuration view and as can be seen in Figure 25, these client views are effectively the only users of the new service.

Although there is a calendar service that could probably be extended to incorporate such functionality, it was decided to create a new, static service for now. The main reason was that the extension of the calendar service would have been undoubtedly a more significant effort. Thus, a basic endpoint serving static data was implemented instead. The holiday data is fed to the service on start-up from a JSON file, which was generated from a publicly available holidays API³². The service can be reached via `"/holidays"` and will return all the holidays from all the states by default. However, it accepts a state code (e.g., NI for Niedersachsen) according to the state part of ISO 3166-2:DE³³ and a year value as a query parameter. In that case, the data is filtered before it is returned.

SUBJECT TYPES SERVICE A significant flaw of the existing data model was that courses could not be assigned to specific subjects. It was possible to give courses a name reflecting the subject (e.g., "Biologie 10A"), but this unstructured data cannot be reliably analyzed and used.

Hence, a static subject types service, reachable under `"/subjectTypes,"`

³² <https://ferien-api.de>, Accessed: 27.11.2018

³³ <https://www.iso.org/obp/ui/iso:code:3166:DE>, Accessed: 27.11.2018

was introduced. It only supports GET requests and serves mappings of ids and subjects. It is supposed to represent all available subject types across all school profiles and federal states.

The data for the different subject types is based on previous non-public research of the Schul-Cloud team. Arne Oberländer created a tree-like taxonomy for federal states with their respective school types and the associated possible subject types. For this first implementation of the service a distinct set of all possible subject types, independent from the more detailed classification, was used solely. Certainly, the service could be extended to allow querying subject types for specific federal states and school types in the future.

Nonetheless, the prerequisites for using structured subject types in the data models is established with this implementation.

EXTENSION OF COURSE MODEL As described, the course model did not have any structured association to the type of a subject. Using the new subject types service, the course model was extended with an optional reference to a subject type as visualized in [Figure 25](#). Thus, a course can but does not have to have one associated subject.

Corresponding to the class model, where users can choose to use the grade level and a suffix to automatically generate the name of a class (e.g., "8a"), it seems reasonable to allow a similar automatic solution for the course name using the subject type and class name in the future (e.g., "Biology - 8a").

TOPIC TEMPLATE SERVICE To allow the persistence of topic templates, the topic template service was created. It allows creating, updating, deleting and obtaining topic templates under the route `"/topicTemplates"`.

The mongoose data schema of a topic template contains the mandatory name, number of weeks and units per week properties. Moreover, the references to a user (the owner of the template) and a subject have to be present. Optional is the reference to an associated grade level, a content string, lecture units, examinations and associated material that references the file service.

Therefore, as of now, topic templates are always defined to be associated with precisely one subject. However, they do not necessarily have to be assigned to one specific grade level.

TOPIC INSTANCE SERVICE The topic instance service, like the previous, offers an interface for the creation, deletion, modification, and obtainment of entities.

Due to topic instances being per definition an actual implementation of topic templates, the data model of both entities overlap. Either of them defines a reference to the user model, a name, the number of

weeks, units per week, content, lecture units, examinations and an array of materials containing references to files.

The difference is that topic instances are linked to a parent topic template and a course as well. Besides, a UTC start date is required. Contrariwise, no explicit reference to a subject type and a grade level is needed. Following the reference to the related course or parent template, the subject type and grade level are always indirectly defined as well.

The topic instance service functions as a regular feathers.js service which uses the mongoose adapter to define its [CRUD](#) operations. However, there is one unique hook that is activated in case of a create request to alter the normal behavior of the process. Based on the supplied parent template id in the payload of the request, the new topic instance is pre-filled with the values from its parent. Thus, the number of weeks, units per week, content, lecture units, examinations, and the material are copied from the template. In that way, the inheritance from the parent is modeled.

EVALUATION

In order to evaluate the implemented concept proposal for a school year planner in the Schul-Cloud, as it was defined in [Section 3.3.1](#), a qualitative user study was performed. A total of four teachers were given a list of tasks and questions to assess the workflow, usability and general acceptance of the web-based planning tool, followed by a debriefing.

The test process is described on the subsequent pages:

TEST PROCESS Testers were given access to a development version of the Schul-Cloud that had the planner functionality integrated. That was achieved by either testing locally in person or by providing a tunnel to the local setup via ngrok¹. In both cases, participants got a document with a sequence of instructions and questions to follow that would simulate typical use-cases. The exact instruction file used for the user study is attached in the appendix [Section A.2](#).

At first, the setting was explained. Testers were asked to imagine that they were close to a new school year and that they just got their new classes assigned by the school administration.

With that scenario in mind, the participants were instructed to log into the Schul-Cloud and navigate to the calendar view. Here, they should state the classes that they were assigned to in the present school year, the meaning of the yellow columns, and the red line. In that way, comprehension of the calendar elements should be evaluated.

Then, the testers were asked to switch to the class configuration view. At this point, they had to answer the question, what the classes and subjects were, that they had been assigned to in the last year.

Afterward, their attention was brought to the fact, that there is already a topic template “Oper Hänsel und Gretel” present from the last year. However, another template for the topic “Die Sonate” needed to be created for the current school year. Hence, the testers had to switch to the topic template creation view, where they were instructed to fill out the form and submit it. As part of that, they were asked to add a picture to the material selection of the template. Thereby, the creation of new topic templates with material should be assessed.

After completing this task, the next step was to add the newly created topic to the beginning of the current school year for the two classes 7a and 7b. Moreover, testers should extend the length of the topic

¹ <https://ngrok.com/isaservicethatoffersintrospectabletunnelstolocalhost>.

for the class 7b for one week. This should simulate that some classes need adjustment of topic length depending on external factors (e.g., cancellation of lessons) or individual properties (e.g., the class's understanding of the topic).

In the following, it was asked how many free weeks were remaining for the two classes. This question was posed to evaluate whether the display of the value in the user interface was prominent enough.

At last, the participants were asked to delete a useless topic instance and topic template that was already pre-configured. Afterward, they should save the current state and end the user study by switching back to the calendar view.

A verbal debriefing followed the completion of the tasks. The teachers were asked to assess the solution in comparison to their current style of planning. They should state what they prefer and why. Thus, the advantages and disadvantages of the proposal, as well as the existing method were evaluated. Moreover, the general comprehension and usability were discussed.

RESULTS The results of the evaluation can be divided into two sections.

On the one hand, there is the usability of the planner views: How well does a view communicate its available features to the user? Does the provided functionality behave in a comprehensible way? Does the interface rely on existing standards and pre-taught knowledge of users?

On the other hand, there is the question of the reception of the concept proposal: What benefits do teachers expect? What are the concerns? Would teachers instead use the proposed platform or stick with their current process?

Usability The testing scenario led to the identification of several strong and weak points of the current design.

All participants were able to recognize what classes were assigned to them for the current school year. That the yellow columns stand for holidays was also understood by most testers. One was unsure at first and assumed they might be exam dates. However, he corrected himself shortly later on. The red line being the current date was only correctly guessed by one participant. Others assumed it might be the middle of the school year or the end of the grading period².

To answer the question asking for the assigned classes of last school year, the testers had to change the view by selecting the former school year. Everyone was able to do so and could consequently state this information. The creation of topic templates was no hindrance too. No

² As of the date of the tests, at end of November, the line was almost in the middle of the calendar.

participant had problems to fill out the form, add a file and, finally, create the template.

In the next task - the assignment of topics to classes - all testers intuitively understood that they could use drag and drop to attach topics to classes.

Afterward, the participants were asked to determine the amount of the class's remaining free weeks for the whole school year. Although there is a label that shows this information, most testers overlooked it and just counted the free columns to answer the question. Only one person used the available display of the value.

In the following, all participants were able to delete a topic instance successfully. While two testers dragged the element onto the trash icon to delete it, one person navigated to the edit screen of the instance by hovering over the element and pressing the edit icon to switch to the topic instance view and, at last, delete the element.

Furthermore, the testing showed that all users adjusted the length of a topic via the form. No-one tried to drag on the left or right side of the topic elements to adjust the size.

The context menu, which would open when hovering over a topic template or instance for a longer time, proved to be a major source of confusion for most of the testers. Albeit, they opened the menu sometimes they could not understand that the hover was, in fact, the reason. Instead, they tried clicking the element to open the menu again.

Another point worth mentioning is that topic instances that get created from templates need to be saved before they can be edited. Thus, solely a disk icon is shown at first. However, after the disk icon is clicked and the instance got saved, the displayed icon changes to the pen icon. This behavior was unclear to testers.

To conclude the findings, most of the problems uncovered can be rated as minor. Although they complicate the user flow, they do not prevent the users from accomplishing their major use cases. Solely, the unclear opening of the context menu for topic templates and instances is to be rated as a significant problem. That is because it is fundamental that users can edit and delete elements. If users are not able to reliably use these functions, they cannot deal with some common use cases.

Despite this, users showed general satisfaction with the interface and usability. One tester stated that "one knows these functions [and icons]" from other platforms, which makes the user experience very fluid and self-explanatory. Another person stated that "everything was intuitive."

Conceptual The feedback from the participants of the user study concerning the concept proposal was generally positive as well. It was stated that "the planning is very pretty," the "overview makes

total sense" and that "the structure is logical." The possibility to configure topic blocks in the way it is offered "is how teachers do it" and resembles the teacher's current process very well.

In comparison to the typical paper or document-based planning approach, the web-based planning tool offers a better overview of the school year. In that regard, one tester stated after the test that "[she] currently [does] [the planning] in a table in [Microsoft] Word and [the planner] is much nicer." Another teacher explained that so far "[she] [has] circa 5000 folders on [her] laptop that are all called lesson preparation" and "it is incredibly annoying to do [the planning] in multiple single documents." This is why she thinks that the web-based tool would work great.

Moreover, teachers estimated that time is likely to be saved when templates from former years can be just copied to new classes. The vision of being able to share material associated with a topic (e.g., worksheets) with students from the course with one click and without having to print it, was regarded to hold enormous potential.

A point worth improving, however, could be the current subclassification of topic instances. One teacher mentioned that he was not sure if a lesson unit would depict a single lesson or a block of lessons. However, he found it quite necessary to know, in order to be able to plan the content of a lesson unit. Thus, he would like to have the information or the possibility to configure how many lessons a unit is made of.

Moreover, the same tester reported that he needs to be able to plan units in more detail for the visitations of his classes. Lesson units can be more specified in a single text field solely. However, he needs to plan his lessons on a minute-wise basis. Therefore, a single text field would yield poor support for this requirement. A functionality as suggested by the PLATON system in [Section 2.1](#) would fit the need. However, it is to say that the tester is still a trainee teacher. As it was explained in [Section 3.2](#) in more detail, this user group has slightly different requirements and is not a representative group for all teachers.

One user study participant mentioned the wish to be able to specify certain content of a topic template to be automatically visible for her students. Still, general information content should solely be visible for the teacher.

In total, only two greater concerns were mentioned that would argue against the adoption of such a platform.

The first of the concerns was data and privacy. One teacher stated that she would be afraid that "everything that is mine will be public." Her teaching material and lesson plans are the core component of her teaching style and work. Hence, they define a very personal and intimate space. It is crucial to her that nobody but herself has access to it. In that sense, she mentioned the concern that the school

administration or the developers of the platform could have access to it. In that regard, it was also stated that if a colleague shares material or tasks with her and tells her to use it for herself solely, she wants to comply with this request.

Secondly, another key concern that would need to be cleared out to support the adoption is the existence of double structures. A tester stated that he "definitely [does] not want to maintain two structures." Therefore, the web-based planner must offer the flexibility and features to be able to replace the local structures completely. An example of a process that is currently not modeled by the application is his Internet research. When he has a certain lesson and topic coming up in the following days, he will search for images, text, and charts on the Internet to find suitable, up-to-date material. Since "there is no [Microsoft] Word, where I could paste it" on the planner platform, he would follow his current process. Thus, he pastes the found content into a local Word document, in order to modify and edit it. After doing so, he would need to upload this file to the planner app. This, however, is extra effort, which does not seem to yield any value, if the file is already locally available.

Despite the described concerns and additional ideas, the user study participants stated that they were very willing to test the concept for a more extended period. Especially the requirements for the rough planning of a school year are fulfilled by the proposal. Regarding a wholesome adoption, including the exact learning matter plan, it would have to make sure that no double structures are needed, and that data and privacy are secured. Moreover, a higher value proposition with more features and functionality could foster the adoption.

FUTURE WORK AND CONCLUSION

Following the evaluation results, as well as other less pursued directions that were touched in the course of this thesis, future steps shall be proposed. This includes high-level usability improvements for the library views and general recommendation on how to iterate on the planner proposal and the Schul-Cloud integration.

USABILITY IMPROVEMENTS

- The red line was not recognized as the current date:
A noticeable improvement to increase the user's understanding would be to add the current date above the red line in the same color. In that way, the connection of line and date can be better visually communicated.
- Yellow columns were not directly recognized as holidays:
Although most participants of the user study understood the meaning of the yellow columns, one person had initial difficulties. It could be worth exploring to add suitable icons to facilitate a faster visual association and improve the understanding for the holiday types (e.g., a sun icon for summer and a leaf icon for autumn).
- The amount of the class's remaining free weeks was not seen:
Solely one tester was able to use the displayed number when asked to state the amount of remaining free weeks. However, this is not crucial information. Making too many aspects of a screen eye-catching can clutter the view for the actual vital functions. It is acceptable and desired to have more experienced users learn non-critical features later.
- The length was not adjusted by dragging on the sides of topic elements:
The user tests showed that this is a feature that has to be taught. It is not intuitively clear that a topic element can be resized by dragging on the left or the right border. Hence, the functionality has to be suggested more explicit. It seems reasonable to display a left and right arrow beneath an element. However, they should not be shown all the time but when the user hovers over a topic. Thus, they would not clutter the view and appear solely when the user wants to interact with it. If he would hover

over an arrow, the cursor icon could change to an arrow too to communicate the different behavior when dragging now.

- The context menu could not be opened:
To open the context menu is a crucial functionality because it is required for main user flows. Most testers tried to open it by clicking on the element. Therefore, this is an option that should be offered. Nonetheless, the hover alternative should be available as well. Especially for key use cases, it is advisable to offer different possibilities to cover most of the individual usage differences. Still, the time until the menu appears could be reduced to reflect the test results.
- The context menu sometimes shows a save-icon, sometimes an edit-icon:
Topic instances cannot be edited before they are not persisted on the server. Hence, only a save-icon is displayed after the dropping a template into a class. After saving it, other icons are shown. This behavior has mainly a technical reason and is confusing for users. The most straight-forward way to improve it would be to add a small explanation text next to the save-icon that expresses that newly created instances have to be saved before they can be edited. Another probably more sustainable and user-friendly idea would be to add automatic saving of the state. Thus, no save-icon would be needed in the first place.

PROPOSAL

- Material Search
As described in the evaluation in [Chapter 6](#), the process of searching for current material for an upcoming class is not sufficiently modeled by the current proposal. Teachers want to be able to paste pictures, charts, texts into documents where they can edit the content based on their requirements and, at last, use it for their class.
It could be explored whether letting teachers directly create and add text documents to a topic template or instance in the web interface could be sufficient. Moreover, they have to have the possibility to edit these documents, including pasting images or texts directly. In that way, it should be prevented that teachers have to have double structures. The whole planning aspect should be manageable via the web-based tool.
- Public holidays and actual timetables
Currently, solely the state's holidays are incorporated into the planner concept. However, public holidays are also fixed dates and could be automatically included based on the state of the school.

Now, if the data of the exact timetable of classes were integrated with the planner as well, the actual number of remaining lessons could be calculated. As of now, only the number of remaining weeks can be determined. Moreover, every generally planned lesson unit could be assigned to a specific date and lesson, which would give teachers more confidence in their plannings.

- Sharing of whole topic templates

Topic templates are generally speaking a specification of the content and material for a topic block. They can contain worksheets, more detailed lesson plans and instructions on how to convey competences to the students. While the Schul-Cloud supports the sharing of general material, it seems worthy to explore the possibility to share whole topic templates. Teachers could be given the possibility to make their templates publicly available - either for teachers of the same school or for teachers of other schools as well.

In the long term, also third-party stakeholders like school book publishers could, for example, release and offer templates based on their school books.

- Lesson units - Integration of editr.io

The proposal provides only a simple text field as a description element for lesson units. However, the evaluation showed that this might not be sufficient for some teachers. Offering a text area to allow for more extensive specifications if needed could be helpful.

In that regard, it could also be explored, whether the material should be assigned to specific lessons directly. Currently, editr.io - a rich text editor for creating content in the Schul-Cloud - is being developed¹. Teachers could be allowed to create and assign editr.io worksheets to specific lesson units in topic templates. As with the rest of the topic template materials, they could be copied and used when creating instances of this template.

- Deep integration of competencies from the core curriculum

The German school system evolves into a competency-based learning strategy. In the SCHIC described in Section 1.2 teachers have to state the specific competencies from the state's core curriculum that they will convey to the students in a specific topic block.

Integration of the core curriculum into the planner platform could allow the assignment of competencies to topic templates - or even lesson units. In connection with the availability of class data in the Schul-Cloud (e.g., the members of a class), individual learning profiles could be generated that were able to state

¹ <https://github.com/schul-cloud/editr.io> - Accessed: 03.12.2018

the exact competencies a student would have been taught in his school life. The vision could go so far, as to let teachers see what kind of knowledge was already imparted to a class, they got assigned to because the current state of the teachings would be automatically documented.

The success of the German education system is highly dependent on the ability to develop a highly-qualified workforce. The adoption and acceptance of the digitalization in schools is a major driver to help with that. However, it is not only the education that can be optimized but also the digitalization of whole school related processes.

This thesis showed that a web-based planning tool has the potential to improve the existing school year planning process of German teachers. All testers praised the approach and saw great potential in the software solution.

The next step would be to iterate on the usability flaws that were uncovered during the user research. Afterward, an empirical user study should be conducted to collect more broad feedback and verify these initial findings.

LIST OF FIGURES

Figure 1	Exemplary curriculum implementation of part C	6
Figure 2	The PLATON lesson planning platform	9
Figure 3	Google Classroom dashboard	11
Figure 4	Overview of Schul-Cloud architecture	13
Figure 5	Schul-Cloud client architecture	14
Figure 6	Exemplary learning matter distribution plan .	18
Figure 7	Topic template view draft	22
Figure 8	Class configuration view draft	23
Figure 9	Calendar overview draft	24
Figure 10	Comparison of view styling	37
Figure 11	Usage of component composability	40
Figure 12	Exemplary Typescript error	41
Figure 13	Interface of class configuration view	44
Figure 14	Visualization of the drag and drop components	47
Figure 15	Interface of topic template view in NEW mode	51
Figure 16	Interface of topic instance view	54
Figure 17	Interface of calendar view - school year	55
Figure 18	Interface of calendar view - three months . . .	56
Figure 19	Interface of calendar view - two weeks	56
Figure 20	Comparison of build pipeline changes	59
Figure 21	Description of cache functionality of Webpack plugin	61
Figure 22	Rendering of React components in the Schul- Cloud	62
Figure 23	Data attributes in the react-root div element .	63
Figure 24	New subject-type setting for courses	66
Figure 25	Excerpt of the server's relevant model relations with conducted adjustments	67

LIST OF LISTINGS

Figure 1	Style definition for the three-step CSS adjust- ment concept	32
Figure 2	Style provider implementation and usage . . .	33
Figure 3	Usage of component provider	35
Figure 4	Partial implementation of component provider	36

Figure 5	Styled components example	42
Figure 6	Implementation of cache generation	61
Figure 7	Setting up of the custom component map . . .	65
Figure 8	AllClassInstances<TopicIndexType> type interface	83
Figure 9	AllTopicTemplates type interface	84
Figure 10	SchoolYear type interface	84
Figure 11	AllClassInstancesType<LocalTopicIndexType>	84
Figure 12	Topic Instance - FormValues type interface . .	85
Figure 13	FormValuesOptions type interface	85
Figure 14	OnFileAdd type interface	86
Figure 15	File type interface	86
Figure 16	TopicInstance - FormValues type interface . .	86
Figure 17	CurrentFormState type interface	87
Figure 18	ClassTopicsData type interface	87
Figure 19	Event type interface	87
Figure 20	TopicTemplateView type interface	88

APPENDIX

A.1 TYPE INTERFACES

```

1 export type AllClassInstancesType<TopicIndexType> = {
2   [schoolYearId: string]: {
3     schoolYearId: string;
4     schoolYearName: string;
5     subjects: {
6       [subjectId: string]: {
7         subjectId: string;
8         subjectName: string;
9         classLevels: {
10          [classLevelId: string]: {
11            classLevelId: string;
12            classLevelName: string;
13            classes: {
14              [classId: string]: {
15                id: string;
16                name: string;
17                topics: {
18                  id: string;
19                  text: string;
20                  color: string;
21                  startIndex: number;
22                  endIndex: number;
23                }[];
24              };
25            };
26          };
27        };
28      };
29    };
30  };
31 };

```

Listing 8: AllClassInstances<TopicIndexType> type interface

```

1  export type AllTopicTemplatesType = {
2    [subjectId: string]: {
3      [classLevelId: string]: {
4        id: string;
5        text: string;
6        width: number;
7        color: string;
8      }[];
9    };
10 };

```

Listing 9: AllTopicTemplates type interface

```

1  export type SchoolYearType = {
2    utcStartDate: number; // first day of school
3    utcEndDate: number; // last day of school
4  };

```

Listing 10: SchoolYear type interface

```

1  export type AllClassInstancesType<LocalTopicIndexType> = {
2    [schoolYearId: string]: {
3      schoolYearId: string;
4      schoolYearName: string;
5      subjects: {
6        [subjectId: string]: {
7          subjectId: string;
8          subjectName: string;
9          classLevels: {
10             [classLevelId: string]: {
11               classLevelId: string;
12               classLevelName: string;
13               classes: {
14                 [classId: string]: {
15                   id: string;
16                   name: string;
17                   topics: {
18                     id: string;
19                     text: string;
20                     color: string;
21                     startIndex: number;
22                     endIndex: number;
23                     isLocal?: boolean;
24                     parentTemplateId?: string;
25                   }[];
26                 }[];
27               };
28             };
29           };
30         };
31       };
32     };
33 };

```

Listing 11: AllClassInstancesType<LocalTopicIndexType>

```
1 export type FormValuesType = {  
2   subjectId: string;  
3   classLevelId?: string;  
4   name: string;  
5   numberOfWeeks: string;  
6   unitsPerWeek: string;  
7   content?: string;  
8   subjectUnits?: string[];  
9   examinations?: ItemType[];  
10  material?: FileType[];  
11 };
```

Listing 12: Topic Instance - FormValues type interface

```
1 type IdTextType = {  
2   id: string;  
3   text: string;  
4 };  
5  
6 export type FormValuesOptionsType = {  
7   subject: IdTextType[];  
8   classLevel: IdTextType[];  
9 };
```

Listing 13: FormValuesOptions type interface

```

1 export type OnFileAddType = (
2 {
3   file,
4   onComplete,
5   onError
6 }: {
7   file: {
8     file: File;
9     tempId: string;
10  };
11   onComplete: (file: FileType, tempId: string) => void;
12   onError: (tempId: string) => void;
13 }
14 ) => void;

```

Listing 14: OnFileAdd type interface

```

1 export type FileType = {
2   file: string;
3   name: string;
4   type: string;
5   id: string;
6 };

```

Listing 15: File type interface

```

1 type ItemType = {
2   typeValue: string;
3   timeValue: string;
4   textValue: string;
5 };
6 type FileType = {
7   file: string;
8   name: string;
9   type: string;
10  id: string;
11 };
12
13 export type FormValuesType = {
14   subject: string;
15   classLevel: string;
16   name: string;
17   parentTemplate: {
18     id: string;
19     name: string;
20   };
21   numberOfWeeks: string;
22   unitsPerWeek: string;
23   content: string;
24   subjectUnits: string[];
25   examinations: ItemType[];
26   material: FileType[];
27 };

```

Listing 16: TopicInstance - FormValues type interface

```
1 export type CurrentFormStateType = Omit<FormValuesType, 'parentTemplate'>;
```

Listing 17: CurrentFormState type interface

```
1 type TopicDateType = {  
2   utcStartDate: number;  
3   utcEndDate: number;  
4   id: string;  
5   text: string;  
6   color: string;  
7 };  
8  
9 export type ClassTopicsDataType = {  
10  className: string;  
11  classes: {  
12    subjectId: string;  
13    subjectName: string;  
14    topics: TopicDateType[];  
15  }[];  
16 }[];
```

Listing 18: ClassTopicsData type interface

```
1 export type EventType = {  
2   name: string;  
3   color?: string;  
4   utcStartDate: number;  
5   utcEndDate: number;  
6 }[];
```

Listing 19: Event type interface

```
1 export type PropsType =
2 | {
3   mode: 'EDIT';
4   initialValues: FormValuesType;
5   valueOptions: FormValuesOptionsType;
6   onFileClick: (file: FileType) => void;
7   onFileAdd: OnFileAddType;
8   onFileRemove: (file: FileType) => void;
9   onCreate?: (values: FormValuesType) => void;
10  onSave: (values: FormValuesType) => void;
11  onDelete: () => void;
12 }
13 | {
14   mode: 'NEW';
15   initialValues?: FormValuesType;
16   valueOptions: FormValuesOptionsType;
17   onFileClick: (file: FileType) => void;
18   onFileAdd: OnFileAddType;
19   onFileRemove: (file: FileType) => void;
20   onCreate: (values: FormValuesType) => void;
21   onSave?: (values: FormValuesType) => void;
22   onDelete?: () => void;
23 };
```

Listing 20: TopicTemplateView type interface

A.2 EVALUATION TEST FORM

Test - Schuljahresplaner in Schul-Cloud

Wir befinden uns am Anfang des Schuljahres. Du hast deine Klassen von der Schulleitung zugewiesen bekommen. Deine Aufgabe ist es nun dein Schuljahr vorzuplanen, um die Zeitzuteilung der Themen einschätzen zu können.

1. Logge dich in die Schul-Cloud ein (Email: lehrer@schul-cloud.org Passwort: schulcloud)
2. Wechsle auf "Schuljahresplaner"

Frage: Welche Klassen hast du dieses Jahr zugeteilt bekommen?

Frage: Was bedeuten die gelb/orangen Spalten?

Frage: Was bedeutet der rote Balken?

- Wechsle nun auf "Schuljahresplaner" -> "Meine Klassen".

Frage: Welche Klassen waren dir im letzten Schuljahr zugeteilt?

- Im letzten Jahr hast du bereits die Themenvorlage "Oper Hänsel und Gretel" für die Klassenstufe 7 erstellt. Allerdings werden für dieses Schuljahr weitere Themenvorlagen benötigt.
- Erstelle eine weitere Themenvorlage "Die Sonate".
 - Sie soll 5 Wochen dauern.
 - In der 1. Unterrichtseinheit der 5. Woche soll eine Kurzarbeit zur "Mondscheinsonate" geschrieben werden.
 - Füge außerdem das Bild "Orchester.jpg" zu der Themenvorlage hinzu (das Bild befindet sich im .
 - Speichere die Vorlage.
- Füge das Thema "Die Sonate" an den Anfang des aktuellen Schuljahres für die Klasse 7a und Klasse 7b hinzu.
- Du weißt, dass Klasse 7b traditionell etwas langsamer ist, als Klasse 7a. Verlängere das Themengebiet "Die Sonate" für diese Klasse daher um eine Woche.
- Füge das Themengebiet "Oper Hänsel und Gretel" für beide Klassen direkt vor den Weihnachtsferien ein.

Frage: Wie viele freie Wochen sind im aktuellen Schuljahr für die jeweiligen Klassen 7a und 7b noch vorhanden?

- Speichere den aktuellen Stand.
- Entferne das Thema "Blub" von der Klasse 7b.
- Tatsächlich macht das Thema "Blub" nicht wirklich Sinn. Lösche die Vorlage komplett.
- Speichere erneut.
- Wechsle zurück zu Schuljahresplaner -> "Kalender"
- Du bist fertig :)

BIBLIOGRAPHY

- [1] K. R. Apt. "Logic Programming." In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)* 1990 (1990), pp. 493–574.
- [2] A. Berg. "Kinder und Jugend in der digitalen Welt". In: Online: <https://www.bitkom.org/Presse/Anhaenge-an-PIs/2017/05-Mai/170512-Bitkom-PK-Kinder-und-Jugend-2017.pdf> (08.08. 2017) (2017).
- [3] R. Bertenrath, L. Bayer, M. Fritsch, B. Placke, E. Schmitz, and P. Schützdecker. "Digitalisierung in Bildungseinrichtungen". In: (2018).
- [4] BMBF. *Neuer Kooperationspartner für die Schul-Cloud*. <https://www.bmbf.de/de/neuer-kooperationspartner-fuer-die-schulcloud-5682.html>. Pressemitteilung. 2018.
- [5] W. Bos, B. Eickelmann, J. Gerick, F. Goldhammer, H. Schaumburg, K. Schwippert, M. Senkbeil, R. Schulz-Zander, and H. Wendt. "ICILS 2013". In: *Computer-und informationsbezogene Kompetenzen von Schülerinnen und Schülern in der 8* (2014).
- [6] A. Burstedde, G. Kolev, J. Matthes, et al. *Wachstumsbremse Fachkräfteengpässe*. Tech. rep. Institut der deutschen Wirtschaft Köln (IW)/Cologne Institute for Economic Research, 2018.
- [7] P. von Campenhausen, N. Düppe, B. Jankofsky, A. Knäring, B. Kölle, D. K. Meyr, M. Nagel, J. Schäfer, and C. Schminder. "Das ABC des schulinternen Curriculums". In: (2016).
- [8] S. Cavanagh. "Amazon, Apple, Google, and Microsoft battle for K-12 market, and loyalties of educators". In: *Edweek Market Brief* (2017).
- [9] C. French-Owen. "The Deep Roots of Javascript Fatigue". In: *Segment Blog* (2016).
- [10] T. Hardwig and F. Mußmann. "Zeiterfassungsstudien zur Arbeitszeit von Lehrkräften in Deutschland". In: *Konzepte, Methoden und Ergebnisse von Studien zu Arbeitszeiten und Arbeitsverteilung im historischen Vergleich. Expertise im Auftrag der Max-Träger-Stiftung, Göttingen: Kooperationsstelle Hochschulen und Gewerkschaften* (2018).
- [11] B. Herzig. "Wie wirksam sind digitale Medien im Unterricht". In: *Gütersloh: Bertelsmann Stiftung* (2014).
- [12] A. Klaus. *Eine Lernplattform für alle*. <https://www.telekom-stiftung.de/themen/eine-lernplattform-fuer-alle>. Blog. 2016.

- [13] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda. "Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web". In: *arXiv preprint arXiv:1811.00918* (2018).
- [14] C. Meinel, J. Renz, C. Grella, N. Karn, and C. Hagedorn. *Die Cloud für Schulen in Deutschland: Konzept und Pilotierung der Schul-Cloud*. Vol. 116. Universitätsverlag Potsdam, 2017.
- [15] A. Osmani. *The cost of Javascript in 2018*. <https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4>. Blog. 2018.
- [16] A. Prognos. *Arbeitslandschaft 2040. Eine vbw-Studie, erstellt von der Prognos AG*. 2015.
- [17] M. R. Raphaël Benitte Sacha Greif. *The State of JavaScript 2018*. <https://2018.stateofjs.com/javascript-flavors/conclusion/>. Survey. 2018.
- [18] U. Schmid, L. Goertz, J. Behrens, L. Michel, S. Radomski, and S. Thom. "Monitor Digitale Bildung: Die Schulen im digitalen Zeitalter". In: *Stiftung, Bertelsmann* (2017), p. 64.
- [19] S. Strickroth. "Unterstützungsmöglichkeiten für die computerbasierte Planung von Unterricht". In: (2016).

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

I certify that the material contained in this thesis is my own work and does not contain unreferenced or unacknowledged material.

Potsdam, 08. December 2018

Georg Berecz