Electronic Communications of the EASST
Volume 83 Year 2025

deRSE24 - Selected Contributions of the 4th Conference for Research Software Engineering in Germany

Edited by: Jan Bernoth, Florian Goth, Anna-Lena Lamprecht and Jan Linxweiler

lea.online - Software Applications for Individuals with Low Literacy

Jan Küster

DOI: 10.14279/eceasst.v83.2623

lea.online - Software Applications for Individuals with Low Literacy

Jan Küster[1*]

[1] jkuester@uni-bremen.de
https://orcid.org/0009-0008-8088-9837
lab media & education; ZeMKI
University of Bremen, Germany

Abstract: In 2018, there were an estimated 6.2 million people between the ages of 18 and 64 with low literacy skills in Germany. The ”lea.online” project designed and implemented a software system with multiple applications in order to gain access to this otherwise closed field and to conduct further research of the underlying competency model. This work presents the principal aspects of the research and development project and the challenges involved. It highlights the relationships between the domain of basic education and software engineering. It also sheds light on issues related to publicly funded software engineering, software architecture, outsourcing, documentation, accessibility, legal implications and sustainability.

Keywords: educational technology, research software engineering, field access, low literacy, web development, mobile app development

1 Introduction

1.1 Low Literacy in German Society

In 2018, the LEO study identified approximately 6.2 million individuals aged between 18 to 64 in Germany with low literacy skills [GBD$^+$19]. These individuals are deemed to have a literacy level that restricts their participation in crucial aspects of society. According to the study, this ranges from literacy at character level to literacy at sentence level (alpha level 1 - 3)[1]. The group is referred to in this paper as the ”target group”. Another important aspect of the study is the demographic findings, which show that this group is not homogeneous. It is highly diverse in terms of gender, age, migration background, educational attainment and occupational status. Various former and recent studies[2] have also shown a similar trend for the German apprenticeship sector and for young people in general [MWWK24]. One public measure to improve this situation is the provision of literacy courses at German community colleges, also known as Volkshochschulen (VHS). However, only 0.7% of the 6.2 million people actually attend such courses, mostly motivated by the desire to improve their job situation or job prospects [GBD$^+$19]. Fear of discrimination and shame are seen as the main reasons for not attending [HM14].

* The author is sponsored on GitHub by Meteor Software and various members of the MeteorJs community for his open source contributions and community ambassador activity

1 Alpha levels are a distinct classification of competencies

2 For example VERA 8, PISA, ULME I-III, and IQB

1.2 otu.lea (2008 - 2010)

Community colleges (VHS) across various regions in Germany do provide literacy courses[3]. The assessment of the individual literacy levels (diagnostics) is crucial for the teacher in order to prepare for an upcoming literacy course with appropriate content and internal differentiation. These initial assessments were originally designed as paper-and-pencil tests, which were administered by a person who was responsible for reading the task instructions aloud. Furthermore, the ongoing documentation of the learner's progress requires a continuous alignment with the comprehensive competency model that was developed during the original "lea."[4] project (BMBF[5], until 2011). The "otu.lea"[6] project (BMBF, 2008 - 2010) attempted to fully digitize the diagnostic material and to make it available to institutions as a free online self-service [KW14]. The application was written in Adobe Flash, which was becoming increasingly outdated and a major security risk, and was therefore deprecated by Adobe itself.

1.3 lea.online (2018-2022)

The "lea.online" project (BMBF, FKZ W143600) started with the initial objective of porting otu.lea to replace Adobe Flash with modern web standards. Further objectives were to improve the diagnostic material and the competency model and to include vocational fields in the learning material. Teachers should be able to monitor and evaluate the diagnostics of their participants over time. The aforementioned LEO study found that 78% of the target group regularly used smartphones, tablets, messenger apps and social media at a similar rate to the general population [GBD+19]. This was seen as a valuable opportunity to provide them with anonymous self-learning based on the existing competency model, but independent of any institution. Therefore, it was planned that a non-intrusive, anonymous learning app would be developed for the target group.

The project took a highly interdisciplinary approach to Research Software Engineering (RSE). It comprises multiple target user groups: individuals with low literacy, literacy course teachers, as well as researchers. Additionally, the material for learning and assessment entails multiple professions (technical/industry, healthcare, food processing industry) spread across four subject dimensions (reading, writing, language understanding, and mathematics). The team consisted of professionals from various disciplines, such as Educational Sciences, Mathematics, Software Engineering, and Law. Despite their different backgrounds, the team shared the same vision of creating a system of sustainable software applications and packages by applying best practices in software engineering[7].

The lea.online architecture, technology stacks and applications are presented in more detail in Section 4. All code that was and is produced for lea.online is publicly available on GitHub [8] under free and open source licenses.

[3] See https://www.vhs-bremen.de/veranstaltungen/grundbildung for recent examples in Bremen.
[4] lea. stands for "Literalitätsentwicklung von Arbeitskräften", German for literacy education for adults; own translation
[5] Bundesministerium für Bildung und Forschung; German Federal Ministry of Education and Research
[6] Online Testumgebung für lea., German for online testing environment for lea.; own translation
[7] Many of the best practices are covered by [LPM22] and the SURESOFT project, see https://suresoft.dev/
[8] https://github.com/leaonline

2 Relation to RSE

A recent publication by Goth et al. defines research software as ”Foundational algorithms, the software itself, as well as scripts and computational workflows that were created during the research process or for a research purpose, across all domains of research.” [GAB+23, p. 5] This section aims to place the software system of lea.online and its project context within that definition in order understand it’s role within the domain of Research Software Engineering (RSE).

2.1 Research Purpose

The actual goals of the former and ongoing research are the validation and improvement of the competency model and to gain insights about the target group as software users. However, access to the target group is aggravated as there is only a fraction actively reaching out to institutions. The software applications of lea.online are created to gain access to the target group and to enable this research while providing a learning benefit to the target group and institutions.

2.2 Software Engineering

The practical parts of the project cover industry-level software engineering to deliver production-ready applications and services to a larger audience of non-professional users. It encompasses stages across the entire software life cycle, ranging from domain analysis, requirements engineering and active development to operations, maintenance, as well as end-of-life. Selected topics are covered in Section 4, Section 5 and Section 7.

2.3 Target Group

The target group represents non-expert users. This contrasts with research software, made for researchers or similar expert users. It implies a wide range of factors to consider when designing software applications. For example, there are several *accessibility* needs that are highly specific to the target group. Ensuring *anonymity* plays a key role in building trust between the target group and the application. It is regarded as a helpful measure in reducing inhibiting factors, such as shame and the fear of discrimination [Kop17, p.46]. This in turn makes it a great challenge for the *security* aspects of the software system. These topics are covered in Section 5 and Section 6.

User experience design and *software stability* must be approached with an awareness of the target group’s sensitivity to errors and its motivational implications. This is partially covered in Subsection 7.3. Finally, *paternalization* is to be avoided, considering all the assistance given to the users.

2.4 Project Context

All development took place within a publicly funded research project. It was as a collaboration between working groups of two German universities (University of Bremen and School of Education, Weingarten). This had a strong influence on the software development process. Requirements on privacy or the incompatibility with on-demand payment procedures made it difficult to rely on cloud-service vendors, especially with those outside of the European Union. The restricted and fixed financial resources, in addition to the global pandemic, resulted in a constrained pool of available developers throughout the project funding period. Budgetary allocations for development contracts were made as a potential compensation for this circumstance. The involved challenges are described in Section 7.

2.5 Related Work

The low literacy of the target group implies a strong need for accessibility and user-centered solutions, especially with respect to user interfaces and interaction design. Consequently, the majority of related work addresses topics, such as user experience (UX), design, usability and the visual aspects of accessibility, such as colors, typography, component design and layout. Related findings from the lea.online project are published by Meyer [MWWK24] [MW21]. They partially build on top of the work by Medhi et al., Belay et al. and Maciel [BMB16] [Mac13] [MPB$^+$11]. Other usability-related publications originated in the otu.lea project, such as [Kop17], and [KKW13]. Finally, there is a lack of visible publications in the field of RSE that explicitly involve the target group.

3 Methodology

This project experience report provides a comprehensive overview of the major software engineering challenges that emerged during the course of the lea.online project. Furthermore, it considers the impact of these challenges on the final project outcome. They are presented in three different parts. In Section 4, the various milestones and their respective challenges are elucidated in chronological order. In Section 5 and Section 6, the challenges are discussed in strong relation to the target group. Finally, in Section 7, the particular software engineering practices that were shaped by the context of an understaffed development team are examined from a technical perspective.

By doing so, this work reflects on a complex phenomenon, namely the development of a software system with production-ready software applications that are based on an existing educational competency model and tailored for a large audience of non-expert users. It is set in the context of a publicly funded research project, in which software is developed with limited resources and under limiting circumstances. At the same time the phenomenon and the context have no clear boundaries and it appears that the project is a viable candidate for a case study [RHRR12]. However, the author was actively involved in the project, which raises the question of bias. Moreover, observing this phenomenon as part of a planned case study, including interviews, formalized protocols and guidelines, was not considered in the project plan.

This requires a critical perspective, in order to avoid falling into the trap of claiming that a software engineering publication is a case study without fulfilling the requirements for being one [Woh21]. Nevertheless, the intention in this work is to provide the same depth and richness as a case study does, with the same objectives ”[...] to better understand how and why software engineering should be undertaken and, with this knowledge, to seek to improve the software engineering process and the resultant software products.” [RHRR12, p. 3]

4 The lea.online Software System

This section describes in chronological order the major milestones of the lea.online project, the challenges involved, and their respective results. This is followed by a description of the final software system architecture.

4.1 Domain Analysis

The software architecture, software design and the selection of the technology stack have all been influenced by a multitude of factors. First, there were the multiple use cases: diagnostics, learning analytics, learning, authentication, content editing and delivery. Secondly, multiple stakeholders were involved in one or more use cases, including the target group, teachers and lecturers, lea. core-team, student assistants and external contractors. While it was straightforward to identify clear boundaries between use cases, designing an appropriate architecture proved to be a challenge.

This led to an iterative approach with a first draft of a distributed system, comprising of multiple standalone applications. The structure of the competency model and the diagnostic material were the blueprint for many database schema models, used in most applications. Despite the clear boundaries there was a hidden coupling between the applications introduced by these models, which led to the later emergence of lea.Backend and lea.Content.

The necessity for authentication across applications was identified at the outset, resulting in the incorporation of the Authentication Service early on. In order to provide a unified design and experience across devices, the browser became the definitive target platform for otu.lea (diagnostics), lea.Dashboard (analytics) and the lea.Backend (editing). The mobile learning application (lea.App) targeted Android and iOS instead.

4.2 Technology Stack

The selection for the technology stack was heavily influenced by the author's skills, experiences, a tight schedule and limited resources. Hence, the main stack for the web-applications and services depended on MeteorJS[9], a free and open source development platform for NodeJS[10] with a tight database integration for MongoDB[11], a NoSQL database with JavaScript-like query syntax and JSON-like document structure. It allowed the project team to develop features quickly and flexibly in small iterations, thanks to its use of a single programming language across the full stack, its strong opinionated abstraction of the data-layer and its built-in reactivity model for the client. Finally, deployment to the university's hosting infrastructure was realized using a single descriptive JSON-based configuration file and a single command[12]. A more detailed approach was applied for the mobile app stack, which is further described Subsection 7.2.

[9] https://github.com/meteor/meteor
[10] https://nodejs.org
[11] https://www.mongodb.com/
[12] One-step-deployment, see [Spo04]; using https://meteor-up.com/

4.3 lea.Backend and lea.Content

The enhancement of the competency model evolved into a comprehensive content management system, named the “lea.Backend”. There was an increasing demand to centrally edit application configurations, as well as the content they consume. Furthermore all the static content should be centrally available to the consuming applications (Diagnostics, Dashboard, App), which led to the creation of lea.Content, a headless service with the single purpose to store and serve content. It is the main target of the lea.Backend editors for units and items, shown in Figure 1. The lea.Backend is only accessible to the core team members and administrators.

One challenge was deciding how content could be edited and consumed at the same time. Two different approaches have been implemented. In one approach, the consuming application requests the content on-demand. This is implemented by otu.lea and the lea.Dashboard. The other approach, implemented by the lea.App server, consists of an initial fetch and update of all necessary content on server startup. Their impact on the final products is described in the respective subsections.

4.4 otu.lea

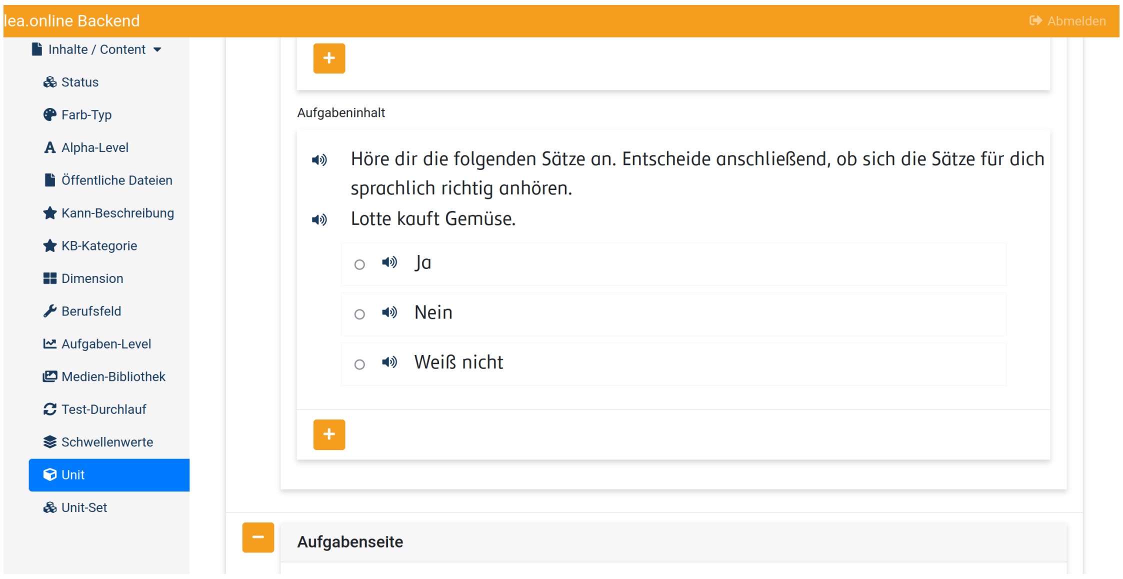
The diagnostics platform’s production app is free for the public to use online[13]. It targets desktop browsers and uses the responsive capabilities of Bootstrap[14] for different screen resolutions. Furthermore, it is installable as a Progressive Web App (PWA)[15].

Scores by Google Lighthouse for website performance are 81 (performance), 87 (accessibility), 78 (best practices) and 100 (SEO). This leaves room for improvement, for example by reducing the initial bundle size delivered to the client[16], by avoiding large layout shifts on page load and by using dynamic imports[17]. However, no performance-related issues were reported by teachers.

Currently the app requests any content (units, tasks, items, images) from lea.Content on-demand. This approach allowed a quick transition from prototype to a full application. Moreover, it reflected updates of the content immediately, which enabled editors to publish content independently. The downside was that it introduced a strong dependency on the lea.Content service, which had to be available in production at all times. This in turn complicated maintenance efforts and created a barrier for developers, who also had to install and run both applications locally. An implementation of the Strategy Pattern[18] that allows switching between the different fetching mechanisms could have helped to overcome this.

In otu.lea, every text and interactive element has auditive assistance using the browser’s builtin Text-To-Speech capabilities, which are described in full detail in Subsection 5.1. Written user input as shown in Figure 2a or selected choices are stored in the browser’s local storage[19] until the unit is completed and the responses are sent to the server. This measure helps to restore the

[13] https://otulea.lealernen.de/willkommen
[14] https://getbootstrap.com/
[15] https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/What_is_a_progressive_web_app
[16] Specifically by removing unused JS and CSS
[17] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/import
[18] https://en.wikipedia.org/wiki/Strategy_pattern
[19] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

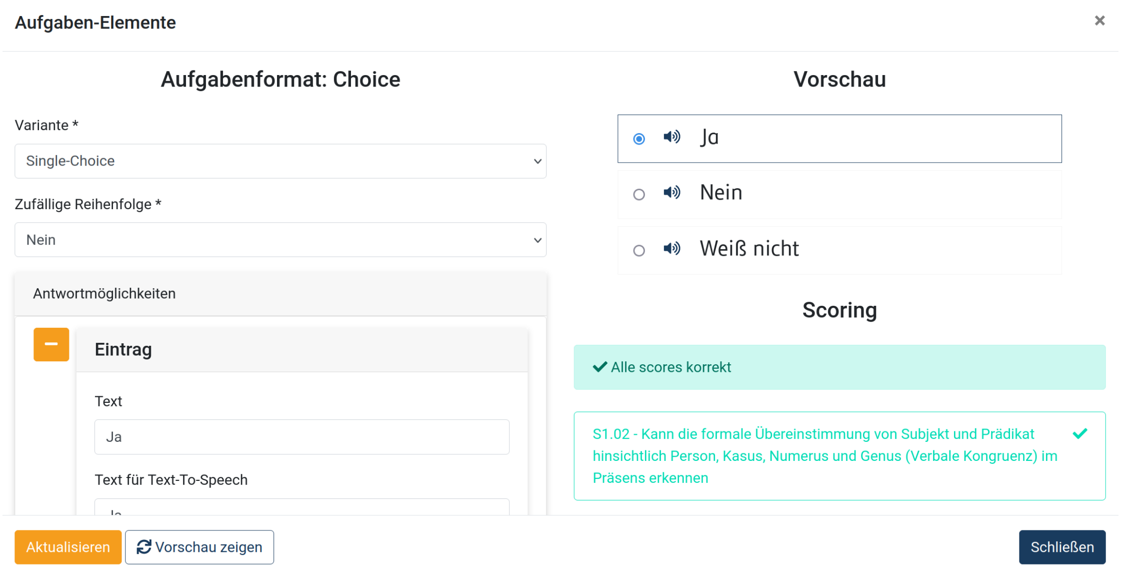


Höre dir die folgenden Sätze an. Entscheide anschließend, ob sich die Sätze für dich sprachlich richtig anhören.

Lotte kauft Gemüse.

- Ja
- Nein
- Weiß nicht

(a) Unit editor



Variante *: Single-Choice

Zufällige Reihenfolge *: Nein

Text: Ja

Text für Text-To-Speech: Ja

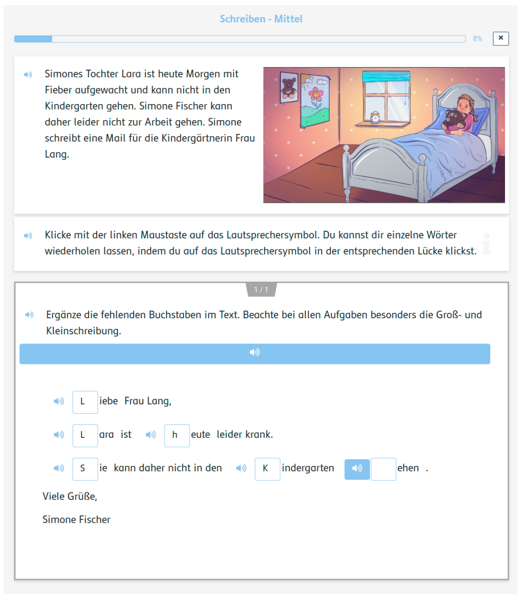
[✓ Ja] Nein Weiß nicht

✓ Alle scores korrekt

S1.02 - Kann die formale Übereinstimmung von Subjekt und Prädikat hinsichtlich Person, Kasus, Numerus und Genus (Verbale Kongruenz) im Präsens erkennen ✓

(b) Item editor with scoring preview

Figure 1: lea.Backend screenshots of various editors



Schreiben - Mittel

8%

Simones Tochter Lara ist heute Morgen mit Fieber aufgewacht und kann nicht in den Kindergarten gehen. Simone Fischer kann daher leider nicht zur Arbeit gehen. Simone schreibt eine Mail für die Kindergärtnerin Frau Lang.

Klicke mit der linken Maustaste auf das Lautsprechersymbol. Du kannst dir einzelne Wörter wiederholen lassen, indem du auf das Lautsprechersymbol in der entsprechenden Lücke klickst.

1 / 1

Ergänze die fehlenden Buchstaben im Text. Beachte bei allen Aufgaben besonders die Groß- und Kleinschreibung.

L iebe Frau Lang,

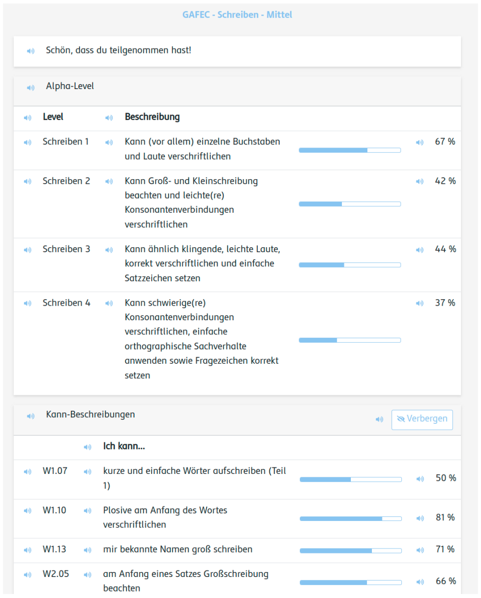
L ara ist h eute leider krank.

S ie kann daher nicht in den K indergarten ehen .

Viele Grüße,

Simone Fischer

(a) Unit view with text input items



GAFEC - Schreiben - Mittel

Schön, dass du teilgenommen hast!

Alpha-Level

Level	Beschreibung	
Schreiben 1	Kann (vor allem) einzelne Buchstaben und Laute verschriftlichen	67 %
Schreiben 2	Kann Groß- und Kleinschreibung beachten und leichte(re) Konsonantenverbindungen verschriftlichen	42 %
Schreiben 3	Kann ähnlich klingende, leichte Laute, korrekt verschriftlichen und einfache Satzzeichen setzen	44 %
Schreiben 4	Kann schwierige(re) Konsonantenverbindungen verschriftlichen, einfache orthographische Sachverhalte anwenden sowie Fragezeichen korrekt setzen	37 %

Kann-Beschreibungen

Verbergen

	Ich kann...	
W1.07	kurze und einfache Wörter aufschreiben (Teil 1)	50 %
W1.10	Plosive am Anfang des Wortes verschriftlichen	81 %
W1.13	mir bekannte Namen groß schreiben	71 %
W2.05	am Anfang eines Satzes Großschreibung beachten	66 %

(b) Diagnostic evaluation summary

Figure 2: otu.lea screenshots of a unit and evaluation summary after a diagnostic session for writing

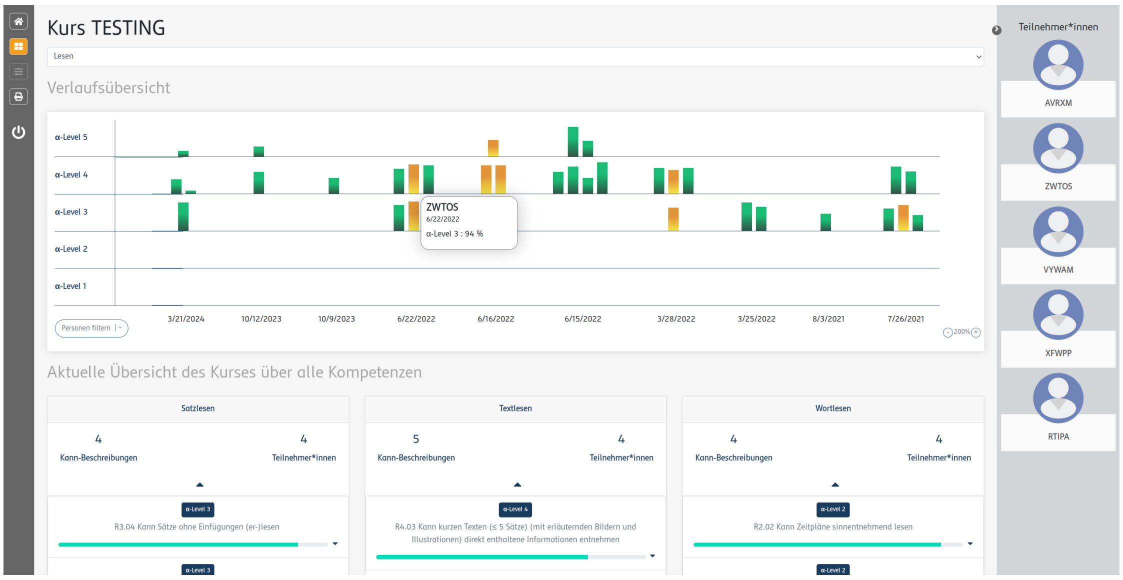
latest page state, for example when the browser has been accidentally closed, without the need to send responses to the server on every interaction.

4.5 lea.Dashboard

While the architecture and software design of the lea.Dashboard is the same as with otu.lea its main difference is the audience (teachers of literacy courses) and the fact that it is limited to a few views. Teachers can add user codes to their class in order to fetch results from otu.lea. The Dashboard processes the results over multiple sessions, in order to create an overview of the classes competencies (see Figure 3a). Furthermore, the results allow for a user-specific overview of the competencies, including a rating of whether they have improved, declined or remained the same (see Figure 3b). The scores for Lighthouse are slightly worse than for otu.lea and there is no progressive web app available. However, this circumstance is negligible as there has been no negative feedback on the performance so far by any of the over 60 institutions using it.

4.6 lea.App

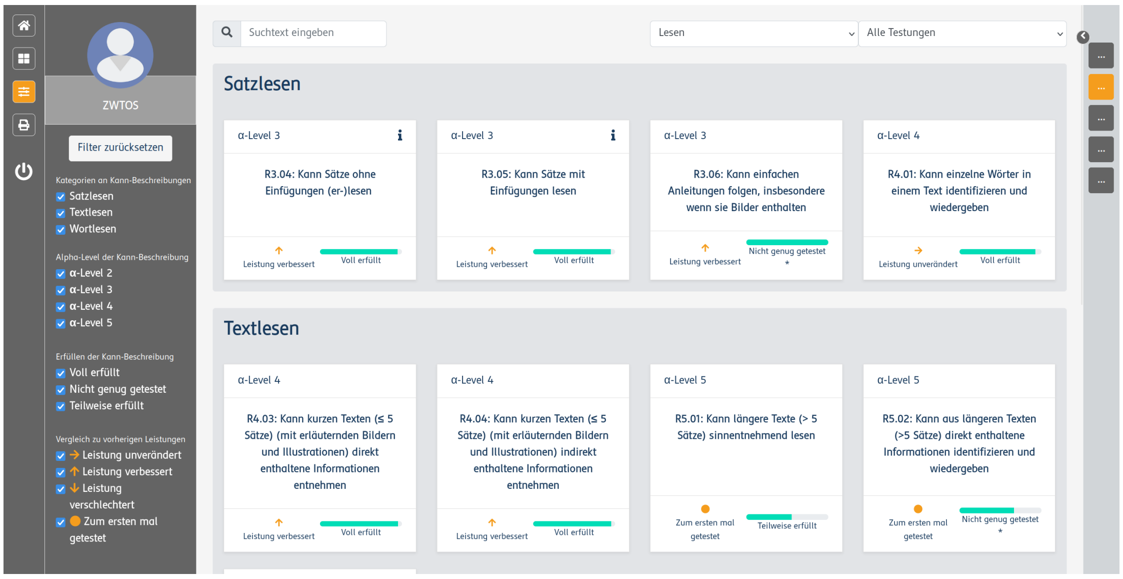
Most of the development time was spent on the lea.App [KMW+23]. It is a native mobile application, targeting Android and iOS. It was developed using React Native, a cross-platform framework to develop native mobile apps using a single codebase in JavaScript and using React to define components, layouts and rendering logic. During build the code is transpiled, then compiled using native bindings. The result is a native app, as opposed to apps that use a webview



Aktuelle Übersicht des Kurses über alle Kompetenzen

Satzlesen	Textlesen	Wortlesen
4 Kann-Beschreibungen, 4 Teilnehmer*innen	5 Kann-Beschreibungen, 4 Teilnehmer*innen	4 Kann-Beschreibungen, 4 Teilnehmer*innen
α-Level 3: R3.04 Kann Sätze ohne Einfügungen (er-)lesen	α-Level 4: R4.03 Kann kurzen Texten (≤ 5 Sätze) (mit erläuternden Bildern und Illustrationen) direkt enthaltene Informationen entnehmen	α-Level 2: R2.02 Kann Zeitpläne sinnentnehmend lesen
α-Level 3		α-Level 2

(a) Class view analytics



Satzlesen

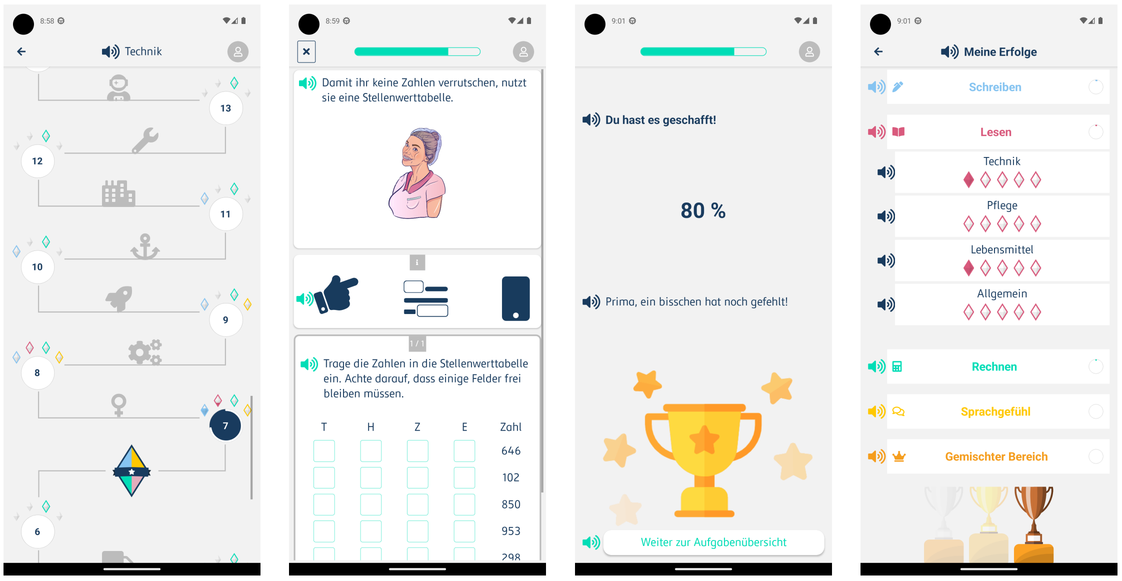
α-Level	Kann-Beschreibung	Vergleich	Erfüllen
α-Level 3	R3.04: Kann Sätze ohne Einfügungen (er-)lesen	Leistung verbessert	Voll erfüllt
α-Level 3	R3.05: Kann Sätze mit Einfügungen lesen	Leistung verbessert	Voll erfüllt
α-Level 3	R3.06: Kann einfachen Anleitungen folgen, insbesondere wenn sie Bilder enthalten	Leistung verbessert	Nicht genug getestet *
α-Level 4	R4.01: Kann einzelne Wörter in einem Text identifizieren und wiedergeben	Leistung unverändert	Voll erfüllt

Textlesen

α-Level	Kann-Beschreibung	Vergleich	Erfüllen
α-Level 4	R4.03: Kann kurzen Texten (≤ 5 Sätze) (mit erläuternden Bildern und Illustrationen) direkt enthaltene Informationen entnehmen	Leistung verbessert	Voll erfüllt
α-Level 4	R4.04: Kann kurzen Texten (≤ 5 Sätze) (mit erläuternden Bildern und Illustrationen) indirekt enthaltene Informationen entnehmen	Leistung verbessert	Voll erfüllt
α-Level 5	R5.01: Kann längere Texte (> 5 Sätze) sinnentnehmend lesen	Zum ersten mal getestet	Teilweise erfüllt
α-Level 5	R5.02: Kann aus längeren Texten (>5 Sätze) direkt enthaltene Informationen identifizieren und wiedergeben	Zum ersten mal getestet	Nicht genug getestet *

(b) User view analytics

Figure 3: lea.Dashboard screenshots of class-view and user-view


Technik
13
12
11
10
9
8
7
6
Damit ihr keine Zahlen verrutschen, nutzt sie eine Stellenwerttabelle.
Trage die Zahlen in die Stellenwerttabelle ein. Achte darauf, dass einige Felder frei bleiben müssen.
T
H
Z
E
Zahl
646
102
850
953
298
Du hast es geschafft!
80 %
Prima, ein bisschen hat noch gefehlt!
Weiter zur Aufgabenübersicht
Meine Erfolge
Schreiben
Lesen
Technik
Pflege
Lebensmittel
Allgemein
Rechnen
Sprachgefühl
Gemischter Bereich

(a) Map with levels (b) Unit with math items (c) Units completed (d) Progress tracking

Figure 4: lea.App selected screenshots of application cycle

to wrap an underlying HTML/JavaScript application. The application has a very simple and linear workflow cycle, which is shown in Figure 4. By keeping this workflow simple, it was possible to put more efforts into the user experience of the individual screens and components.

The app is freely available for the public on Android via https://play.google.com/store/apps/details?id=com.testCompany.leaonline. A closer look at the URL reveals an awkward mistake that was made when the app was configured, namely, the app id. It was set to an initial test value but not updated properly before release and was not noticed, when the first version was published to the Play Store. According to Google there is no way to change it other than to register it as an entire new app, losing existing users and creating unnecessary confusion. The URL is therefore kept as it is. The release for iOS was planned for 2022 but is delayed to autumn 2024 as there were several hidden complexities with React Native, which are described in detail in Subsection 7.2.

The mobile app is self-contained, fetching all content from its own server and therefore largely independent from the lea.Content server's availability. The lea.App server fetches all necessary content and assets at startup, ensuring no unnecessary fetching occurs unless specific environment flags are present. This approach required more effort initially but was much simpler to maintain.

4.7 Authentication Service

In order to authenticate the project team with a single sign-on, a separate service has been implemented using a library[20] that implements the authorization code grant workflow of the OAuth 2 framework, defined in RFC 6749 [Har12]. In combination with a corresponding role system, this enables a granular setting of access. For example, course instructors only have access to otu.lea and the lea.Dashboard, while the project team also have access to the lea.Backend and lea.Content. Users of otu.lea or the lea.App are authenticated by their respective applications.

The OAuth 2 standard proved difficult to understand during implementation due to its high level of abstraction for general applicability. As a result, the authentication service required significantly more effort than planned.

4.8 Shared Libraries

Shared libraries play a vital role in the overall system. They primarily enable reuse of code among several applications but they also provide for a consistent user experience: The corelib[21] contains common schema definitions for the competency model and also the Text-To-Speech engine implementation that is used in otu.lea and the lea.Backend editor. The ui library[22] provides renderers for UI components used in units, tasks and items of otu.lea. They are also used in the lea.Backend editors to render the edited content the exact same way as in otu.lea. The theme package[23] contains SCSS files that define the lea.online Bootstrap theme and is loaded by the lea.Backend, otu.lea and lea.Dashboard to ensure a homogeneous visual user experience. There are also several generic packages. These are also shared with the MeteorJS community or the larger JavaScript community via the NPM ecosystem.

A potential downside to this approach was the increased maintenance requirements due to the volume of the nearly 40 package repositories on GitHub. A major migration to a monorepo-structure could help to dramatically reduce the number of pull requests when updating dependencies.

4.9 Final Software System Overview

The final system architecture represents a distributed system of monolithic individual applications, but with a moderate coupling to the backend and the authentication service. This contrasts with pure microservices, for which a loose coupling between the applications and services is a fundamental design goal [New15]. With the exception of otu.lea, which depends on lea.Content, each application can be run and maintained independently of each other. A failure of one application should not directly affect the other applications.

The diagram, shown in Figure 5 visualizes the software architecture according to the system-level view of the C4 architecture model [Bro17]. A more detailed view of the component-level architecture is additionally shown in Figure 6.

[20] See https://github.comleaonline/oauth2-server which builds on top of https://github.com/node-oauth/node-oauth2-server

[21] https://github.com/leaonline/corelib

[22] https://github.com/leaonline/ui

[23] https://github.com/leaonline/theme

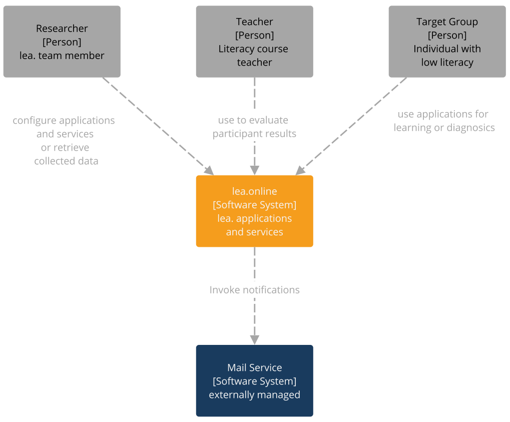


Figure 5: lea.online system-level architecture diagram

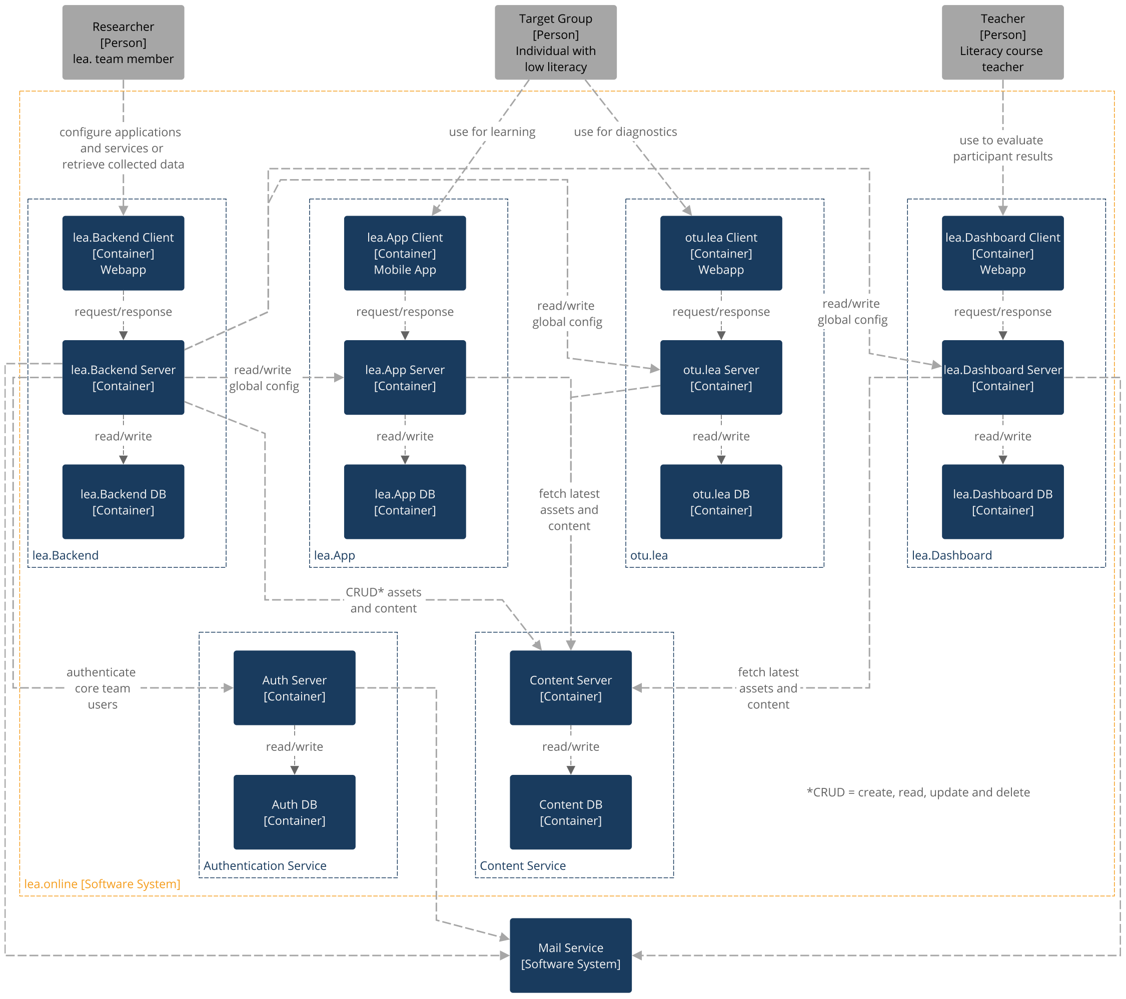


Figure 6: lea.online component-level architecture diagram

Instead of using a shared database, each of the application servers communicates with its own database. This pattern was borrowed from the microservices architecture to achieve self-containment for each application and service. It provides the advantage of selectively updating or maintaining them individually and without affecting the rest of the system. In the later stages of the project, this helped to deliver critical updates in a timely and frictionless manner. However, it increased the overall complexity of the system and the need for data exchange between the applications and services.

The communication endpoints are all RPC (Remote Procedure Call). This was chosen over REST (Representational State Transfer) because there was no requirement for any of the application APIs to be consumed publicly. The endpoints were designed in accordance to the logic of the application. This introduced a level of coupling that was considered a fair trade-off for the faster pace of development.

Authenticated requests require appropriate tokens, either obtained from the application's back-end or from the authentication server. A temporary solution for resource exchange between the lea.App server and the content service has been implemented through using JWT with sealed secrets. This will be replaced by an OAuth 2 client credentials workflow designed specifically for server-to-server communication.

5 Accessibility

As already mentioned in Subsection 2.3 the low literacy of the target group implies several requirements for software accessibility. A starting point for accessibility in web applications is the Web Content Accessibility Guidelines (WCAG) from the W3C[24]. This section reflects on selected topics that were found to be a valuable addition to these and other accessibility guidelines.

5.1 Text To Speech

The former otu.lea Flash-based application used recorded voices to assist with texts and interactive elements. While advantageous in presentation, thanks to the professional speakers' clear voice, this was most inflexible in regards to changing and improving content. Instead, the new otu.lea web application builds on top of the Text-To-Speech (TTS) capabilities of the Web Speech API[25]. Screenreader support was not considered an alternative, because it had never been requested by any institution in the 8 years of use.

The greatest advantage of browser-based TTS lies in its immediate availability without the need for installation. Furthermore, nearly any textual content can be synthesized to provide assistance with low effort. To maximize the effect across browsers, a generic library[26] has been developed that is also usable beyond the lea.online project.

The greatest disadvantage is the variable quality of the various voices, which is dependent on the operating system and browser. It cannot be polyfilled or shimmed by applications or

[24] https://www.w3.org/TR/WCAG22/ with supplemental guidance for cognitive accessibility at https://www.w3.org/WAI/WCAG2/supplemental/#cognitiveaccessibilityguidance
[25] https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API
[26] https://github.com/leaonline/easy-speech

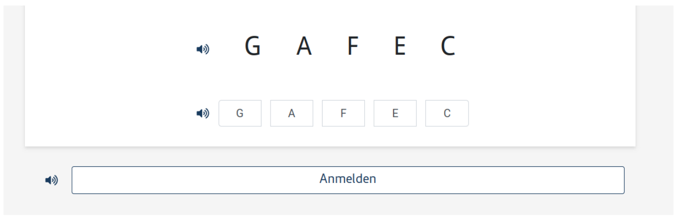


Figure 7: otu.lea login with text-to-speech support

libraries. Additionally, domain-specific terms or some anglicisms are insufficiently supported and result in confusing output. Tools[27] to train custom models of high quality do exist, however the Web Speech API does not provide an implementation to load custom voices. Commercial cloud-based solutions with high quality voices do exist, too. However their on-demand based payment structure is unsustainable in the context of public projects.

A future plan is to move TTS to a standalone service using a custom voice, trained by a professional speaker. The TTS is then synthesized on the server and streamed as compressed audio to the client as partial HTTP response (206). This could provide a consistent TTS experience for all users, including support for domain-specific pronunciations.

5.2 Accessibility vs. Security

Proficiency in any existing authentication procedure by the target group cannot be assumed. At the same time the threshold for frustration among target group members might be low when faced with barriers. Usability testing showed that even with a simple five-character code-based login, some participants still needed help [Kop17]. This led to the requirement that authentication needs to be as free from barriers as possible. Usernames are not required in order to guarantee anonymity.

There were no definitive solutions found during the project phase that satisfied all criteria, including the need for a strong security. User-defined passwords are problematic, as they imply a certain level of reading and writing skill. They would have excluded individuals on the lowest alpha levels. Password-free alternatives should be favored.

One-time logins via email (magic links) were not considered, since only about 30% oft the target group uses emails [GBD+19]. OAuth-based logins with external services, such as offered by several social media platforms were not considered due to privacy concerns.

In otu.lea the five-character login procedure still represents the minimal modest solution. In practice, these codes are generated by the literacy course teachers and handed out to the participants before the diagnostic assessment starts. The login is supported by TTS, which allows the newly generated code and the actually entered code to be read aloud, as shown in Figure 7. A rate-limiter on the login endpoint aims to reduce the surface for simple brute-force attacks, however due to the short length of the code these measures are only basic. Second- or multi-factor authentication involves extra steps in the login-sequence and was thus considered to be an obstacle that could increase the level of frustration.

[27] https://github.com/coqui-ai/TTS

The FIDO alliance defined guidelines for making authentication accessible and is also aware of the target group from a global perspective [Yao22]. A fundamental issue with their guidance is the assumption that users will use devices with assistive technologies in order to set up the use of protocols, like WebAuthn or CTAP (Client to Authenticator Protocol). Recent analytics of the lea.App showed no activation of the operating system's Assistive Technologies (AT) in any account among about 800 entries. Upcoming large-scale inquiries about the target group should explicitly include the distribution of these AT to gain profound knowledge for future decision making.

Similar difficulties may be faced with the rather complex procedures for setting up passkeys[28]. Physical devices (such as USB-dongles) represent a viable low-barrier alternative but were also not considered, as there were no ways to ensure that users would be able to obtain and configure such a device on a large scale. However, there is no distinct evaluation yet on any of these solutions involving the target group. This represents a unique opportunity for upcoming studies.

5.3 Accessibility of Legal Texts

All applications of lea.online contain imprint, terms of service and privacy policy as required by law. The juridical nature of these texts poses a barrier to understanding, even for non low literacy individuals. Texts can be supplemented by TTS but this does not resolve the issue. Alternative texts in easy language and simple language, as suggested by the WCAG, are not legally binding. Future research should explore the simultaneous presentation of text in both legal and accessible languages in the context of mobile device screens. However, a solution that is both accessible and legally valid remains to be found.

6 Privacy

User privacy was a fundamental consideration in the design and implementation of the applications. However, this approach also presented its own set of challenges, which are discussed in this section.

6.1 Privacy design in otu.lea and the lea.App

Privacy concerns were addressed during the application phase of the project by taking an anonymity-first approach. As a result, the user accounts in otu.lea or the lea.App do not store any personal information, and no personal information is requested at any time. Usernames and passwords are randomly generated. However, IPs have to be logged as security measures but are are not associated with any specific account. The user-agent and the screen dimensions are anonymized and stored to provide insight and optimization for layout responsiveness and browser compatibility. In addition, the lea.App collects a more comprehensive set of non-sensitive device information as there is a wide variety of hardware to support in the Android ecosystem.

This privacy-conscious design directly impacts users. If the mobile device is lost or the app is deleted, it becomes necessary to reinstate the learning progress. Nevertheless, this process

[28] For an example see the required steps in https://developers.google.com/identity/passkeys

presents the same challenges as the general authentication procedure, which has been previously outlined in Subsection 5.2. In order to facilitate the restoration, a 12-character-sized restore code is provided, comprising three rows of four uppercase alphanumeric letters each. This code has to be noted by the user manually. This method is optimal in regards to privacy as there is no necessity to request personal data. However, is lacks a proper user experience and is prone to errors, as confirmed by a recent logfile analysis. As with general authentication, a secure, privacy-aware, and accessible solution remains to be found.

6.2 GDPR vs. Research

Legal consultations underlined that participation in research by the target group requires double-opt-in to be legally valid. However, the privacy and accessibility design does not involve email as a means to realize this in an automated way. Therefore, the alternative is a paper-based participation agreement by attendees of literacy courses. This circumstance currently prevents the publications of recent findings that reflect back on the target group. A solution that is legally valid and applicable is yet to be found.

7 Software Engineering

This section considers a number of selected challenges and specific software engineering practices that were undertaken in the context of an understaffed development team.

7.1 Outsourcing Development

Outsourcing was regarded as a potential solution to compensate for the lack of resources and expertise. The contracts were based on detailed requirements documents that outlined the desired functionality, accompanied by explanations of abbreviations and domain-specific terminology. There were minimal issues in terms of contractors' understanding of the deliverables.

However, in the case of one contract, the delivered React Native components were only deemed to be functionally acceptable, yet lacked a clean code design, comprehensive documentation, and sufficient testing. Some components were mere copies with only minimal differences, indicating a lack of fundamental knowledge, such as that pertaining to polymorphism. This resulted in a much higher integration effort than anticipated, as these classes had to be rewritten. In retrospect, the non-functional aspects of code quality and testing requirements were insufficiently described in the requirements document, and the acceptance criteria were too focused on the functional aspects. Acceptance criteria should additionally include how tests are expected to cover the newly provided code.

Another time-consuming aspect of this contract was by the lengthy and exhausting process of communication. It was not possible to establish direct contact with the responsible developers, as all communications were conducted through the project manager. All commits were made in a private repository by an anonymous GitHub account that had been specifically created for this contract. In light of these circumstances, it was postulated that the manager may have subcontracted the development process to a third party in an effort to reduce costs, albeit at the potential expense of quality and communication. As a potential future measure, the requirements document should explicitly demand contact with the relevant developers, who will implement the solutions, in order to prevent managers from gatekeeping and controlling the communication. Additionally, there should be an explicit policy that prevents the contractor from outsourcing development to a third-party.

7.2 Cross Platform Mobile Development

The elicitation and selection for a proper stack towards a sustainable mobile app development was a challenging task in the middle phase of the project (late 2019 to early 2020). At the time, the following factors played an important role in the decision-making process:

1. **Project plan.** A production-grade release had to be published by the end of 2021.
2. **Personal.** The stack should be suitable to a very small team of one full-time developer and up to three student assistant developers. Managing multiple code-bases must be avoided.
3. **Platform.** The (mobile) target platforms are Android and iOS. It should take as little effort to deploy both. At the same time it should be possible to develop on as many Desktop platforms as possible.
4. **Parallel Development.** The app is to be developed in parallel with the other applications (dashboard, diagnostics, services etc.) in order to meet the overall release schedule.
5. **Skillset.** The developers' current knowledge and experience with programming languages, their primary paradigms, frameworks and libraries has to be considered. A stack with familiar technologies is prioritized to reduce the mental load while working on multiple projects.
6. **Learning curve.** The learning curve of the stack and its available resources (documentation, tutorials etc.) should be considered to keep on-boarding within a manageable time.
7. **Ecosystem.** The stack should reside within a free and open ecosystem, with available packages/libraries and an active community of contributors and maintainers and/or a strong backer behind the main technology, such as a company or foundation.

7.2.1 Selection Process

At the time of the selection, there was already a prototypical server-backend written using MeteorJS. The pre-selection process relied on GitHub, because it is the largest platform hosting open source repositories, including the most popular but also most recent ones. It resulted in a final comparison between Flutter[29] and React Native[30], as these were the most popular repositories when the development began. The Flutter framework offered a more complete set of tools to

[29] https://github.com/flutter/
[30] https://github.com/facebook/react-native

target multiple platforms and a more comprehensive documentation, including a UI widget catalog out-of-the-box. It is backed by Google and provided its own package management. The Dart language appeared similar to JavaScript in its syntax. In contrast, React Native required the entire codebase to be written in JavaScript and JSX. There were no new concepts or paradigm to learn. The documentation was sufficient but not as detailed as with Flutter. It provided compatibility with the NPM[31] registry which contains over one billion packages for the JavaScript ecosystem.

7.2.2 Selection Outcome and Retrospective

Both platforms were able to cater to the needs of the project plan, personal, platform and learning curve. There was a slight advantage for Flutter, due to its richness in documentation. However, the selection went for React Native, which better met the existing skillset and offered a much larger ecosystem. Using the same programming language for all application was assumed to be the best way to enable parallel development.

Retrospectively the author questions this choice. First, the NPM ecosystem contains a large number of outdated or abandoned React Native packages. A remedy was provided by Expo[32], a framework for React Native with native integrations and build-tools. Nowadays, Expo is even suggested by the official React Native installation guide. Given that Expo has its own release-cycle and dependencies, it demanded an unplanned additional maintenance effort. Its toolchain for building the native target platforms dramatically simplifies the process in comparison to the native workflow. However, full release management from code to app store is only available to subscribers of their commercial cloud services. This option was deemed unsustainable, due to the payment-on-demand structure and server location outside of the EU. This resulted in an intensified deployment effort, due to the deployment process being only partially automated.

7.3 Connectivity and Offline-first

A disastrous decision in the planning phase was to defer the implementation of full offline capabilities to a later stage in the project, while relying on the Websocket-based MeteorJS backend for the mobile app. With that design, the mobile app assumes a stable connection to the lea.App server in order to fetch new units, items and images. In isolated view the design and implementation are stable and cause no major errors in production. However, an internal review of production errors revealed a group of edge-cases that caused such errors when the app ran in the background for a longer time and was then brought back to the active state. At that point the app's state management failed to reconnect to the server, causing a timeout on any attempted request. Further investigation revealed the overall state of ”being connected” vs. ”should reconnect” is influenced by the interplay of multiple factors:

1. Whether a connection to the internet is established on the operating system level (via WiFi or mobile data).
2. Whether the app is open.

[31] https://npmjs.com
[32] https://expo.dev/

3. Whether the app is active[33].

4. Whether the app has established a Websocket connection with the server and there is no connection timeout.

On top of these factors, it is assumed that all the connectivity-related code is free of errors. The sum of these factors made it difficult to ensure availability at all times and the attempted fixes for the above described errors resulted in unplanned extra effort. Retrospectively the choice for a Websocket should have been avoided in favor of a stateless REST-based solution and an offline-first design from the beginning. A future concept for migrating to an offline-first architecture should be evaluated. This evaluation could focus on the identification of best practices and challenges in refactoring and software architecture.

7.4 Addressing the Bus Factor

The bus factor (or truck factor) is a measure used to describe the number of employees needed to depart in order to stall the entire project. It is assumed to have the same effect as if they were to be struck by a vehicle of considerable mass, such as a bus or truck [JETK22]. A value of one is assigned to the worst case scenario, reflecting the fact that critical knowledge is concentrated in the hands of a single individual.

The original project plan anticipated the employment of two half-time developers but it was not possible to find a second developer. Consequently, the single existing developer (the author) has been employed full-time to attempt to meet the project milestones. This resulted in a total centralization of knowledge, which imposed a high risk of losing valuable information about code, design and processes. Therefore, countermeasures were implemented with a focus on transparency and documentation alongside code quality and automation.

The entire set of repositories for all applications, services and libraries was developed in public on GitHub from the initial commit. This aimed to force a certain degree of understandable branch-naming and commit conventions, making the code more searchable for specific features or fixes. In retrospect, this process could have been optimized by using conventional commits[34] to avoid non-scoped or generic commit messages. Additionally, a strict branch-per-feature or branch-per-fix convention should have been applied in order to backtrack changes in isolated environments. The effects of these measures have not yet been researched but could be a viable subject of future research.

Another measure was to provide extensive documentation for the mobile app development installation, since it is the most complex process in the lea.online system, involving multiple tools and frameworks to be installed. It was tested during the on-boarding of new student assistants without any additional help. The procedure revealed a lack of reproducibility for the installation of the mobile platform SDKs across different development platforms and missing troubleshooting information.

[33] https://reactnative.dev/docs/appstate
[34] https://www.conventionalcommits.org

8 Critical Reflection and Outlook

This work provided an overview of the research and development project lea.online, presenting selected findings on topics in research software engineering. It aimed for a horizontal view by presenting a wide range of topics with varying depth, thereby underscoring the complex nature of interdisciplinary research projects. It was frequently observed that sustainable solutions, which may initially require a greater degree of effort, could prove to be a worthwhile investment in the long term. However, this was not always feasible within the given project timeline.

Many of the reported topics allow for further investigation and may become the subject of future research. The author is explicitly open to collaboration. Additionally, topics such as software architecture or privacy could have been described in more detail but would have constituted a publication in themselves. Finally, future work would undoubtedly benefit from a more standardized empirical research design that is prepared in advance of the projects.

The lea.online project has formally ended but the software system remains operational and maintained by the author's working group. Applications for follow-up research projects including active work on the software system are in current progress.

Bibliography

[BMB16] E. G. Belay, D. S. McCrickard, S. A. Besufekad. Claims-to-Patterns Approach to Leverage Mobile Interaction Design for Low-Literacy Users. In *Proceedings of the 7th Annual Symposium on Computing for Development*. ACM DEV '16. Association for Computing Machinery, New York, NY, USA, 2016.
doi:10.1145/3001913.3001928
https://doi.org/10.1145/3001913.3001928

[Bro17] S. Brown. The C4 model for visualising software architecture. 2017.
https://c4model.com/

[GAB+23] F. Goth, R. Alves, M. Braun, L. J. Castro, G. Chourdakis, S. Christ, J. Cohen, F. Erxleben, J.-N. Grad, M. Hagdorn, T. Hodges, G. Juckeland, D. Kempf, A.-L. Lamprecht, J. Linxweiler, M. Schwarzmeier, H. Seibold, J. P. Thiele, H. von Waldow, S. Wittke. Foundational Competencies and Responsibilities of a Research Software Engineer. Nov. 2023. arXiv:2311.11457 [physics].
doi:10.48550/arXiv.2311.11457
http://arxiv.org/abs/2311.11457

[GBD+19] A. Grotlüschen, K. Buddeberg, G. Dutz, L. Heilmann, C. Stammer. LEO 2018 - Leben mit geringer Literalität. Pressebroschüre, Hamburg, 2019.
https://leo.blogs.uni-hamburg.de/

[Har12] D. Hardt. The OAuth 2.0 Authorization Framework. Request for Comments RFC 6749, Internet Engineering Task Force, Oct. 2012.
https://datatracker.ietf.org/doc/rfc6749/

[HM14] D. Heisler, G. Mannhaupt. *Analphabetismus und Alphabetisierung in der Arbeitswelt*. Peter Lang Verlag, Berlin, Germany, 2014.
doi:10.3726/978-3-653-03543-8
https://www.peterlang.com/document/1048095

[JETK22] E. Jabrayilzade, M. Evtikhiev, E. Tüzün, V. Kovalenko. Bus factor in practice. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '22, p. 97–106. Association for Computing Machinery, New York, NY, USA, 2022.
doi:10.1145/3510457.3513082
https://doi.org/10.1145/3510457.3513082

[KKW13] I. Koppel, J. Küster, K. D. Wolf. Usability testing with female functional illiterates, Usability-testing mit funktionalen analphabetinnen. *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)* P-218:293–296, 2013.

[KMW+23] J. Küster, I. A. M. Meyer, M. Windler, K. D. Wolf, I. Koppel. lea.online App. Oct. 2023.
doi:10.5281/zenodo.10816689
https://zenodo.org/records/10816688

[Kop17] I. Koppel. *Entwicklung einer Online-Diagnostik für die Alphabetisierung: eine Design-Based Research-Studie*. Springer VS research. Springer VS, Wiesbaden, 2017.

[KW14] I. Koppel, K. Wolf. otu.lea: eine niedrigschwellige Online-Diagnostik für funktionale AnalphabetInnen in der Kursarbeit. *Alfa-Forum*, pp. 38–41, 01 2014.

[LPM22] J. Linxweiler, S. Peters, S. Marcus. SURESOFT:Principles of Software Engineering. Sept. 2022.
doi:10.5281/zenodo.7120360
https://zenodo.org/records/7120360

[Mac13] F. R. Maciel. PALMA: Usability Testing of an Application for Adult Literacy in Brazil. In Marcus (ed.), *Design, User Experience, and Usability. Health, Learning, Playing, Cultural, and Cross-Cultural User Experience*. Pp. 229–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[MPB+11] I. Medhi, S. Patnaik, E. Brunskill, S. N. Gautama, W. Thies, K. Toyama. Designing mobile interfaces for novice and low-literacy users. *ACM Trans. Comput.-Hum. Interact.* 18(1), may 2011.
doi:10.1145/1959022.1959024
https://doi.org/10.1145/1959022.1959024

[MW21] I. A. M. Meyer, K. D. Wolf. Intuitive Visualization of Complex Diagnostic Datasets to Improve Teachers' Individual Support of Learners Based on Data

Driven Decision Making. In Stephanidis et al. (eds.), *HCI International 2021 - Posters*. Pp. 102–108. Springer International Publishing, Cham, 2021.

[MWWK24] I. A. M. Meyer, K. D. Wolf, M. Windler, J. Küster. Digitale berufsfeldbezogene Förderung von Literalität und Numeralität in der arbeitsorientierten Grundbildung mit der lea.App. *Berufs- und Wirtschaftspädagogik - online*, 2024.
https://www.bwpat.de/ausgabe/spezial-ht2023/meyer-etal

[New15] S. Newman. *Building microservices: designing fine-grained systems*. O'Reilly Media, Beijing Sebastopol, CA, first edition edition, 2015. OCLC: ocn881657228.

[RHRR12] P. Runeson, M. Höst, A. Rainer, B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, 1 edition, Mar. 2012.
doi:10.1002/9781118181034
https://onlinelibrary.wiley.com/doi/book/10.1002/9781118181034

[Spo04] J. Spolsky. *Joel on software: and on diverse and occasionally related matters that will prove of interest to software developers, designers, and managers, and to those who, whether by good fortune or ill luck, work with them in some capacity*. Apress, Berkeley, CA, 2004.

[Woh21] C. Wohlin. Case Study Research in Software Engineering—It is a Case, and it is a Study, but is it a Case Study? *Information and Software Technology* 133:106514, May 2021.
doi:10.1016/j.infsof.2021.106514
https://linkinghub.elsevier.com/retrieve/pii/S0950584921000033

[Yao22] Yao Ding. Guidance for Making FIDO Deployments Accessible to Users with Disabilities. White paper, FIDO Alliance, 2022.
https://fidoalliance.org/wp-content/uploads/2022/10/Guidance-for-Making-FIDO-Deployments-Accessible-to-Users-with-Disabilities_FINAL.pdf