Hochschule Konstanz
Fakultät Informatik

Bachelor Thesis

A Holistic Approach to Developing an Engaging and Effective E-Learning Platform

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Bachelor of Science

In the
Department of Computer Science
Hochschule Konstanz University of Applied Sciences

Submitted by
B. Eng. Niels Boecker

Email
niels.boecker@htwg-konstanz.de

Student Number
292041

Supervisor
Prof. Dr. Marko Boger

Second Examiner
Prof. Dr.-Ing. Oliver Haase

Date of Submission
February 28, 2018

A Holistic Approach to Developing an Engaging and Effective E-Learning Platform

Bachelor Thesis

Hochschule Konstanz University of Applied Sciences

Submitted by: B. Eng. Niels Boecker

Email: niels.boecker@htwg-konstanz.de

Student Number: 292041

Supervisor: Prof. Dr. Marko Boger

Second Examiner: Prof. Dr.-Ing. Oliver Haase

Date of Submission: February 28, 2018

Application: https://codetask.in.htwg-konstanz.de

Source Code: https://github.com/nbckr/codetask

Declaration of Authorship

Version: 2
Stand: 17.09.2017
Autor: Prof. Dr. Rainer Mueller

Hiermit erkläre ich, Niels Boecker, geboren am 7. Juli 1989 in Stuttgart,

(1) dass ich meine Bachelorarbeit mit dem Titel:

A Holistic Approach to Developing an Engaging and Effective E-Learning Platform

in der Fakultät Informatik unter Anleitung von Prof. Dr. Marko Boger selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

(3) dass die eingereichten Abgabe-Exemplare in Papierform und im PDF-Format vollständig übereinstimmen.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 28. Februar 2018

Abstract

Building an e-learning system makes for an exciting exercise in software engineering, as it strives to solve a significant real-world problem and touches many aspects that go beyond technology. Centered on the question of which factors determine learning success and how they can get employed in the context of online education, this study concerns itself with the analysis and implementation of a web-based e-learning application that comprises interactive programming exercises.

A solid theoretical framework complemented by comprehensive analysis will pave the way to solving this practical problem. Accordingly, a detailed account will be given on the technical challenges and solutions. Several state-of-the-art concepts and technologies in the field of web development will be applied, such as centralized state management and near real-time synchronization with remote clients.

In a holistic approach that goes beyond these technological considerations, determinants of learning success and the development of an engaging e-learning product will be researched, such as the driving psychological factors behind learning and the subconscious emotions triggered in the process of human-machine interaction. Specifically, user experience guidelines and best practices as well as gamification methods will be put into practice.

As a whole, the study offers a big-picture approach composed of a review of concerned factors as well as the implementation of what is hopefully a web-based learning platform that students will actually *want* to use.

Table of Contents

List of Figures

In order to obtain a uniform appearance, all custom and adopted charts were reproduced in a content-preserving manner. This was conducted by junior graphic designer Tim Peter, who has four years of experience in designing scientific publications for *Fraunhofer Institute for Industrial Engineering IAO*.

List of Abbreviations

2G	Second-generation cellular technology
AWS	Amazon Web Services
CD	Corporate design
CI	Continuous integration
CLI	Command-line interface
CSRF	Cross-site request forgery
DIN	Deutsches Institut für Normung
DOM	Document object model
DSL	Domain-specific language
ES	ECMAScript
FLIP	First, last, invert, play
fps	Frames per second
IDE	Integrated development environment
IoT	Internet of things
JSX	JavaScript Syntax eXtension
JWT	JSON Web Tokens
MOOC	Massive open online course
MVC	Model–view–controller
NPM	Node package manager
PaaS	Platform-as-a-Service
PWA	Progressive webapp
Sass	Syntactically awesome style sheets
SDK	Software development kit
SPA	Single-page application
SSR	Server-side rendering
SVG	Scalable vector graphics
UI	User interface
UID	Unique identifier
UX	User experience
VCS	Version control system
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines

1 Introduction

1.1 Purpose and Structure

The study at hand concerns itself with the analysis and re-implementation of an interactive e-learning application that teaches the *Scala* programming language. This application has been built for the *Department of Computer Science* at *Hochschule Konstanz University of Applied Sciences* and should have already been in use. However, some technical and usability-related issues made it impractical hitherto. Consequently, the department requested to look into said shortcomings and to produce an improved version of the platform. This was the starting point and raised some inspiring questions, including, but not limited to, the choice of appropriate technologies and tools.

In a holistic approach, the study will explore which factors determine learning success and how they can get employed in the context of online education. Guided by this question, the technical aspects of e-learning will be examined: What are inherent challenges and opportunities? How can it help to include impaired students and reach isolated learners in developing countries? Then, the subject focus will shift to the development of online courses and platforms. What are the underlying methods of building an engaging and meaningful interface? Which tools are available to increase students' motivation and affect their attitude towards a system? In order to research various potential determinants and pursue a full picture of concerned factors, aspects of human behavior, psychology, learning motivation, and pedagogy will get covered alongside the technical topics.

Based on the findings of the research, an educated analysis of the existing application will be conducted. Also, the application will be compared to some of the most successful real-world e-learning platforms. All subjects will be examined in terms of the instruments used to enhance motivation and engagement. As a result, the strengths and weaknesses will become clear, thereby giving directions as for how to implement and optimize the new application.

With this in mind, the design and development of the web application will be carried out. The findings of the theoretical framework and the observations of the software analysis will be put into practice and complemented with technical considerations. Advanced concepts of web development will be introduced to offer centralized local state management in conjunction with near real-time synchronization between clients. Eventually, the application will be deployed in a productive environment and made available to department and student body.

To come full circle and get a comprehensive overview, the results yielded from theory, analysis and practice will be revisited and discussed, evaluating the employed tools and methods to reach a final conclusion.

"Learning is naturally fun", says Shantanu Sinha, former president and COO of the popular e-learning platform *Khan Academy*, "and students should *want* to learn. However," he continues, "most students seem to steadily lose their natural enthusiasm and curiosity, as they grow older"[1]. *Khan Academy* and its competitors make an effort to fight the shortcomings of e-learning and education in general, making for a fun and engaging experience. This study is an attempt to follow their lead and create an engaging application for the department.

In short, the following chapters will cover the theory and practice of building what is hopefully a web-based learning platform that students will actually *want* to use.

1 Sinha 2012 [emphasis in original].

1.2 Organization of Research

The body of research addressing e-learning, user experience, usability, accessibility, learning motivation and gamification is huge. There are academic communities and journals that concern themselves with some of these topics exclusively, such as the *Journal of Online Learning and Teaching*. A selection of relevant literature, journal articles, studies, and papers has been consulted and will be discussed and put into context. Regarding technical aspects, the most valuable resources were online articles, blog entries, project documentation etc., as web technologies are such a fast-moving field that traditional printed publications cannot seem to keep up.

2 Theoretical Framework

In this chapter, the theoretical groundwork for the implementation will be laid and several aspects connected to implementing an e-learning platform with state-of-the-art web technologies will get covered in-depth.

2.1 Online Education

Online education has become very popular, it is shaping the way we learn and challenging the monopoly position of traditional classroom teaching: In a 2016 study of 25,000 young people from around the world, 77.84% reported having taken online courses in the past.[2] To gain a better understanding of this phenomenon, this section will examine different perspectives of e-learning. After introducing important terms and basic knowledge, the psychology of learning motivation will be dissected. Eventually, some technical and organizational aspects of developing and providing online courses will be discussed.

2.1.1 Key Terms and Definitions

E-LEARNING—or ONLINE LEARNING, a term often used synonymously—is a manifestation of distance learning. It could be generally defined as "the delivery of education content via all electronic media"[3]. However, it is virtually only used in combination with the internet and web-based learning. Hewitt offers a comprehensive definition and differentiates e-learning from adjacent terms:

> E-learning (electronic-learning) refers broadly to the use of information and communication technologies for the electronic delivery of instructional content and the support of educational processes. Computer-supported learning can vary from being completely "self-paced," with no human instructor, to being highly coordinated, with the instructor interacting with students at a distance. Thus, it is a broad concept that encompasses such phrases as *online learning*, *Web-based learning*, *computer-based instruction*, *Web-based training*, and *virtual education*. E-learning systems have been developed for many different purposes and audiences, for both formal and informal learning contexts. With the rapid growth of the Web over the past decade, the popularity of the term *e-learning* has faded slightly, in favor of the more widely used term, *online learning*. However, both terms still appear regularly in the media, and the phrases e-learning and online learning are often employed interchangeably.[4]

Another closely related, and sometimes overlapping term is EDUCATIONAL TECHNOLOGY, which is defined as "the study and ethical practice of facilitating learning and improving performance by creating, using, and managing appropriate technological processes and resources"[5].

While online education has been around since the 1960s, it has grown significantly since the start of the millennium.[6] In 2008, the term MASSIVE OPEN ONLINE COURSE (MOOC) emerged; a MOOC is an online course that offers unlimited participation and open access. Typically, MOOCs are available under an open license and free of charge. Besides filmed lectures, readings, and problem sets, they often offer interactive exercises and forums to enable interactions among students and staff.[7]

There has been a lot of media attention and hype around MOOCs, going as far as calling them a "revolution". However, Maha Bali argues that MOOCs are just the next logical step, following

2 Refer to Yu and Hu 2016.

3 Milovanovic 2010, 191.

4 Hewitt 2015 [emphasis in original].

5 Robinson et al. 2013, 15.

6 Refer to Yu and Hu 2016.

7 Refer to Kaplan and Haenlein 2016, 441–50.

online learning and open educational resources.[8] Platforms like *Coursera*, *edX* or *Udacity* allow users from all over the world to attend courses by renowned universities like *Harvard*, *Stanford* etc. Some higher-education institutions even consider offering credits for their online courses instead of a mere statement of accomplishment.[9]

A special variant of online education are so-called PROGRAMMING KOANS. They consist of actual source code, more specifically: "a set of failing unit-tests, for which the learner is required to fix the code or fill the required fields with proper values in order to make each test pass"[10]. Koans are available for a multitude of programming languages, and they are usually targeted at learners with some degree of prior knowledge.

2.1.2 Learning Motivation

MOTIVATION is "the extent to which persistent effort is directed toward a goal"[11]. Commonly, a distinction is made between intrinsic and extrinsic factors. Intrinsic factors include the individual's attitude and expectations as well as goals and emotions; the reward of doing something is the mere satisfaction it brings. Extrinsic factors, on the other hand, stem from external incentives like reward and recognition, payment or social pressure.[12]

LEARNING MOTIVATION, according to Law et al., denotes "the extent to which persistent effort [of] a student [is directed] toward learning"[13].

Another popular theory of motivation is the SELF-DETERMINATION THEORY. It concerns itself with the motivation behind choices made without external influences. This theory puts forth three main needs humans need to fulfill, namely:[14]

- **Competence:** Gaining mastery over some challenge
- **Relatedness:** Social interaction with other people
- **Autonomy:** The ability to make choices and have control.

2.1.3 Opportunities and Challenges

E-learning comes with its own set of opportunities and challenges. According to the aggregated opinions of many researchers in the field[15], the most important opportunities are:

- **Education for all:** Can overcome the exclusion of traditional education; can overcome barriers of disability; can reach remote areas and underdeveloped countries.
- **Delivery anywhere and anytime:** Lessons can be reviewed at the individuals' convenience; not time-bound; not stationary.
- **Personalized learning:** Yields higher retention of learning content; supports several different learning styles; materials can be reviewed as often as needed; smaller content units contribute to a more lasting learning effect; in particular helpful to underperforming students and students with special needs.
- **Improved collaboration and interactivity:** Peer-to-peer collaboration or interaction with experts; can encourage more critical reasoning, enjoyment, and peer contact than traditional learning.
- **Effective use of IT:** Rapid and just-in-time delivering of knowledge and skills; cost-effective way for organizations to train employees; cost savings through eliminated travel expenses; better skilled workforce.

8 Refer to Bali 2014, 44.
9 Refer to Lewin 2012.
10 Iravanian 2012.
11 Law et al. 2010, 220.
12 Refer to ibid.
13 Ibid.
14 Refer to Greeff et al. 2017, 15.
15 Refer to Fahy 2008, 169; Milovanovic 2010, 192–199; Lewin 2012; Kearns 2012, 199; Morris 2013, 251–252; Bali 2014, 44–52; MacKenzie and Ballard 2015, 265–266.

- **Timely feedback:** Some feedback can be given fully automatic, hence virually in real-time.
- **Convenience:** Less intimidating; allows to try new ideas and make mistakes without public exposure.
- **Improved IT skills:** Technical literacy benefits from practice.

The central challenges, limitations and shortcomings, on the other hand, comprise:

- **High dropout rates**: Retention rates are often under 10%.
- **Individual requirements:** Requires a high degree of technical competence, autonomy, and strategic approach from students; lack of a physical human counterpart; some students are uncomfortable with computers.
- **Technical prerequisites:** Internet connection, bandwidth, electricity supply, etc.
- **Lack of understanding and readiness:** On the part of educators, most are unaware of the potentials of technology; reluctance to hand over more control to students; increased workload as instructors are challenged to convey their intentions accurately and provide appropriate feedback.
- **Lack of quality control:** Strong variations in content quality; alleged scarcity of high-quality teachers.
- **Cheating and false identities:** Especially grave if academic credit is offered.
- **Limited scope:** Certain content is not suitable for online delivery by nature.

Morris also somewhat unfoundedly criticizes that MOOCs allegedly suffer from too many students and too low interactivity.[16]

16 Refer to Morris 2013, 251–252.

Concluding, most studies end on a positive note and testify that the opportunities outweigh the challenges and e-learning fundamentally improves students' learning outcomes.[17] One meta-analysis of 50 studies comparing online with face-to-face education found that online students performed modestly better.[18] Some have argued that e-learning is especially efficient in relation to technical and programming-related content[19], which is of special interest in the context of this study.

2.1.4 Considerations on Course Design

When developing online courses, there are several things to keep in mind, which are closely related to the opportunities and challenges discussed above. Meaghan Lister analyzed 17 relevant publications from educational technology journals and condensed their knowledge into four important considerations[20]:

- **Course structure:** A comprehensible structure with clearly communicated expectations; including an orientation at the beginning.
- **Content presentation:** Diverse activities; ability to make choices, i.e. choose activities that are consistent with their learning interests and needs.
- **Collaboration and interaction:** Discussion forums, chat, email; both student-to-student interaction and student-instructor interaction valued; formal or informal; building community; collaborating on projects.
- **Timely feedback:** Instructor's monitoring and constructive feedback.

17 Refer to Lewin 2012; Gilbert 2015, 26–27; Nguyen 2015, 315–316; MacKenzie and Ballard 2015, 255–256; Bataev 2017.

18 Refer to MacKenzie and Ballard 2015, 263.

19 Refer to Milovanovic 2010, 197–198; Law et al. 2010, 219; Kearns 2012, 205; Stifterverband 2012; Elgamal et al. 2013, 30; Fang et al. 2014, 3419–3424.

20 Refer to Lister 2014, 671–678.

Especially the third point is one that many fellow researchers share—interaction in the shape of peer-to-peer and student-instructor communication is key in a learning environment.[21]

Davis et al. mostly emphasize that the learning outcomes should be at the center of the course creation and directly translated into course content.[22]

While there is broad agreement that the potential of modern technology should be embraced, some reviewers reiterate that pedagogical standards must not be compromised, regardless of the instructional medium employed[23] and that previous research should not be ignored in the enthusiasm for new tools[24].

There are plenty of resources that explain the creation of online courses in great detail, one of them being *Indiana University*'s exhaustive course on the topic.[25]

2.1.5 Accessibility in the Context of E-Learning

Among the population with disabilities, only very few succeed in completing their higher education. Numerous obstacles complicate their learning: lack of accessible facilities and equipment in institutions, inaccessible educational resources, transportation, residential housing, interpersonal challenges, etc.[26]

In this context, e-learning is a potential game-changer, as it can overcome most of the inclusion barriers. To accomplish this, however, educational platforms, as well as content, have to be developed with accessibility in mind. One could therefore even speak of a certain responsibility that differentiates e-learning from other web contents in terms of accessibility.

Tim Berners-Lee defines ACCESSIBILITY as follows:

> Accessibility is the art of ensuring that, to as large an extent as possible, facilities [...] are available to people whether or not they have impairments of one sort or another.[27]

Unfortunately, some studies imply that more often than not, e-learning platforms are hardly accessible to impaired learners, and subsequently fail to deliver on the promise of "education for all".[28] Betts et al demand that online learning requires "an institutional commitment to accessibility", as they have observed that the aspect often was an "afterthought" to the actual implementation of e-learning.[29]

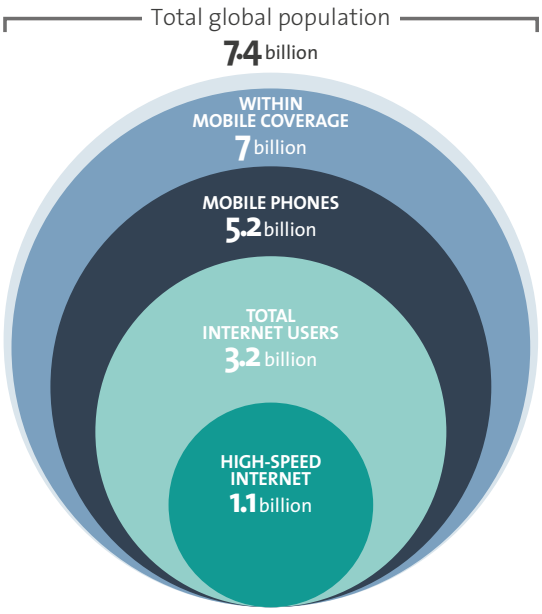


Figure 1 The internet remains unavailable, inaccessible, and unaffordable to a majority of the world's population.

21 Refer to Milovanovic 2010, 197–199; Lister 2014, 675–676; Croxton 2014, 314–322; Banna et al. 2015, 249–260; Cole et al. 2014, 113–126.

22 Refer to Davis et al. 2008, 125–126.

23 Refer to Caplan and Graham 2008, 262.

24 Refer to Fahy 2008, 169.

25 Refer to Indiana University n.d.

26 Refer to Betts, Riccobono, et al. 2013, 12; Sanchez-Gordon and Luján-Mora 2014, 530–531.

27 Berners-Lee 1999.

28 Refer to Ferati et al. 2016, 251–257.

29 Refer to Betts, Cohen, et al. 2013, 50–61.

In order not to discriminate or discourage potential e-learning participants, Kearns et al. propose a set of rules that can profoundly improve the usage of assistive technology like screen readers[30]:

- Provide uniform, predictable course layouts, with learning materials in consistent, logical places.
- Each course should include a syllabus page, offering students a comprehensive roadmap.
- Create a clear content structure, enriched with metadata.
- Follow the Web Content Accessibility Guidelines (WCAG).

Ferati et al. point out that it is feasible to design the e-learning platform so that the basic course templates are optimized for accessibility. As a result, all courses that later derive from them and even pre-existing courses would benefit.[31]

2.1.6 Offline Capability in the Context of E-Learning

With e-learning and MOOCs in particular, education is available regardless of geographical location. From a global perspective, this makes them especially interesting for developing countries like Africa. Most students in Africa complete primary education below expected literacy levels and consequently get dismissed from higher education. In fact, the majority of potential students is excluded by the education system.[32]

More than 8 in 10 Africans have a mobile phone, it is now the fastest growing mobile phone market in the world. In principle, the hardware for accessing e-learning is available. However, the network coverage varies and is limited to 2G in most regions. Other regions of the world are even worse: Out of the global population of 7.4 billion humans,

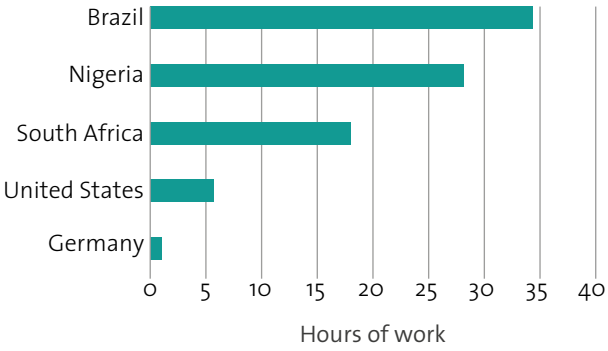


Figure 2 Hours of work required to afford a 500 MB entry-level mobile data package.

only 3.2 billion have some kind of access to the web (status as of 2015; Figure 1). Even where the internet is available, it is not necessarily affordable. Comparing Brazil with Germany, mobile data amounts to more than 34-fold (Figure 2).[33]

Hence, there is high demand for technical solutions. Native apps often work without connectivity, but they make for heavy downloads and entail regular updates—a 20 MB application will take more than 30 minutes to download via 2G, provided that the download does not fail half-way through because of connection issues. Also, installing an app occupies hard disk, which is much scarcer on cheaper phone models.[34]

The currently trending technology called Progressive web applications (PWAs) may tackle some of the issues at hand. PWAs are just websites at their core, which is why they are not very large in terms of file size. They have some enhanced features that make them feel and behave more like native applications. Most prominently, they employ special JavaScript files called Service workers to intercept network requests and implement highly customizable caching strategies. Another feature named Background sync records requests that happen while a device is offline and transmits them once connectivity is reestablished. For users in countries like Africa, this means that a temporary loss of connection is not fatal, in fact, it might go unnoticed.

30 Refer to Kearns et al. 2013, 78–82.

31 Refer to Ferati et al. 2016, 526.

32 Refer to Renz et al. 2017, 736–737.

33 Refer to Lawson 2017.

34 Refer to ibid.; Marco et al. 2015, 1199–1206.

A comparative study concerned with improving e-learning access in Africa reinforced that PWAs are a relevant improvement to the traditional HTML5 AppCache approach.[35]

2.2 Current Trends in Web Development

The technological landscape in web development is changing and evolving rapidly. In this section, an attempt is made to capture a snapshot of the current state—knowing that most observations on the tools and practices will soon be outdated.

One of the most valuable resources in this regard is a recent survey conducted by Greif, Benitte and Rambeau, collecting data from more than 23,000 web developers. Their annual survey is somewhat misleadingly titled *The State of JavaScript*, when really it covers a much broader range.

2.2.1 JavaScript Language Flavors

By now, the *ECMAScript 6* (*ES6*)[36] standard is widely established and used approximately as much as "Plain" JavaScript, meaning the old *ES5* standard. The developers' acceptance rates of the new standard are much higher however. Also, the typed JavaScript-superset *TypeScript*, that offers static type-checking additionally to *ES6* features, is on the rise.[37]

2.2.2 Front-end Frameworks

React.js (mostly just referred to as *React*) is clearly dominating front-end frameworks[38], but *Vue.js* (or just *Vue*) is making big gains that may very well be connected to *Angular's* (mostly indicating version 2 or higher, in contrast to *AngularJS* referring to version 1.x) diminishing popularity. According to Greif, *Angular* is shifting to a new role and instead of trying to compete with *React* head-to-head refocusing itself on the enterprise market.[39] Other observers are even more explicit in their verdict: According to Maximilian Stroh, "Angular has already lost to React no matter how hard Google tries to hide it"[40].

Considering that *Vue* is a rather young framework, its rise in significance is impressive—it has established itself as "the biggest threat to React's crown"[41]. While *Polymer* does appear on the list, it is one of the lesser known and used frameworks.

2.2.3 State Management Tools

On the client, *Redux* is the most used solution for managing data, with *VueX* also being mentioned frequently. *Google*'s *Firebase* gained a lot of attention as well.

While traditional REST APIs are still used almost exclusively to transfer data between server and client, an alternative technology called *GraphQL* and related tools like *Apollo* generate a lot of curiosity in the community.[42]

2.2.4 Back-end Frameworks

Speaking of back-end technologies, *Express* is the most important one by far. *Meteor* raised some awareness but does not seem to be able to capitalize on it. *Node.js* has also gotten many mentions, whereas traditional programming languages like *Java*, *PHP* or *Python* seem to lose ground.[43]

Other than that, *Docker* has revolutionized the back-end development. Simply put, it allows for software to be configured once, and subsequently deployed as often as needed, within a controlled

35 Refer to Renz et al. 2017, 739–740.

36 There has been some confusion about how to call this JavaScript version. The terms ES6 and ES2015 are often used interchangeably. In this study, the term "ES6" shall be used.

37 Refer to Greif et al. 2017.

38 Technically, React as well as Vue are libraries rather than frameworks. Most articles do not bother with this differentiation though.

39 Refer to Greif 2017.

40 Stroh 2017.

41 Greif 2017.

42 Refer to Greif et al. 2017.

43 Refer to ibid.

Figure 3 Survey results from "The State of JavaScript 2017" illustrating the level of awareness and satisfaction of technologies.

environment and with repeatable parameters. With *Docker-compose*, complex systems of containerized services can be orchestrated. *Docker Swarm* and *Kubernetes* add the ability to dynamically scale up and down on demand. The next trend in terms of solving back-end problems seems to be going "serverless". Bare functions get deployed to cloud providers like *Firebase* or *Amazon Web Services* (AWS), so that clients can simply accesses them via the given URL.[44]

2.2.5 Testing Tools

The JavaScript testing landscape is dominated by *Mocha* and *Jasmine*, followed by *Jest* and *Enyme*. The two latter ones stand out because of their very high satisfaction ratings.[45]

2.2.6 CSS and Styling

Apart from plain CSS, which many developers still use, the most prevalent CSS libraries in descending order are: *Sass* (short for "Syntactically awesome stylesheets"), *Bootstrap* and *Less*. Out of these, *Sass* has the highest satisfaction rate with 89%.[46]

2.2.7 Build Tools

NPM (short for "Node Package Manager") and *Webpack* are the most frequently used and most satisfying build tools. Other widespread technologies like *Gulp*, *Browserify* or *Grunt* have much lower acceptance rates.[47]

2.2.8 Mobile and Desktop Frameworks

Native apps are still the most reliable mobile solution, even though *React Native* attracts a lot of attention. PWAs are among the top mentions as well.

2.2.9 Features

Modern features that developers value in JavaScript apps include:

- Server-side rendering
- Code splitting
- Optimistic updates
- Hot module reloading
- Time-travel debugging
- Real-time operations
- Dead code elimination
- Progressive enhancement
- Service workers
- Offline usage

Admittedly, the median for every single one of these features is a mere "Nice-to-have, but not important".[48]

2.3 Comparison and Evaluation of Relevant JavaScript Frameworks

As a basis for the decision which framework to use for the implementation—if any—the three most important potential candidates *Angular*, *Vue* and *React* will get described and evaluated. The metrics are mostly inspired by Neuhaus' comparison.[49]

44 Refer to Stroh 2017.

45 Refer to Greif et al. 2017.

46 Refer to ibid.

47 Refer to ibid.

48 Ibid.

49 Refer to Neuhaus 2017.

2.3.1 Considerations on the Cost of Frameworks

There is no doubt that frameworks make it easier and more convenient to quickly build web applications. They help to work around inconsistencies and bugs of the platform. Not to mention that proficiency in popular frameworks helps developers to become more attractive on the job market. As Lewis points out however, the convenience of the developer often contrasts with the needs of the user, e.g. a quick page load. Web applications that have been built with the use of frameworks tend to take a long time specifically at the first load. Even if you leave aside the transfer time, it takes a while for JavaScript to bootstrap the framework and build the initial view (Figure 4).

Other than time and bandwidth, users have to pay in the currency of CPU usage (meaning battery consumption), memory usage and potentially frame rate. There are also some costs for developers involved: They have to learn the framework, refresh their knowledge after each major update, and debug it. On top of that, including a framework in an application has deep implications: It entails an inversion of control, as the framework now controls the app's lifecycle and offers the developer entry points to run code. Unlike a library, exchanging a framework is difficult if not impossible.

A series of extensive performance tests showed that plain JavaScript is significantly faster than any framework, making framework overhead especially expensive on mobile. Lewis concludes that the decision whether or not to use a framework is important, hence not be taken lightly. His tendency clearly is to do without.[50]

Eugeniya Korotya disagrees and praises the advantages of using JavaScript frameworks. Besides efficiency and safety, she points out that in terms of money, applications actually become *cheaper*, as frameworks enable developers to work faster.[51]

2.3.2 Notes on the Absence of Web Components

As has been addressed in 2.2.2, *Polymer* (which strictly speaking ranks among libraries) and web components in general only play a minor role in today's framework landscape. And this seems to be the beginning of the problem: "The main failure is obvious: they are nowhere to be seen."[52]

Browser support is stagnant, developers—after a wave of excitement when the idea of web components first surfaced—are not overly interested, Polymer does not appear in any of the articles that

50 Refer to Lewis 2015b.

51 Korotya 2017.

52 Dimandt 2017.

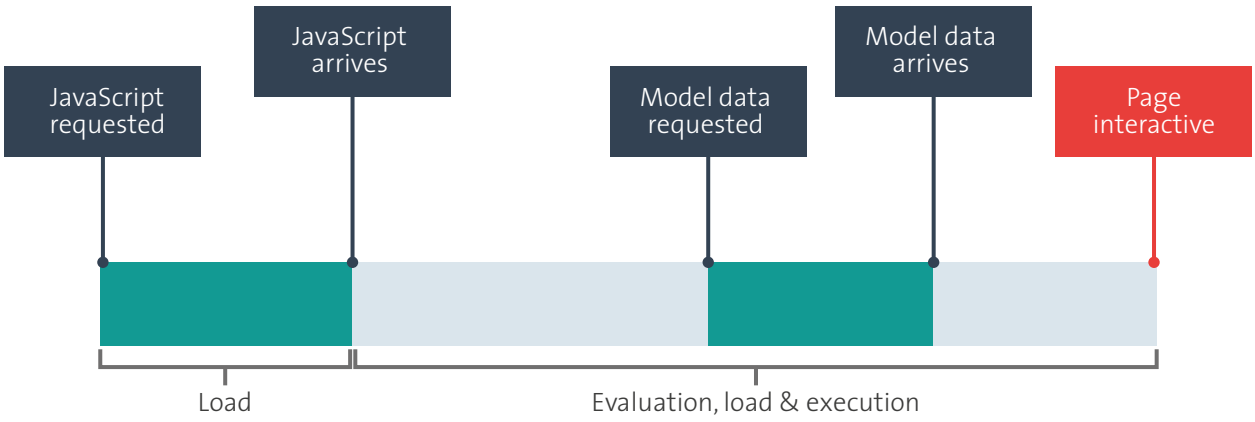


Figure 4 Time to interactive: Measuring how long it takes for an app to get to a state where someone can interact with it.

discuss which framework to choose. According to Dimandt and Rauschmayer, these unfortunate developments are mostly caused by the *W3C* (short for "World Wide Web Consortium") and tedious standardization endeavors.[53]

Moreover, in 2013, *React* entered the stage, to be followed by other front-end frameworks that would all offer the same features web components promised, without the incompatibility issues, the standardization baggage, and the restrictions web components impose.[54] [55]

Indeed, from a developer's point of view, frameworks like *React*, *Angular* or *Vue* offer exactly the experience that web components promise. During development, working with custom, reusable, self-contained components has become commonplace. The transpiled code that gets shipped to the browser consists of regular, compliant HTML and JavaScript—but that's just fine in a large majority of cases.

On a side note: Developing with a framework and using web components are not mutually exclusive. Like native DOM (Document object model) elements, web components can e.g. be used as leaves in the tree of *React* components.[56] In fact, there are even ways to turn e.g. *Vue* custom components into "real" web components.[57]

2.3.3 Lifecycle and Strategic Considerations

Angular is a framework developed and maintained by Google with the most recent major release being version 5.

React is a JavaScript library, but it commonly gets classified among frameworks. It is built by *Facebook*; the most recent release is *React* 16.

Vue is different from its competitors in that it is not backed by a big company, but essentially developed by one programmer, Evan You, supported by a group of programmers and sponsors. In 2016 and 2017, the reach and significance of *Vue* has grown remarkably. In the last quarter of '17, version 2.5 has been released.

After *React* had shipped with a proprietary license for a long time, it is now available under the *MIT* license, like its two competitors. Reviewers agree that *React* and *Angular* are future-proof, while there is a small amount of uncertainty concerning *Vue*.[58]

React APIs have been quite stable, and updates seldomly caused a problem. With *Vue*, things look similar. After some breaking changes between version 1 and 2 of *Angular*, which brought a lot of critique onto the framework, it has been mostly stable.

2.3.4 Prevalence and Satisfaction

The level of awareness and acceptance has already been presented in 2.2.2.

In terms of *GitHub* popularity, *React* is the most starred repository, while *Vue* boasts the highest ascent, and is very likely going to take the lead soon. *Angular* does not seem to attract as much sympathy (Figure 5; Table 1). This is also mirrored in the survey results discussed in 2.2.2—While *Vue*'s reach is smaller, the satisfaction rate is very high.

53 Refer to ibid.; Rauschmayer 2015.

54 Web components are DOM, so they have to work with the questionable DOM APIs, attributes have to be strings (which means e.g. commas must be escaped with a backslash), etc. Libraries like Polymer try to find their ways around this, introducing the next problem as their solutions are not compatible with one another.

55 Refer to Dimandt 2017.

56 Refer to Rauschmayer 2015.

57 Refer to Tarchichi n.d.

58 Refer to Sidorenko 2017.

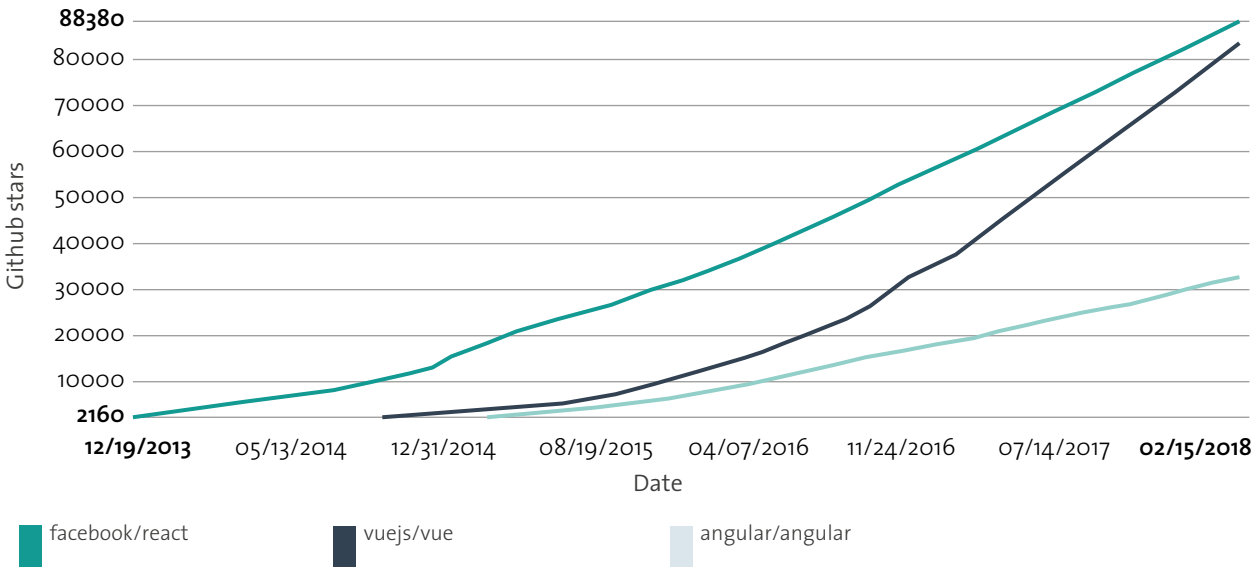


Figure 5 GitHub star history comparing the popularity of React, Angular and Vue.

	Total stars	Stars per day	Contributors
React	88,381	82.2	1,167
Angular	33,066	34.3	591
Vue	83,779	113.5	167

Table 1 GitHub pupularity of React, Angular and Vue.

2.3.5 Enterprise User Base

React boasts a large user base, incl. *Airbnb, Uber, Netflix, Twitter, Pinterest, Reddit, Udemy, Wix, Paypal, Imgur, Feedly, Stripe, Tumblr, Walmart*, and others. Also, *Facebook*, *Instagram* and *WhatsApp* use it on their own pages.[59]

Google uses *Angular* in some of its own projects. While some other known companies used the original *AngularJS*, it is hard to find companies that have committed to *Angular 2* or higher.

There are also few major companies using *Vue* so far, those mostly being *GitLab, Adobe, IBM, Nintendo, JSFiddle* and *Codeship*. Also, multiple large Chinese tech companies incl. *Alibaba* and *Baidu* have adopted the framework.[60]
In regards of *NPM* package downloads, *React* dominates by far. In *Google Trends*, *Vue* lags behind.

2.3.6 Typed vs. Untyped

TypeScript support is deeply built into *Angular*. While this does offer many benefits, it also comes with some learning overhead. Type-checking addons are available for *Vue* and *React* as well, but they are not native to them. This makes *Angular* a good fit for enterprise-based applications with high security requirements.

2.3.7 JSX vs. HTML

React does not separate template and logic, it combines them in an HTML-like language called *JSX* (short for "JavaScript Syntax eXtension"). *Angular* templates are enhanced HTML. *Vue* takes middle ground—its components are in one file, but separated areas: template, script and style.

59 Refer to Neuhaus 2017.

60 Refer to Greif 2017.

2.3.8 Virtual DOM

Virtual DOM is a concept first popularized by *React*. It is a second, lightweight layer of abstraction on top of the DOM, which itself is an in-memory representation of the HTML. Unlike the native DOM, updating the virtual JavaScript nodes is not very expensive in terms of performance. The framework then patches the real DOM efficiently (Figure 6).[61]

In version 2.0, *Vue* also introduced a virtual DOM. *Angular* does not have this concept.

2.3.9 Framework vs. Library

Angular is a full-blown "batteries included" framework. With no further setup or dependencies required, developers can simply start coding. Then again, it brings a lot of overhead and complexity. *React* and *Vue* in contrast are flexible, their core libraries are leaner, and they are typically combined with various custom packages.

2.3.10 State Management, Data Binding and Dependency Injection

Given that applications are large enough, *React* is typically used with Redux or Mobx, while *Vue* is used with *Vuex*. *Angular* and *Vue* (if used without *Vuex*) offer two-way data binding, while *React* implements the idea of unidirectional data flow. Both concepts have advantages and disadvantages. Additionally, *Angular* includes dependency injection, making for clean and flexible code.[62]

2.3.11 Flexibility and Downsizing

React and *Vue* allow to select only the necessary parts, therefore offering more flexibility in creating small bundles. They can also be progressively worked into an existing page. This is not possible with *Angular* because of its holistic philosophy and its use of *TypeScript*.

E.g., when *Adobe* migrated the pre-existing code base of *Behance* and *Adobe Portfolio* to *Vue*, the ability to integrate into their page instead of do-

61 Refer to Krajka 2015.

62 Refer to Neuhaus 2017.

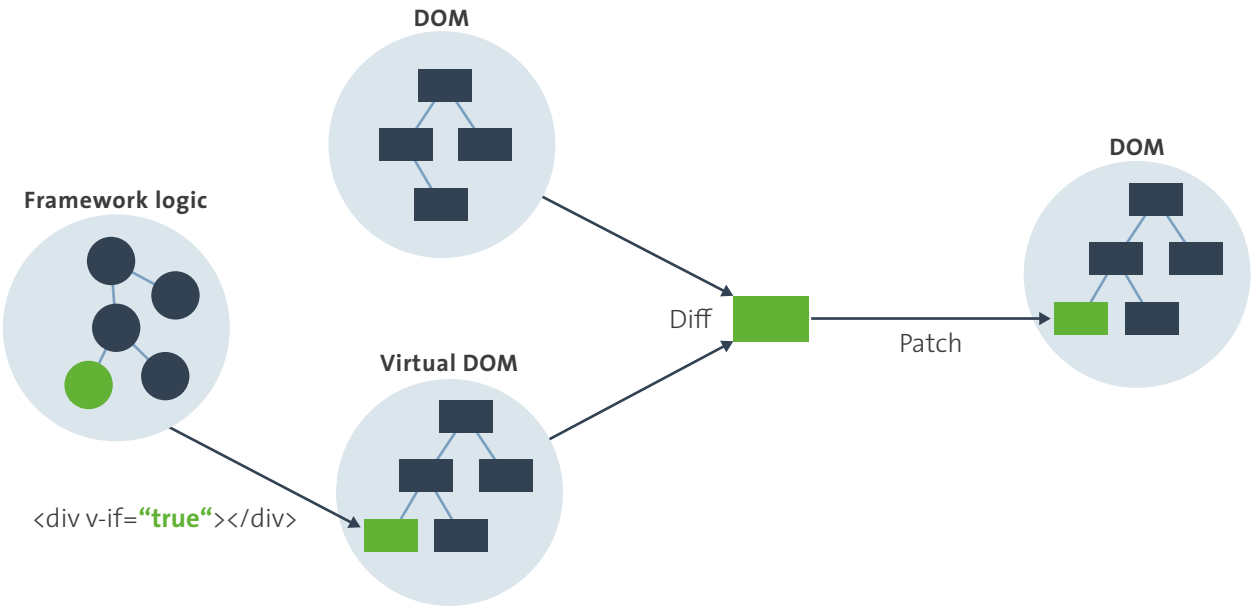


Figure 6 Efficient updates of the native DOM by computing the "diff" to the virtual DOM .

ing it from scratch allowed for an unhurried pace and was considered a major advantage.[63]

2.3.12 Size and Performance

Angular's feature-richness takes its toll when it comes to file size: The compressed file is more than five times as large as the respective *Vue* bundle. *React*'s footprint is about twice the size of *Vue*.

Even though *Angular* does not implement a virtual DOM, performance benchmarks show that all three are pretty close to each other. Neuhaus warns that benchmarks are not very accurate and thus should not be the decisive criterion.[64]

2.3.13 Testing

All frameworks offer ways and tools to test their components. However, *Vue* lacks the guidance of its competitors. This will be improved soon, says creator Evan You.[65]

2.3.14 Universal and Native Apps

Unlike *Vue*, *React* and *Angular* support native development. This lack of a native solution in *Vue* is one of the biggest pain points and has been mentioned by more than 24% of participants in a *Vue*-related survey.[66]

For each of the three frameworks, server-side rendering (SSR) libraries are available.

2.3.15 Learning Curve

Angular's learning curve is the steepest. A lot of elusive setup is involved; the huge stack of features and learning discourage many developers, not to mention *TypeScript*. Learning *React* involves learning the *JSX* language and choosing among large amounts of third-party libraries; still, most reviewers agree that it is still easier than *Angular*.

Vue, on the other hand, is undoubtedly the easiest to start with. In a survey among *Vue* developers, the most important reason behind adding *Vue* to their stack by far was that it is "pretty easy to start with"[67].

For some companies, this was the major decision criterion:

> Angular, with all its structure, modules, models, and controllers, and dozens of other things [...] introduces an unnecessarily high level of complexity. For [engineers who rarely do front-end work], most of it sounds like weird magic spells. But when they actually saw Vue.js, they felt empowered to dig into it right away.[68]

2.4 Authentication for Single Page Applications

With the rise of SINGLE-PAGE APPLICATIONS (SPA), web applications tend to become more independent from back-ends. This trend goes hand in hand with a shift in authentication paradigms. Token-based authentication replaced the traditional cookie-based mechanisms in many places. Other trends that have influenced this evolution are web APIs and the INTERNET OF THINGS (IoT)[69].

In this section, both approaches will be discussed and compared (Figure 7).

2.4.1 Comparing Cookie-Based with Token-Based Authentication

Cookie-based authentication is the traditional approach. It is field-proven and has been the default for a long time. The most characteristic attribute of this approach is that it is stateful. Both server and client have to keep track of a session while a client is logged in. The process steps are as follows:

63 Refer to Monterail and You 2017, 32, 38.

64 Refer to Neuhaus 2017.

65 Refer to ibid.

66 Refer to Monterail and You 2017, 22.

67 Refer to ibid., 18.

68 Refer to ibid., 58.

69 Refer to Kukic 2016.

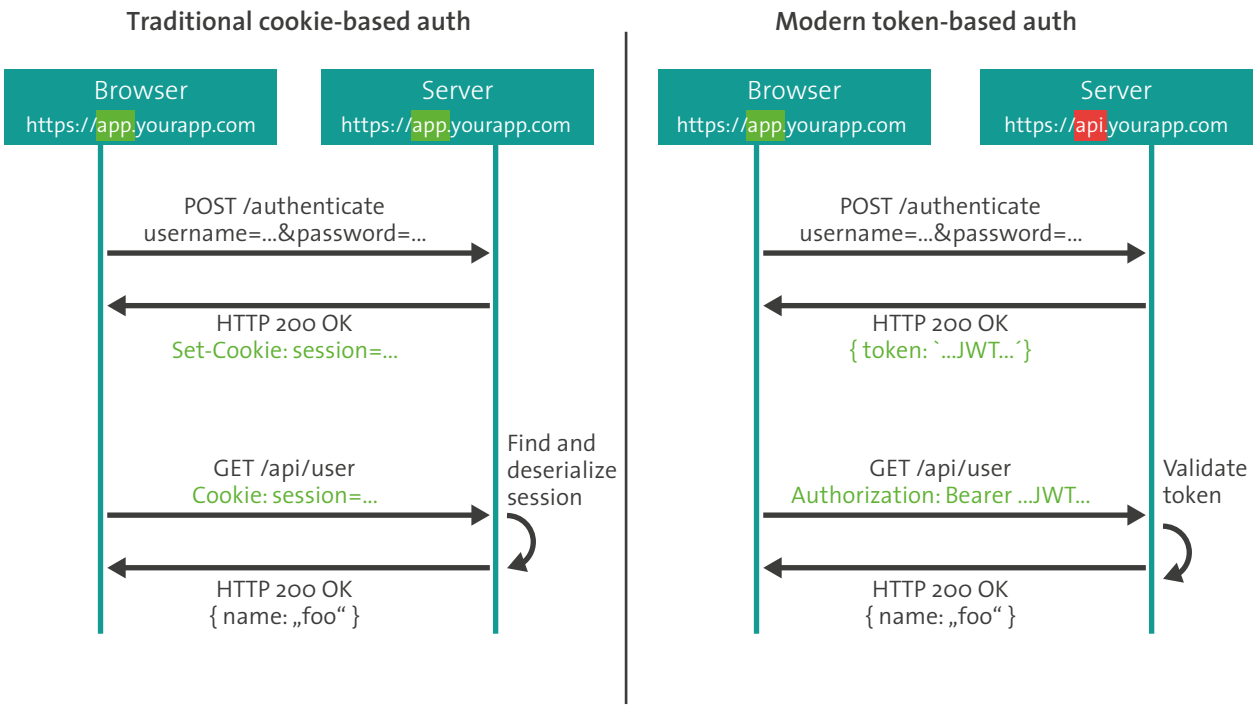


Figure 7 The difference between cookie-based and token-based approaches to authentication.

1. [The] user enters their login credentials.
2. [The] server verifies the credentials are correct and creates a session which is then stored in a database.
3. A cookie with the session ID is placed in the user's browser.
4. On subsequent requests, the session ID is verified against the database and if valid the request processed.
5. Once a user logs out of the app, the session is destroyed both [on] client and server side.[70]

When this technique is in place, it is crucial to use HTTPS. Additionally, it is recommended to implement a Cross-site request forgery (CSRF) mitigation strategy.[71]

Token-based authentication, on the contrary, is stateless, hence it inherently goes together with SPAs and RESTful APIs better. The server does not keep record of which users are logged in, nor of which tokens have been issued. A token is included in the headers of each HTTP request, and the server statically verifies the authenticity of the request. This comprises the following steps:

1. [The] user enters their login credentials.
2. [The] server verifies the credentials are correct and returns a signed token.
3. This token is stored client-side, most commonly in local storage - but can be stored in session storage or a cookie as well.
4. Subsequent requests to the server include this token as an additional Authorization header [...]
5. The server decodes the [token] and if [it] is valid processes the request.

70 Ibid.

71 Damphousse 2015.

6. Once a user logs out, the token is destroyed client-side, no interaction with the server is necessary.[72]

Token-based authentication helps to decouple front-end from back-end. There are several advantages: Each token is self-contained, the server does not need to keep track, it merely needs to sign tokens on login and verify the validity of tokens. Also, the new approach facilitates cross-domain data exchange as is typical for web APIs. What's more, additional data like the username can be directly included in a token as it is a JSON object. This is often used to store the user's permission level, resulting in simplified server code. The server-side performance generally improves, as the data embedded in the token removes the need for database lookup. Lastly, tokens are easier to implement in native mobile apps and IoT devices.[73]

Technically, tokens can be stored as cookies, in the session storage or the local storage of the browser. Common practice is storing them in the local storage, even though it is restricted to the current domain[74]. The benefits outweigh this restriction: Most importantly, there is no file size limitation as with cookies, and CSRF gets prevented inherently. This, however, does not change the need for a secured connection.[75]

2.4.2 JSON Web Tokens

Even though there are different possibilities to implement tokens, *JSON Web Tokens* (*JWT*) are the de-facto standard and are generally accepted as the best solution hitherto.[76] They offer a structured way to declare who a user is, and, through defining scopes, what she is allowed to access. With the help of a token, a third party can validate the sender's identity and decide whether she is authorized to perform a specific action.

Figure 8 An illustration of the strings in a JSON Web Token.

A signed *JWT* is a string that gets embedded in the HTTP request header. It consists of three parts separated by dots: *JWT* Info, *JWT* data, and signature (Figure 8).

- The first part contains information about the *JWT* itself, like which algorithm was used to encrypt the third part.
- The second part contains the actual data, providing information about the sender, her authorization and the expiration date of the token.
- The First and second parts are technically JSON files that have been base64 encoded, making them publicly readable. The combination of the two, including the separating dot, can be thought of as "unsigned token".
- The third string is the result of taking the unsigned token and encrypting it with the ES256 algorithm.

The recipient of a request has to get the sender's public key to verify the validity of the signed *JWT*.[77]

As always with cryptography and authentication mechanisms, it is possible to implement the algorithms manually, yet strongly discouraged. Instead, it is common practice to leave the low-level work to an established *JWT* library.

72 Kukic 2016.

73 Refer to ibid.

74 Local storage is sandboxed to one domain, meaning that it cannot be accessed even by sub-domains.

75 Refer to Kukic 2016; Damphousse 2015.

76 Refer to Damphousse 2015.

77 Refer to Gaunt 2018.

2.4.3 Future Solutions to Online Learning Integrity

As the role of online education and academic integrity gradually become more important, more accurate solutions for user authentication will be needed in the medium-term. There is some research in this respect that looks into emerging technologies like biometrics.[78]

2.5 Centralized State Management in Web Applications

Lately, a trend can be observed in regard to state management in web applications towards using global data stores. Most famously, *Facebook* has branded this pattern as Flux architecture. The story goes that the engineers at *Facebook* had trouble with the unread messages count in the notification bar. Due to the complexity of their system, its state was hard to keep under control. Unpredictable results were caused by its dependencies, interconnections and cascading updates of shared state.[79]

The solution they came up with was to eschew traditional *Model–view–controller* (MVC) in favor of a unidirectional data flow with all state encapsulated in a singleton data store (Figure 9). This allowed for highly decoupled components. To request changes, a component has to dispatch an action, that the store will then process:

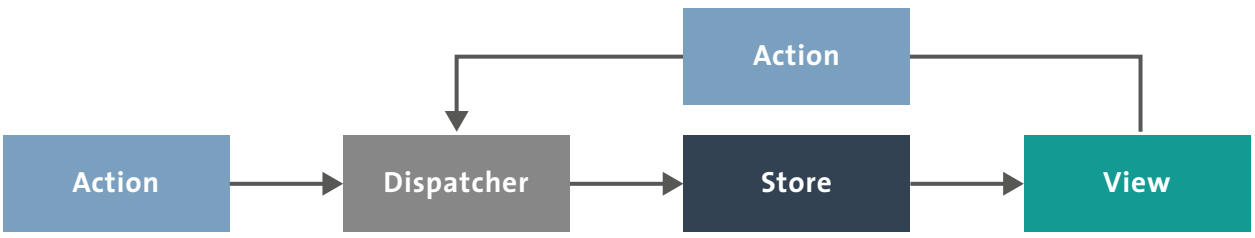


Figure 9 Unidirectional data flow in Flux.

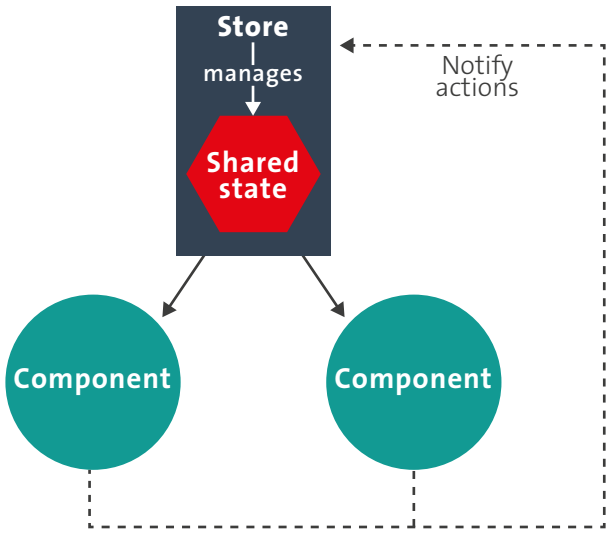


Figure 10 General model of centralized state management.

> Control is inverted with stores: the stores accept updates and reconcile them as appropriate, rather than depending on something external to update [their] data in a consistent way. Nothing outside the store has any insight into how it manages the data for its domain, helping to keep a clear separation of concerns. Stores have no direct setter methods, but instead only a single way of getting new data into their self-contained world — the callback they register with the dispatcher.[80]

Multiple implementations, slight variations and framework-tailored libraries have been built since, most notably *Redux* and *Vuex*. They enforce certain principles to keep the state in a transparent and predictable state even when shared across multiple components (Figure 10). Nevertheless, they also add a level of indirection, more concepts

78 Refer to Lee-Post and Hapke 2017, 135–145.

79 Refer to Gore 2017b.

80 Facebook 2014.

for developers to understand and some boilerplate code. It is a "trade-off between short term and long term productivity"[81], and for smaller applications, a global event bus might be sufficient.

2.6 User Experience

User Experience, often abbreviated as UX, is a term frequently used, but not that easy to grasp. In this section, an attempt is made to illustrate the aspects that make for user experience, and how they are interconnected.

2.6.1 Concepts and Methodology

The terms *user experience* and *usability* often get mixed up. Usability is a subset, it is one element of user experience and plays an important role for all human-machine interfaces incl. software, websites, and mobile apps. It is based on objective criteria and can be measured, as defined by the *DIN* (Deutsches Institut für Normung):

> Usability is the extent to which a product is used by a specific audience for the purpose for which it has been designed with effectiveness, efficiency, and satisfaction.[82]

Notably, good usability is mostly imperceptible, while usability shortcomings are not. According to Steven Krug, the most important factor of usability is not to make the user *think*—what an element is and how it is used should always be self-evident.[83]

User experience, on the other hand, takes the concept of usability and extends it with aesthetical and emotional factors. It is a holistic approach that covers the complete experience before, during and after interfacing with a product. Hence, it is inherently more difficult to find objective measures. The *DIN* offers a definition nonetheless:

> A person's perceptions and responses that result from the use and/or anticipated use of a product, system or service.[84]

Peter Morville's *User Experience Honeycomb* model offers a good overview of the facets of UX (Figure 11). It argues that in order to be meaningful and valuable, information must be:

- **Useful:** Your content should be original and fulfill a need.
- **Usable:** Site must be easy to use.
- **Desirable:** Image, identity, brand, and other design elements are used to evoke emotion and appreciation.
- **Findable:** Content needs to be navigable and locatable onsite and offsite.
- **Accessible:** Content needs to be accessible to people with disabilities.
- **Credible:** Users must trust and believe what you tell them.[85]

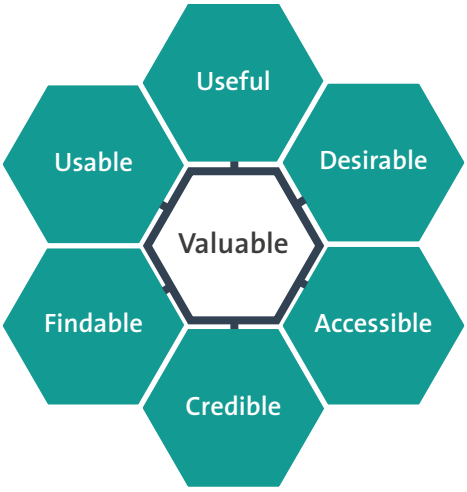


Figure 11 Facets of the User Experience according to Morville's User Experience Honeycomb.

81 Vuex Documentation n.d.

82 DIN Deutsches Institut für Normung 2017.

83 Refer to Krug 2013, 11.

84 DIN Deutsches Institut für Normung 2011.

85 U.S. Department of Health & Human Services 2014; Morville 2004 [emphasis in original].

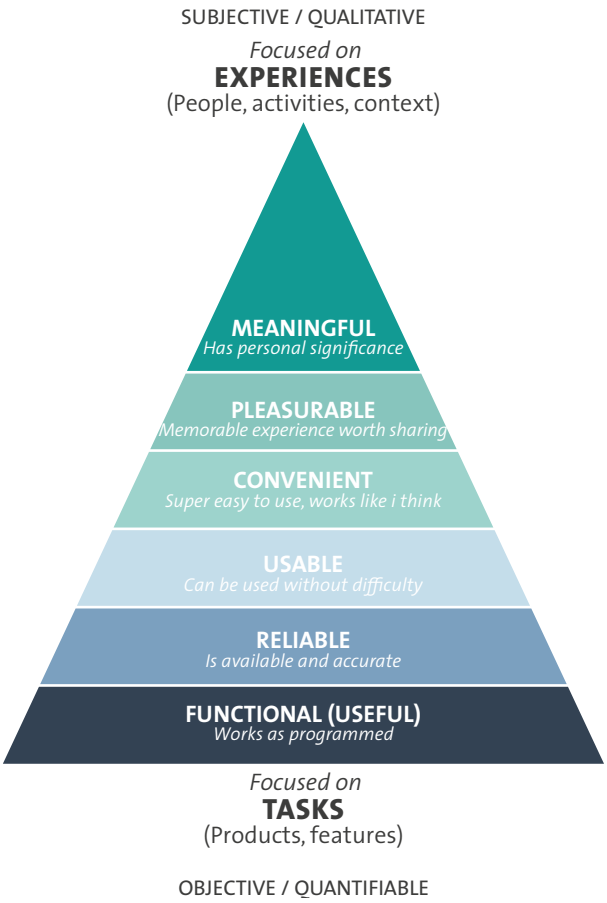


Figure 12 Anderson's User Experience Hierarchy of Needs model.

Another useful model to evaluate the quality of user experiences is Stephen Anderson's *Hierarchy of needs*. From bottom to top, it shows a basic product maturity continuum, starting with bare functionality, and reaching up to a meaningful and fun user experience (Figure 12).[86]

2.6.2 Relevance in the Context of E-Learning

User experience is an important factor in the success of *all* digital products. It determines how long users will stay in an application, how they perceive a brand, and whether or not they recommend it to others. Subconsciously, it is often put on a level with the application itself.[87]

What's more, research has proved time and again that UX (and usability in particular) play an important role in the context of e-learning.[88] Randomized trials showed that improvements in an e-learning platform yielded in improved motivation an engagement with the resource and increased the chances of achieving the desired educational outcomes.[89]

Patti Shank has researched the frustrations that users experience with online learning and consequently identified two specific types of usability[90]:

1. Technical usability, which deals with minimizing system-related frustrations.
   - Simple, consistent navigation scheme.
   - Media optimized for quick download.
   - Courses designed to function similar to each other.
   - Commonly used functionality easily and quickly accessible.
   - Etc.
2. Learning usability, which deals with minimizing the learning-related frustrations.
   - Managing expectations.
   - Making help available.
   - Providing reality checks.
   - Etc.

Other authors also use the term pedagogical usability which overlaps with the given defi-

86 Refer to Anderson 2011, 11–13.

87 Refer to U.S. Department of Health & Human Services 2014.

88 Refer to BCIT Learning and Teaching Centre 2001, 2; Fahy 2008, 189–190; Shank 2009; Jeffels 2011, 1–3; Lehr 2012; Lane and Hull 2013, 24–25; Davids et al. 2014, 155–158; Kiget et al. 2014, 98–101; López 2016; Herald and Thorpe 2016.

89 Refer to Davids et al. 2014, 159.

90 Refer to Shank 2009.

nition of learning usability. It denotes "whether the tools, content, interface and the tasks of the web-based learning environments support various learners to learn in various learning contexts according to selected pedagogical objectives"[91].

Shank concludes—and many of her fellow researchers come to a similar conclusion—that improving usability of all nuances creates some extra work on top of designing content, activities and assessments, but eventually is necessary "since the negative outcomes from poor usability end up in the instructors' and students' laps"[92].

2.6.3 Design, Layout and Aesthetics

> Aesthetics include everything that appeals to the senses—not just what we see, but also what we hear, smell, taste, and feel. [...] In the digital realm, this includes visual design, motion, sound, and even haptic (tactile) responses.[93]

Cognitive science studies have confirmed that aesthetics communicate function, help the users to make associations and inform them how to respond to certain elements. Knowledge of past experiences or general world knowledge subconsciously influence visual comprehension: "If it looks like a button, it must be a button". Elements like shadows communicate three-dimensionality and order; contrast and connectedness form logical units; shared characteristics imply associations; color codes are implicitly understood, e.g. a success message with red letters will put the user off.[94]

On top of that, these associations affect how we feel about a product, and consequently how we will behave. Research on how emotions influence interactions has shown that enjoyable things will also be perceived as easy to use and convenient.[95]

This makes visual design an important tool in respect of the UX aspects desirability and usability.

Within 3.42 seconds on average subjects will judge the credibility of a website, based mostly on factors like layout, typography, font size, and color scheme.[96] Subconsciously, users trust applications more if attention has been put to the details, whereas visual inconsistencies fail to convince of the content's value.[97]

Visually elaborate content, however, is often under suspicion of hiding poor scholarship or argument.[98] Language commonly used in the technology field like "eye candy" and "styling" according to Anderson serves to limit the importance of visual design and separate aesthetics from usability.[99] David and Glore in contrast argue that not only are design, aesthetics and usability inexorably linked, they have a significant impact on usability, credibility and learning success in e-learning environments. Their findings suggest that "the aesthetics of a course, particularly the layout, the use of graphics, and the ease of use, were important in motivating [students] to engage and persist in web-based learning". Therefore, they encourage educators to consider aesthetic qualities when developing course materials and e-learning platforms. They also point out that more often than not, due to limited institutional resources and support, "elements that are considered nice to have but not necessary, like aesthetics, are neglected or implemented less than professionally"[100].

Anderson warns not to treat visual design as decoration that can be added ex-post. Aesthetic choices appropriate to one's audience and business goals, respecting common design conventions and aesthetic suggestions, should be made throughout the development.[101] In regard to visual

91 Jeffels 2011, 1.
92 Shank 2009.
93 Anderson 2011, 18.
94 Refer to ibid., 17–24.
95 Refer to ibid., 25–34; David and Glore 2010.
96 Refer to David and Glore 2010.
97 Refer to Soueidan 2013.
98 Refer to Lynch 2009.
99 Refer to Anderson 2009.
100 David and Glore 2010.
101 Refer to Anderson 2011, 51.

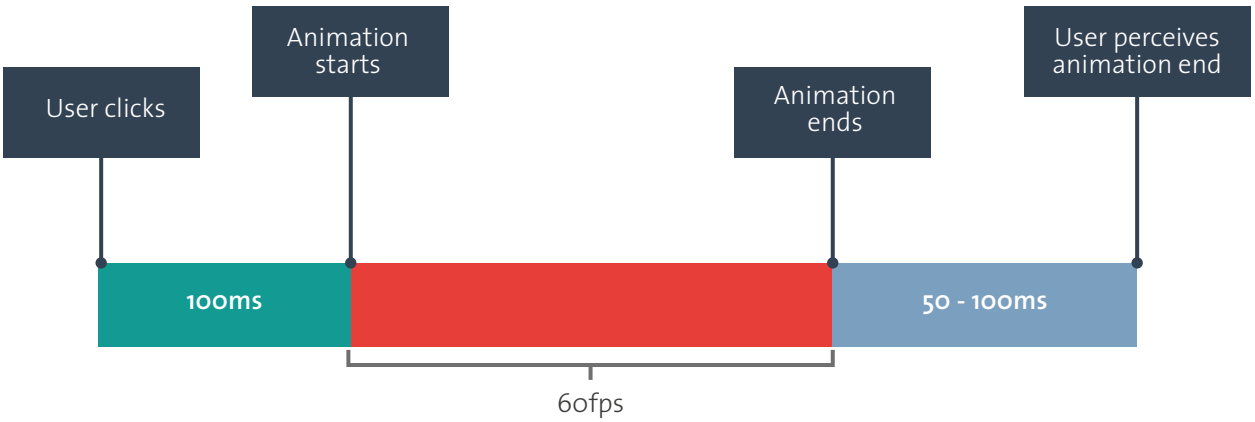


Figure 13 User perception of "instant" responses when interacting with a website.

design, the *DIN* norm stresses the factors compactness, consistency, recognizability, readability and comprehensibility.[102] Fahy offers some tangible advice on the use of color, fonts, textures etc. in support of e-learning success.[103]

2.6.4 Animations and Transitions

In the context of web development, *animating* typically denotes changing particular CSS properties over time, which can either be done natively in CSS or manually, using JavaScript.

Animations, according to *Google*'s Paul Lewis, are a huge part of making compelling web applications and adding life to projects. He points out that animations should support and reinforce interaction, e.g. add a subtle glow or bounce to elements with which the user currently interacts. He also encourages animating between views, while sticking to short durations. In terms of the speed curve, he recommends the CSS ease-out timing function as it mimics natural motion the closest.[104]

Lewis also elaborates on the cost of animating certain properties. Developers should stick to changing transform and opacity, where feasible, as these changes can oftentimes be handled by the so-called *compositor thread*, which differs from the browser's primary thread. If an animation triggers paint, layout, or both, however, the "main thread" will do the work. This has impacts on performance and frame rate because the thread jointly takes care of styling, layout, painting, and JavaScript execution.[105]

In order to always maintain 60fps[106], Lewis devised a technique called FLIP (short for "First, Last, Invert, Play"). It is based on the observation that users still perceive a reaction as immediate if it starts within 100ms at most (Figure 13). This timeframe can be used to do the heavy-lifting beforehand: The initial and final state of the animated elements get computed and a transform gets applied, while cleverly suppressing it from actually changing the rendered output immediately. The animation itself hereby gets pre-computed and, if possible, remapped to transform and opacity changes. As a consequence, only the composite layer gets animated and animations play out smoothly.[107]

102 Refer to DIN Deutsches Institut für Normung 2000.

103 Refer to Fahy 2008, 175–177.

104 Refer to Lewis 2018a; Lewis 2018b; Lewis 2018c.

105 Refer to Lewis and Thorogood 2018.

106 Modern browsers refresh content in sync with the device's refresh rate, which for most screens is 60Hz. Hence, a browser should yield 60 frames per second.

107 Refer to Lewis 2015a.

2.6.5 Information Architecture and Content Structure

Information architecture is "the structural design of shared information environments"[108]. It deals with identifying "information atoms", organizing into categories, and naming these categories. According to Morville and Rosenfeld, it relies mostly on understanding the intersection of context, content and users (Figure 14).[109]

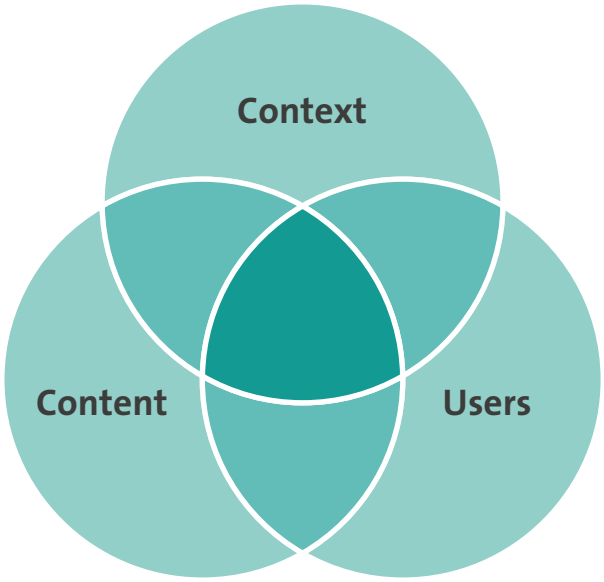


Figure 14 The three circles of information architecture.

In terms of presenting chunks of information, an important and often-cited rule is *Miller's law*. In his 1956 paper, cognitive psychologist George A. Miller argues that the number of elements humans can hold in working memory is 7 ± 2. As for information organization, this implies breaking up items into chunks of at most nine elements.[110] Some authors recommend limiting the number of items to as little as five, where feasible.[111] Similar in spirit, *Hick's law* states that the time it takes to make a decision increases with the possible alternatives.[112]

Another perceptual phenomenon related to Miller's law is the *Serial position effect*. It describes the tendency of recipients to recall the first and last items in a series best, and the middle items worst.[113]

In general, breaking up large entities into smaller chunks can improve user engagement, as will be discussed in the context of gamification on p. 25.

A comprehensible and consistent navigation is what connects the items and groups of items. Ensuring that users can intuitively navigate a web page is very important—if they cannot find their way around it, they will simply stop using it.[114]

2.6.6 Interface and Interaction Design

To ensure technical usability and reduce system-related frustrations, it is important for the interface to be easy to use and intuitive. An intuitive interface means shifting attention from manipulating said interface to actually accomplishing a task.[115]

By definition, interface design is strongly intertwined with usability, visual design and content structure. There are however several guidelines, best practices and resources often referred to, e.g. Nielsen and Molich's *User interface design guidelines* and Ben Shneiderman's *Eight golden rules of interface design*.[116] As there is considerable overlap between these two, only the latter will be elaborated.

Shneiderman proposes this collection of principles which are heuristically derived from experience and applicable in interactive systems:[117]

108 Morville 2006.
109 Refer to Morville and Rosenfeld 2006, 24.
110 Refer to Miller 1956; Davidson 2017.
111 Refer to Krug 2013, 66; Davidson 2017.
112 Refer to López 2016.
113 Refer to Davidson 2017.
114 Refer to Krug 2013, 51.
115 Refer to Mendoza-Gonzalez 2016, XVIII.
116 Refer to Wong 2018.
117 Refer to Shneiderman 1987.

- **Strive for consistency:** Similar situations and sequences of actions should be consistent and standardized. Familiar icons, colors, menu hierarchy etc. should be utilized.
- **Enable frequent users to use shortcuts:** Interfaces should allow for quick navigation and operation, so that frequent users can achieve their goals quickly.
- **Offer informative feedback:** Every action should yield appropriate, human-readable feedback within a reasonable amount of time. Users should always know where they are, e.g. in the process of a multi-page questionnaire.
- **Design dialog to yield closure:** Sequences of actions should be organized in groups and have a beginning, a middle and an end. Informing feedback after finishing provides the user with a feeling of alleviation, making room for the next action.
- **Offer simple error handling:** Systems should be designed as fool-proof as possible, but if errors do occur, the system should offer ways to solve the problem and recover gracefully.
- **Permit easy reversal of actions:** Allowing users to reverse their actions relieves anxiety, since errors can be undone, and encourages deeper exploration of unfamiliar options.
- **Make the user feel in control:** Users like to be the initiators of actions, active rather than passive, and in full control of events. Thus, the system should behave as they expect.
- **Reduce short-term memory load:** As human attention is limited, an interface should only show a limited number of items of any group. Displaying multiple pages or content frames at once should be avoided.

In his definitive work on the topic of usability, *Don't make me think*, Steve Krug also offers some advice that can be applied to interface design:

- **Make all elements self-evident:** Every question raised adds to cognitive workload and distracts attention from the task at hand.[118]
- **Create a clear visual hierarchy:** Apply prominence, grouping, and nesting elements.[119]
- **Stick to conventions:** Only break conventions if there are very good reasons for it.[120]
- **Define areas:** Break up pages into clearly defined areas, make it obvious which elements can be interacted with.[121]
- **Always allow for a new start:** The homepage is important as a fixed place, a chance for the user to start at zero.[122]
- **Persistent navigation:** On every page except the homepage and forms, persistent navigation should appear on every page.[123]
- **Identifiable pages:** Every site should offer a clearly identifiable ID, page name, sections, local navigation, "You are here" indicator, and a search box.[124]

118 Refer to Krug 2013, 11–16.
119 Refer to ibid., 31.
120 Refer to ibid., 36.
121 Refer to ibid.
122 Refer to ibid., 58.
123 Refer to ibid., 63.
124 Refer to ibid., 85.

2.7 Gamification

This section explores how the application of game mechanics is exploited to increase motivation and support users in achieving their learning objectives.

2.7.1 Concepts and Methodology

Generally, the term GAMIFICATION refers to the application of game elements (game theory and game mechanics) in non-gaming contexts to engage and motivate users.[125] This is closely related to human behavior. In particular, the relevant ideas derived from psychology are:[126]

- **Sequencing:** Users are more likely to take action when complex tasks are broken down into smaller entities. Learning goals should be defined, numerous, and most importantly, measured.
- **Appropriate challenges:** Users delight in challenges, especially ones that strike a balance between being overwhelming and being boring.
- **Status:** Users constantly assess how interactions enhance or diminish their standing relative to others and their own past successes. They strive for competition and recognition.
- **Achievements:** Users are more likely to engage in activities in which meaningful achievements are recognized.

The quasi counter-draft to gamified e-learning systems are so-called *serious games*. While e-learning provides an educational platform with some degree of embedded game elements, the latter are video games with embedded educational content.[127]

125 Refer to Bright 2015.

126 Refer to Anderson 2011, 3, 100–103; Iñigo 2015.

127 Refer to Greeff et al. 2017, 14.

2.7.2 Relevance in the Context of E-Learning

There is a significant amount of research supporting the proposition that gamification of e-learning systems can significantly improve student motivation, engagement, and persistence.[128] According to Tolle, the benefits of using gamification in the context of e-learning courses include: It provides instant feedback; it builds engagement; it generates more student loyalty; it boosts productivity; it gives more influence over students' actions; it increases learning retention; it obtains more time spent; it is fun and enjoyable in the students' eyes.[129] According to various studies, 80–89% of learners would be more productive if their learning process included game elements like a point-system.[130]

Shantanu Sinha is a resolute advocate of gamification. He suggests that a lot can be learned from the game industry, as it has "turned motivation and incentive systems into a science". The sense of immediate success and progress, he hopes, will provide "a personalized path full of individual triumphs" in the context of online education, so that students will be able to "reclaim their natural enthusiasm and passion for learning"[131].

Greeff et al. point out that the educational context brings a unique benefit as both types of motivations can be addressed—game elements target intrinsic motivation, but there is also the extrinsic need to succeed and improve in a course.[132] Moreover, Penfold remarks that gaming elements provide learning experiences that are not even possible with traditional face-to-face education.[133]

128 Refer to Iglesias Cancio 2014, 3; Tolle 2014; Lopes and Mesquita 2015, 10; Iñigo 2015; Bernik et al. 2017, 715–716; Katambur 2017.

129 Refer to Tolle 2014.

130 Refer to Bright 2015; Iñigo 2015.

131 Sinha 2012.

132 Refer to Greeff et al. 2017, 15.

133 Refer to Penfold 2016.

Bernik, Radoševic and Bubaš have performed experiments with a gamified e-learning course of programming lectures. An experimental group took the gamified course, while a control group was presented with the regular course. Both courses had the same educational content, the difference was merely in the game elements. Remarkably, the respondents in the experimental group accessed the materials 3.69 times more often than the control group (Figure 15).[134]

While the result described above clearly is a substantial indicator of the effect of gamification on student motivation, it is also noteworthy that the activity in *both* groups decreases a fair amount with increasing course topics. Apparently, the positive effect of gamification is not unbounded. Also, there are some pitfalls to avoid—Staubitz et al. point out that employing gamification on an e-learning platform must be handled very carefully, as "a thoughtless implementation will let the platform fall back to the simple positive reinforcement mechanisms of behaviorism and programmed learning"[135]. Sinha agrees that there are many motivation systems with imperfect incentives that can amplify the wrong behavior.[136]

2.7.3 Tools and Principles in the Context of E-Learning

In this section, an overview of tools and methods for the gamification of e-learning is given, as proposed by Emilia Iñigo:[137]

- **Create clear learning pathways:** Give students the sense that they are progressing towards something; arrange content in a level structure; make learners complete content in order to "level up" and unlock the next learning unit; provide series of small, achievable challenges to reach an overall goal; Gather the user's tasks and advances on a dashboard.
- **Motivate students to complete content:** Use UI features like progress bars to subconsciously urge learners towards completion; together with achieved goals, show which objectives are yet to achieve.
- **Create a sense of accomplishment:** Include a point system; offer some kind of reward after each step; issue points, coins, badges, medals, trophies, etc.
- **Keep students returning:** Reward users for interacting with the system, e.g. for logging in for a number of consecutive days.
- **Exploit competitive urges:** Juxtapose users in a leaderboard.
- **Keep to deadlines:** Add a time limit to a given task.

134 Refer to Bernik et al. 2017.

135 Staubitz et al. 2017.

136 Refer to Sinha 2012.

137 Refer to Iñigo 2015; Growth Engineering 2016; Penfold 2016.

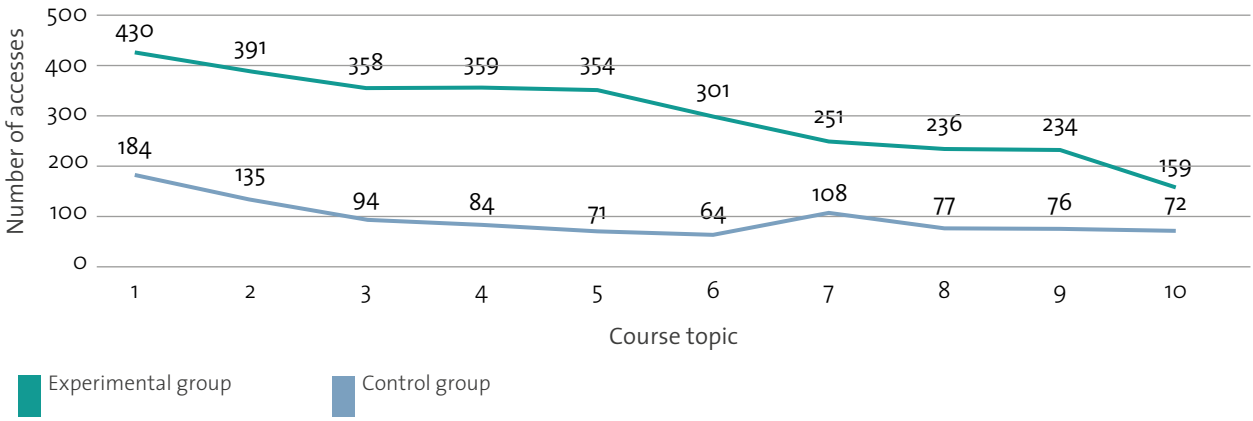


Figure 15 Frequency of accessing gamified vs. regular educational materials in ten subsequent course topics.

3 Analysis

To determine where energy is best spent and how the project can benefit the most, a thorough inspection and valuation of the pre-existing code base, as well as the running application, has been undertaken.

3.1 Codetask – The Good Parts

Codetask was mostly built by Daniel Jehle as his final project and bachelor thesis.[138] Subsequently, other students and professors have worked with the application and added content.

The application has been designed to help students learn the programming language *Scala*. More specifically, students of the *Department of Computer Science* at *Hochschule Konstanz University of Applied Sciences* in their first semesters. Students may register accounts and log into the system, where they may enroll in multiple courses. Courses represent a collection of one or more chapters, which in turn comprise one or more tasks. A task can be one of three types:

- **Koan Tasks:** Similar in spirit to the aforementioned *programming koans*, these tasks consist of code with gaps that need to get filled to make the code executable.
- **Code Tasks:** These tasks are also made up of a code editor with some unfinished code. The student must implement the missing parts so that a defined condition is met, and an underlying *Scala* test case succeeds.
- **Video Tasks:** Here, the student has to watch an educational YouTube video from start to finish.

Other than students, there are roles for admins and teachers, which are authorized to create, edit, or delete courses and users. New content can be created by utilizing a custom-made domain-specific language (DSL) and either using the admin upload form on the website or a script that comes with the back-end code.

From a technical point of view, the platform is based on the typical *three-tier-architecture* of web-based applications, comprising *presentation tier*, *logic tier*, and *data tier*. The application is built with the *Play Framework* and the *Scala* programming language. This is a sound choice—in fact, it is the same stack *Coursera* uses[139].

An adequate API provides data access to clients. The general data model is well-crafted. Other than serving the front-end, the back-end implements a parser that transforms *Scala* source code (resp. the aforementioned DSL) into the internal JSON format as well as an interpreter that is supposed to process the user's code submissions at runtime.

On top of that, a user interface has been built that utilizes *Polymer* in conjunction with the *Bootstrap* library. To actually have a code editor for the user to interact with, the *Ace* editor has been integrated. The interface offers visual cues on which tasks a user has already finished, which can be credited as a first step towards gamification.

So, in short, *Codetask* is a web application that solves a real-world problem and offers a broad spectrum of functionality—in theory. A closer look will reveal several serious issues.

138 Refer to Jehle 2016.

139 Chen 2016.

3.2 Technical Shortcomings

3.2.1 General Availability

The first obstacle was actually accessing the application. Even though a subdomain had been registered under the department's domain, a call to said URL returned an HTTP 404 error, leaving the server itself in an undefined state. What's more, after locally checking out the source code, it was not in a runnable state. A lot of reverse engineering was necessary to run the program, and the documentation did not help much. Only after starting a local *MySQL* server; adjusting the `persistence.xml` file so that non-existent databases would automatically get created; and hacking the very same database to manually create a user with admin rights, the application would finally run.

3.2.2 System Architecture and File Structure

There are few packages in the code base and the way the system is designed and code is distributed among these packages reveal some deep misconceptions. E.g., the package `models` does contain some data objects. However, it also incorrectly contains a configuration file and a lot of business logic, such as the `CourseParser`, `CourseService` and `Interpreter` classes. Important principles like separation of concerns are broken and the scopes of responsibility are unclear.

The front-end part is not much better structurally: It is no separate repository, nor a distinct folder. The *Polymer* components are all thrown in a `node_modules` subdirectory of the `public` folder that *Play* serves at runtime. Hence, there is no separation between original code and third-party dependencies, which also live in this folder.

3.2.3 Code Quality

The code base is pervaded by various *code smells*, i.e. problematic code patterns: Large classes, long methods, commented-out code, non-meaningful naming, a low level of abstraction, lots of duplicate code etc.[140]

At some points, especially in the front-end code, central language features seem to be unknown to the developer, as they get cumbrously reprogrammed. Web frameworks are usually data-centered, and *Polymer* is no exception. It offers several ways to adjust a component's contents to its state, e.g. two-way data binding, computed binding, attribute binding or the `toggleClass` method. In the code base however, such adjustments are all made manually, and state gets untangled in plain if-else statements.

3.2.4 Authentication and Data Exchange Between Server and Client

For authentication, a copy-pasted code fragment from a public *GitHub* page implements cookie-based *Basic authentication*. As this is an important and complicated matter, one would be better advised to rely on a well-tested and field-proven library like e.g. *Silhouette*. What's more, the API is secured inconsistently. While some routes are only accessible with the appropriate authentication header, others can be freely accessed. Strictly speaking, cookie-based authentication is also not a good bet when served over an insecure connection (see 2.4.1), and the application did not have an SSL certificate at that point.

The flow of data is not ideal, e.g. when a user opens her course overview in the browser, a massive JSON gets transmitted, which contains deeply nested exhaustive data of all courses the user is enrolled in. The only data really needed at this point would be the names and progress of the relevant courses. Not to mention that after the user clicks on a specific course, another request is made to retrieve a JSON containing this course's

140 Refer to Fowler et al. 1999, 76, 78, 84, 87.

data. The data from before, despite containing the selected course, gets discarded.

3.2.5 Semantically Correct HTML

On a semantical level, the rendered HTML does not describe the contents ideally. E.g. in the course view, each single task does not consist of one root element with multiple children (as would have been the default behavior when implementing web components), but two elements: a `div` containing the description, followed by a custom element hosting the actual task. Of course, both are part of a logic entity and should be implemented like that. Whilst invisible to most users, such inaccuracies have implications on the page's accessibility and make it harder for people with impairments to navigate it.

3.2.6 Operability

All of the above are relatively small flaws that one might be able to overlook and are most likely owed to time constraints. What is much worse: Some parts are not in working order and some features that appear to operate in fact do not.

- The user's progress for a given course that can be seen on the admin pages is simply wrong.
- While the admin pages allow creation of new users, trying to log in as a newly created user produces an error.
- When the user dispatches her solution to a *code task*, the *Scala* interpreter that processes the code on the server throws a cryptic exception and silently fails.

Even for a prototype, such unannounced surprises are unfortunate; for a live application deployed to production, they would be inexcusable. In the context of e-learning, error-prone systems quickly drive users away: "It is extremely frustrating to attend an online class and to face technical challenges"[141].

141 Banna et al. 2015, 259.

3.3 Usability-Related Shortcomings

3.3.1 Layout, Style and Aesthetics

The first thing to notice when opening the web application is that it is not a SPA, but a traditional multi-page website. Each navigation causes a complete page reload. There is nothing inherently wrong with that approach, but it does feel slightly outdated and certainly less smooth and snappy than users have come to expect from a modern web application.

The login screen fails to attract users and create a positive first impression. It is a bleak white page without a logo, or name, or any other indicator of what content the application offers. The few elements that can be found seem a little lost amid all the white space. What follows is more of the same: Overused *Bootstrap* default styles on mostly empty pages. Additionally, the separate views visually diverge considerably. Font size, spacing, placement of elements etc. oftentimes have no evident underlying system.

Whilst most views radiate a sense of emptiness, the course view goes to the other extreme. It is overloaded and throws all the tasks of the current chapter at the user. It is not really clear where to begin—how to solve the task at hand.

Moreover, usability is impaired because some elements deviate from common design conventions. Aesthetics and visual comprehension are a means of sensemaking, as discussed in 2.6.3, hence the application increases cognitive load by breaking said rules. E.g., there are drop-down menus that look like normal text, so it is not evident that they can be interacted with.

In fact, quite a few elements do not communicate clearly what their function is or how to use them. E.g., the *koan tasks* encourage the user to fill out the gaps. But then what? In a small usability test, users have instinctively searched for a button to run the code. Instead, the input gets validated after each keystroke, and only after the correct answer is put down, the highlighting gives the user a cue. Another disorientating example are buttons

merely labeled with an icon, where the choice of icons does not comply with common semantic understanding. The *writing pen* icon e.g. usually connotes editing content, whereas here it means "go to course learning view".

In the course view, the *save* button is positioned at the top left. Apart from being suboptimal in the first place—it would be better to simply store the user's progress instead of interrupting her with an alert pop-up and make her click on the *save* button—the button should be on the bottom right instead. The corresponding action chronologically happens at the end of the interaction with the page. Due to the western habit of reading left-to-right, top-to-bottom, placing something at the top left means placing it at the start. Also, the TERMINAL AREA, the bottom right quadrant, is where the user's viewing pattern typically ends, making it an ideal place for call-to-action such as buttons or links (Figure 16).[142] In addition, the position and general appearance of the button make it align so strongly with the ensuing menu that it gets assimilated visually, hence no longer perceived as a button.

The application makes use of window.alert and window.onbeforeunload to trigger a browser built-in alert dialog in several situations. This should be avoided, as it is indisputably an obsolete technique and interrupts the user unexpectedly, e.g. when she wants to leave the course view with unsaved state. Furthermore, there is no consistency, as some of the alerts actually do come as in-page *Bootstrap* modal windows, and some messages are in English, while others are in German (resp. the browser's preferred language).

3.3.2 Navigation

If the user navigates from the initial login view to the signup view, there is no way to get back. It gets better in the actual application, as the breadcrumbs on the top of the page allow navigation to some extent. Still, the first screens a user interacts with are important as they strongly inform her

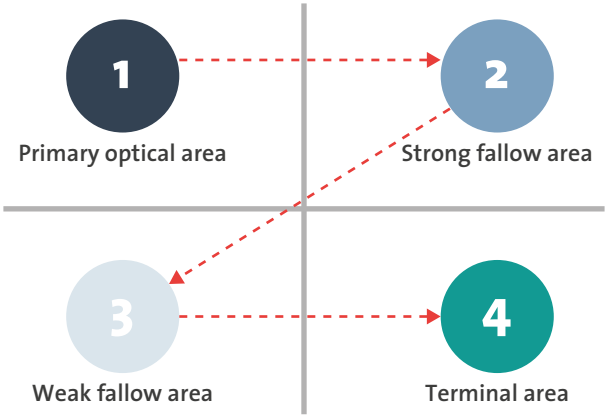


Figure 16 The so-called Gutenberg diagram, illustrating the user's eye movements on a display.

impression of the page. Also, breadcrumbs instead of an actual header with a title and menu are far from being a generally accepted design approach.

The paths users have to take to achieve their objectives are sometimes confusing, e.g. when an admin wants to open courses, she cannot access them the same way regular students do but has to go to the course management page and then click on "Try courses"—only to end up on the very same resource other users access on a different path.

When an error occurs (e.g. a server error or the user enters a wrong URL and receives a 404 response), the error message string is the sole content of the returned HTML. There is no link to bring the user back into the application, let alone a solution to recover from the error gracefully.

3.3.3 Supporting the Users to Achieve Their Objectives

Some system-related frustrations stem from the lack of assistance and guidance. A good example can be found on the admin's course upload page:

- The file dialog accepts and show all file types, even though all files except .scala files will cause an error.

- The *course title* input field does not get populated with available data, e.g. the file name as default value.

142 Refer to Andrade 2013.

- The *course title* input field is barely visible, and if the user chooses a file and tries to submit without entering a course title, an error will be produced—the whole upload process has to be performed de novo. This could have been prevented with simple form validation to highlight missing fields and prevent premature submission.

- There is a functionality to add chapters to a course subsequently, which actually overwrites existing chapters with the new ones without a warning.

3.3.4 Content-Related Frustrations

The courses that are currently available derive from different sources and authors, consequently, they differ in quality, language use, style and structure. The length of chapters varies extremely—some courses have up to 43 tasks while others merely consist of one. The heterogeneity they show in regards to length is lacking when it comes to conceptual variety however. The tasks soon become repetitive, and only three types of tasks (one of which is broken, as discovered in 3.2.6) may be too few for an engaging learning experience.

Some tasks have clearly been designed without the user in mind. E.g., a line in a *koan task* says `scala.math.Pi should be _____`, with the expected answer being "3.141592653589793". Even in the unlikely event of a user actually knowing the first 15 decimal places of pi by heart, she would not know how many characters exactly to input. Besides, the pedagogical intention of the author presumably was to teach *Scala*'s math library rather than the actual value of the mathematical constant, so it should probably be just the other part of the line that the user has to complete (even if that would be harder to map to an actual *Scala* unit test).

Assuming that the *code tasks* would be operable, the instructions would oftentimes be scarce and confusing. The last lines of code, that actually test a given condition and determine if a task is solved, laboriously get extracted and removed in the back-end, only to later get sewed onto the user's submission again. The user never sees them. Unfortunately, these lines of code would actually improve the understanding of the task at hand, hence it would be preferable to show them in the front-end.

There are some *koan tasks* with zero gaps. They are meant to be plain informational material. However, the data model does not support this edge case, so these tasks count towards the unsolved tasks and it is impossible to complete the affected chapter.

3.4 Studies on Accomplished E-Learning Platforms

As Davis et al. point out, there are many successful examples of online learning systems available to learn from, thereby gaining knowledge to improve own e-learning implementations.[143] In a small comparative study, some of the most important e-learning platforms have been analyzed in terms of interface design and UX. Unfortunately, they are generally not open source, so analysis is restricted to interacting with their applications.

Comparing *Khan Academy, Coursera, edX* and *Udacity*, the following similarities stand out:

- The user interfaces are very similar in that they are clean and simple (Figure 18).
- Use of layout and colors is unobtrusive, as not to take attention from the content.
- Content is hierarchically structured, usually in three layers (e.g. *course* > *week* > *section*).
- The content structure gets visualized and users' expectations get managed very well (e.g. for upcoming lessons there is an estimated duration and indicators of the types of tasks).
- The visualization of the structure is combined with progress bars of some kind to reflect progress and include basic gamification.
- The subjects offer user-to-user or student-teacher interaction without exception.
- Common conventions were followed, e.g. a persistent navigation bar always allows access to the most frequently used actions.
- Users work through various tasks, incl. watching short video lectures, interactively solving problems and quizzes, and doing readings.
- All websites implement responsive web design.
- Text is used in conjunction with icons to convey meaning.
- All platforms have some form of reward system, such as points, achievements, badges, etc.

The language training app *Duolingo* is a particularly strong advocate of gamification. It presents daily challenges and notifies the user when she does not achieve her goals or gets overtaken by a friend. Even in the middle of a lesson, it will take the time to surprise users with encouraging messages from the app's mascot (Figure 17).

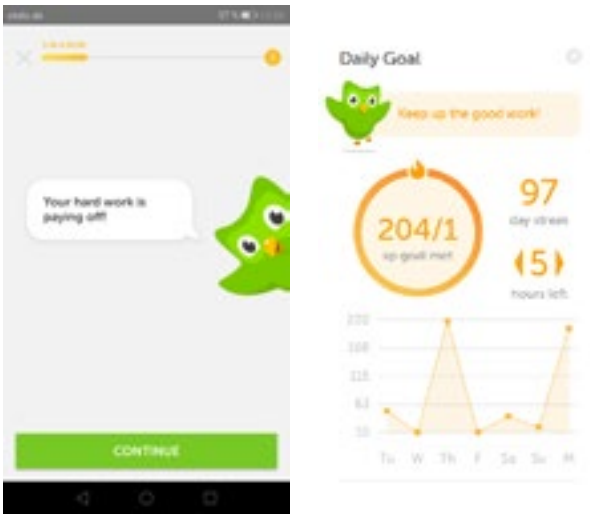
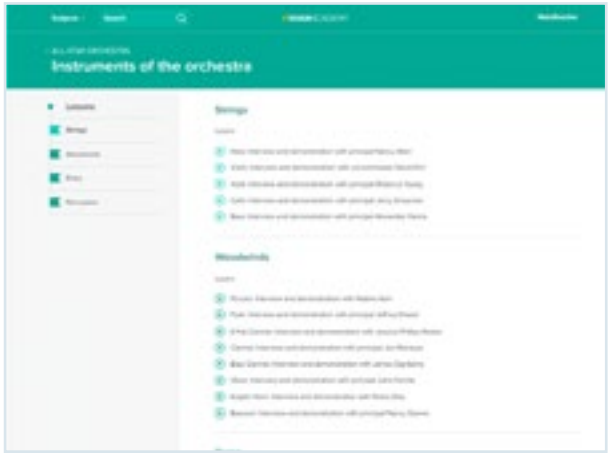


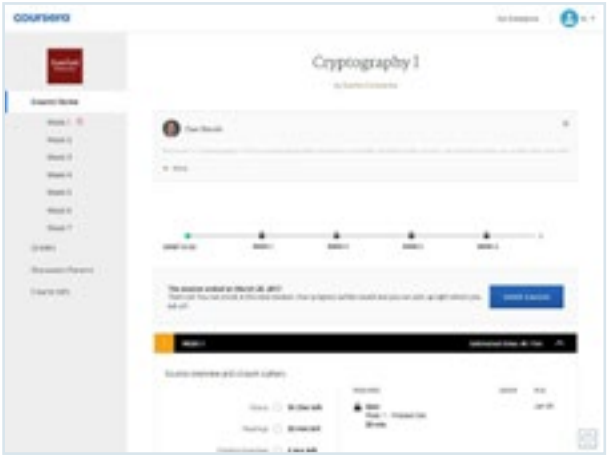
Figure 17 Duolingo motivates the user with daily challenges and encouraging messages (screenshots).

143 Refer to Davis et al. 2008, 140.

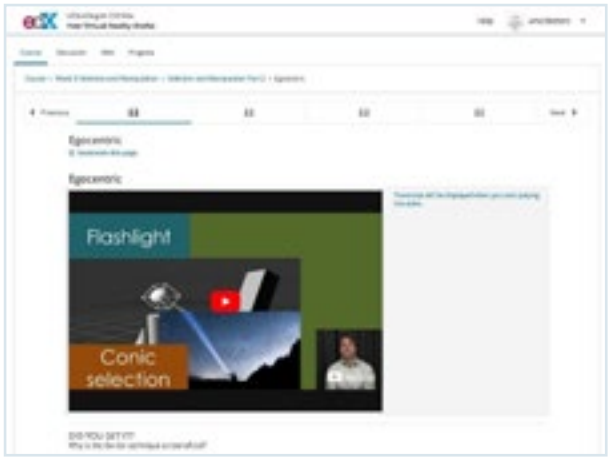
Khan Academy



Coursera



edX



Udacity

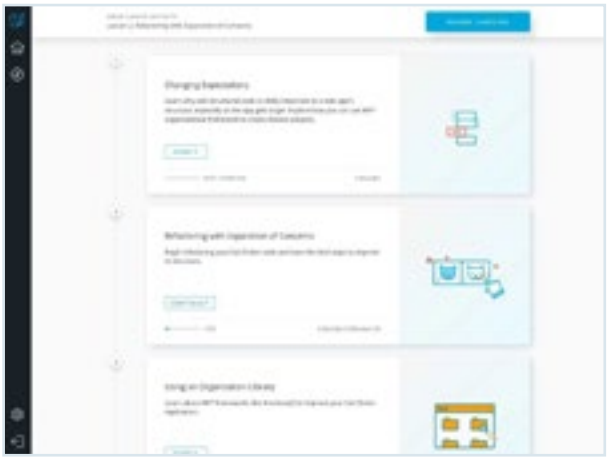


Figure 18 Comparison of dashboards and courses UI of accomplished e-learning platforms.

4 Implementation

Based on the theoretical groundwork and insights gained in the analysis, a new interface for *Codetask* and alternative back-end solutions have been implemented. This chapter will discuss various aspects of the implementation process.

4.1 Scope and Limitations

The preceding analysis has proven the existing application to be a double-edged sword: On the one hand, it offers a remarkable set of features and tries to solve a real-world problem. The general data model is good, and new courses can be created with a custom DSL. Also, the standards of the analysis may have been too harsh considering that the research object is a student project, and expertise in different fields as well as countless working hours would have been necessary to meet the given expectations.

On the other hand, several more or less severe flaws have been found, fixing which would go well beyond the scope of this study. Taken together, these flaws add up to an application that virtually could not be used. Barring the real operability issues discussed in 3.2.6, it is mostly usability barriers that take their toll. In the *Hierarchy of needs* model discussed in 2.7.1, the application—again, ignoring the partially broken functionality—would inhabit the lowest level: It works as programmed. It is therefore fair to say that *Codetask* is a typical example of the damage done by excluding usability considerations from the development process.

So, the question is: Where could adjustments be made to take *Codetask* from where it is now to an experience users happily engage in, one that provides them with real value and permanent learning progress?

As the department explicitly expressed the need for the application to be in working order by the start of the following semester, this self-evidently is the main goal of the practical project. The secondary objective is making the application more engaging and easy to use, improving its standing in the *User Expercience Hierarchy of needs* pyramid, so to speak.

With these goals in mind, the decision has been made to take the following measures:

- Build a new user interface to completely replace the old one.
- Improve user experience to increase engagement.
- Use gamification techniques to increase engagement.
- Filter *code tasks* to circumvent the most critical back-end bug.

After starting development with the conception of creating a new client for the existing server, it soon became evident that numerous server-side problems delayed the process and the authentication concept would not fit a SPA. In fact, as the new front-end gets served independently and the interpreter functionality is broken anyhow, parsing new courses is the single exclusive feature still required from the back-end.

Consequently, the new front-end has been redesigned to be decoupled from the old server and is able to run independently. Authentication, user management and data persistence functionalities have been moved to *Firebase*, as will be described in further depth in the following sections.

Note that new data may still be parsed using the old client or the script, and the resulting JSON files can get imported into the new application. It is completely backward-compatible, enhances data progressively if at all, and leaves the door open to reunite front-end and back-end, should there be any need for that in the future.

4.2 Discussion and Selection of Core Technologies

To expand on 2.3.1: Lewis' critique that frameworks are expensive is certainly legitimate. However, as *Codetask* is primarily targeted at students of the local university, it can be expected that most users will browse the page connected to the campus WiFi, so the extra bandwidth is bearable. On the whole, it is clearly infeasible to write the application in plain JavaScript, given the restrictions and goals of this project.

Many have tasted and contrasted the important frameworks. All kinds of metrics were found to compare them: Performance; bundle size; lines of code; *NPM* package downloads; *Google* trends; *GitHub* stars; degree of activity in the community; ease-of-use; learning curve; ecosystem; sustainability; support from a large company; historical misdeeds; etc. It is mostly *Angular*, *React* and *Vue* that are competing for the pole position. Some insights were gained in the comparison in 2.2.2.

Remarkably, the three major frameworks seem to be almost on a par. Most independent, unbiased reviewers are of the opinion that it comes down to personal preferences and favored programming paradigm—virtually no one offers a clear-cut recommendation.[144]

The author of this study was therefore "spoilt for choice". Eventually, *Vue* was chosen, mostly because it seemed slightly more interesting from an academic point of view. It is the youngest one, less has been written about it so far. In addition, the unparalleled sympathy the framework is met with on *GitHub* and in the *State of JavaScript* survey excites a closer look.

The other technologies applied in the course of this project have mostly arisen out of this choice. In essence, they comprise:

- *Vue CLI*
- *Vue-router*
- *Vuelidate*
- *Vuex*
- *Webpack*
- *NPM*
- *TypeScript*
- *Element UI*
- *Axios*
- *Firebase Realtime Database and Authentication*
- *Vuexfire*
- *Docker*
- *Dokku*

4.3 Project Setup and Workflow

4.3.1 Project Creation with the Vue CLI

The *Vue* team is currently implementing a command-line interface (CLI) that helps to create new projects from templates, called vue-cli, resp. Vue CLI[145]. Even though the project is still in alpha phase, it is fully functional and could successfully be employed for setting up this project.

The CLI allows developers to start from different templates, ranging from very simple to rather complex project setups. For the task at hand, the main webpack template has been used. It is meant to be the starting point for large, serious projects and sets up a sophisticated workflow that will be discussed in the following section.

144 Refer to Morelli 2017; Schae 2017; Sidorenko 2017; Korotya 2017; Neuhaus 2017; techsith 2017.

145 There is no proper, canonical spelling for many tools and packages, as the lowercase hyphenated package names compete with official brand names.

Package management and build workflow are powered by *NPM*, development and dependencies get stored in the package.json file.

4.3.2 Module Bundling and General Workflow with Webpack

Webpack at its core is a module bundler, and as 2.2.7 has shown, is the leading technology in its domain. It combines many modules and files into few bundled assets, while figuring out and resolving the dependencies between them. Various so-called *loaders* allow for their corresponding module types; e.g. the vue-loader transforms *Vue* components to plain JavaScript. Webpack is very flexible, custom loaders can implement all kinds of functionalities.

Some of Webpack's features used for this study are:

- Linting the source code, i.e. providing static code analsis to make sure code style is consistent.
- Running a local development server with state preserving hot-reload, compilation error overlay, source maps and more.
- Running unit tests.
- Transpiling *ES6* and *TypeScript* to plain JavaScript, i.e. transforming a language into another, to make it compatible with older browsers.
- Transpiling *Sass* into plain CSS.
- Securely accessing secret environment variables.
- Building assets for production, incl. "uglifying" and "minifying" the code, i.e. removing comments, renaming symbols and generally compressing the code as much as applicable.

4.4 JavaScript Standard and CSS Preprocessor

By and large, the project was implemented in *ES6*. Some files were later upgraded to *TypeScript*, which can coexist with regular *ES* in a *Vue* application (see 4.6.2). In order to promote best practices and ensure a consistent code base, the *ESlint* static code analysis was activated and configured to prevent compilation while the code was non-compliant. For this project, the *JavaScript Standard Style* has been chosen, a set of rules that currently enjoys great popularity in the developer community. Most notably, it represents a small paradigm shift in that it abandons the end-of-line semicolons that have been considered good practice for the longest time. Regarding *Vue*-specific code, the official *Vue Style Guide* was obeyed in most cases.

Most components make use of the *Sass* preprocessor for styling, as it improves readability and reusability.

4.5 Application Architecture and Structural Design

In order to define the components—or "information atoms"—and their layout, dependencies, ordering and nesting, sketches and wireframes were created (Figure 19). The basic structure is pretty straightforward and defines how the components have been drafted and implemented: There is a landing page with login and registration forms and there is the actual application. It has a dashboard, some subpages, e.g. for administration or user settings, but most importantly, there is the learning view. This is where students will spend most of the time, where most interaction will happen.

The analysis has clearly shown that putting all tasks in one view is unflattering. If anything, both UX theory and gamification aspects suggest breaking large blocks of information down to smaller units. Consequently, after the redesign, each view shows one task at most. It should be noted that this change does *not* increase cognitive

Figure 19 Wireframes and sketches from the design phase.

load, as the vast majority of tasks do not refer directly to their predecessors, so short term memory presumably will not be strained.

A navigation bar allows moving back and forth between tasks in the learning view. All courses, chapters, and tasks of the same type are structured equally. The dashboard shows all enrolled courses, as well as the chapters of the selected course.

A clear component structure begins at the markup level. In fact, the component-based architecture of JavaScript frameworks forces developers to think about structure either way. With a clear application architecture, clean markup and semantically structured content are a by-product, as it were. In order to improve accessibility and satisfy modern standards, HTML5 semantic elements were used to describe the document outline (Figure 20). E.g., the application scaffolding is made up of `header`, `nav`, and `main`. Each task maps to a `section`, while the indicator of chapter progress is an `aside`.

Other than markup, a component typically comprises CSS and JavaScript. Technically, each component occupies its own file, a so-called *single-file component* which comprised of a `template`, `script` and `style` block.

From the authorization point of view, the application is divided into two areas: The *outer application*, i.e. the landing page, and the *inner application*, meaning all content that can only be accessed after users are logged in (Figure 21).

4.6 State Management and Data Flow

This section will illuminate how data is structured, how it is bound and propagated within the application, and how it synchronizes with the remote database.

4.6.1 Adapting and Enhancing the Existing Data Model

By and large, the data model of the original implementation has been adopted.[146]

Some adjustments have been made in how objects get identified for the sake of improving clarity and resolving dissonance with the routes, which were written with readability in mind (e.g. avoiding zero-based unit numbering). Also, additional fields have been added in the attempt to improve UX and apply gamification (e.g. course description and rating). User-related model changes could be implemented independently, as user management has completely moved to the realms of the new implementation. Course-related changes are performed at the admin course upload site, as it is the single entry point in this regard.

4.6.2 Gradual Integration of TypeScript

Removing the back-end server from the project added to the need for type safety and clear definitions on the client. Hence, the *TypeScript* programming language was leveraged. Via the optional static type-checking, *TypeScript* facilitates a clear data model and helps to identify and rectify problems much earlier in the process. Also, it offers better auto-completion, better error handling and better error messages, thereby accelerating development.

146 Refer to Jehle 2016, 27.

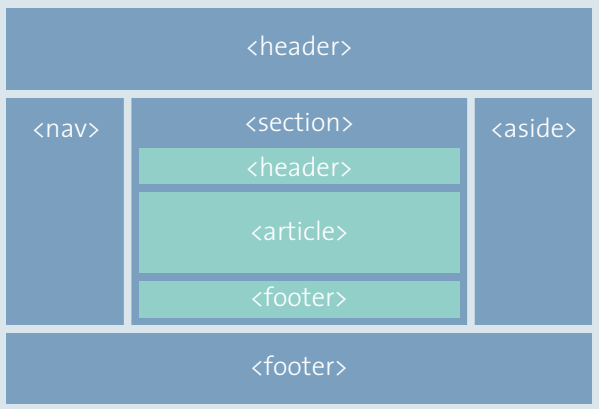


Figure 20 Typical document outline using HTML5 semantic elements.

Luckily, *Vue* has greatly improved its *TypeScript* support in version 2.5 and the framework's core team collaborates with the *TypeScript* team. For all core libraries (*Vue*, *Vue-Router*, *Vuex*), type definitions are available. It still is not part of the framework by default, but it can be included—albeit the process is quite cumbersome and requires several small changes scattered all over the application.[147]

147 Refer to Papa 2017.

Because *TypeScript* is a superset of JavaScript (resp. *ES6*), it was integrated incrementally. *Vue* allows for the `script` block in each component to specify its programming language, so *ES6* and *TypeScript* can exist side-by-side. To represent the application's business logic, classes and interfaces have been defined.

4.6.3 Reactive Data Propagation

Vue comes with a powerful yet lightweight and unobtrusive reactivity system. Data models are just plain JavaScript objects. When they change, all correspondent views change automatically.

Under the hood, *Vue* converts all defined properties to getter-setter pairs using `Object.defineProperty`. This way of adding properties to objects allows to specify accessor methods, which will internally run when the property is accessed, thereby offering an entry point for the framework to when values get written or read. Thus, *Vue* can track dependencies and implement a classic observer pattern.

Every component has a corresponding watcher instance. When the component first renders, the watcher records all properties that were read as

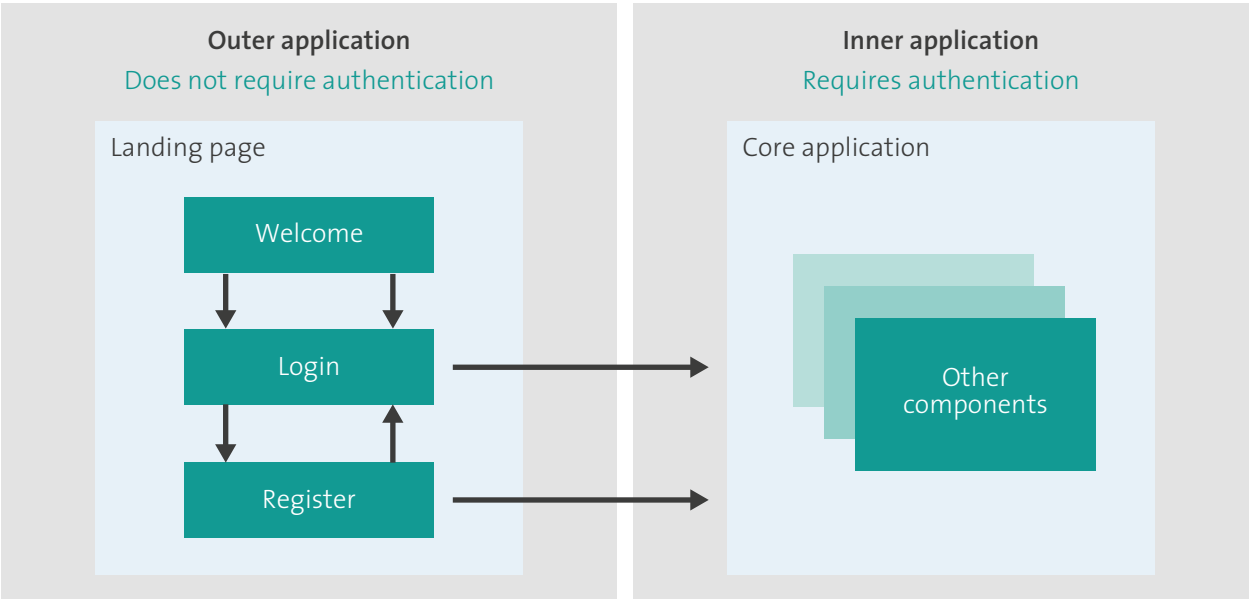


Figure 21 Application architecture in terms of access control.

to understand the dependencies, and registers as an observer. Later, when one of these properties changes (i.e. the setter code gets executed), it will notify the watcher, which in turn causes the component to re-render (Figure 22).[148]

4.6.4 Traditional HTTP Requests and AJAX with Axios

There are several ways to access remote data in *Vue*. At first, vue-resource was the default solution for HTTP, resp. AJAX requests. It is still a valid choice, but in 2016, it has been retired from official recommendation status. Instead, developers are encouraged to use the *Axios* library. It offers all the same functionality with a similar interface. Plus, it is universal, supports canceling requests, and ships with *TypeScript* definitions. Also, it is Promise-based, thus allowing for readable code and elegant handling of asynchronous responses.[149]

Considering component architecture—meaning how and where to implement AJAX—several patterns have emerged as *Vue* does not provide or enforce an official way. Anthony Gore has identified four basic patterns, each with its own advantages and disadvantages[150]:

- All AJAX requests may be sent from the root instance, where all state is stored as well. Data gets passed down to child components as properties (or *props* in *Vue* terminology), which in turn fire a custom event to trigger the root instance's functions.
- Components can also be responsible for managing their own network requests and state independently. With this approach, it is encouraged to create "container" components to aggregate shared functionality.
- Route navigation guards can trigger the required data to get fetched dynamically.
- Lastly, both state and AJAX logic can be managed in a *Vuex* store, and network requests can be wrapped in *Vuex* actions. This approach will be discussed in the next section.

148 Refer to Vue.js Guide 2018b.

149 Refer to You 2016.

150 Refer to Gore 2017a.

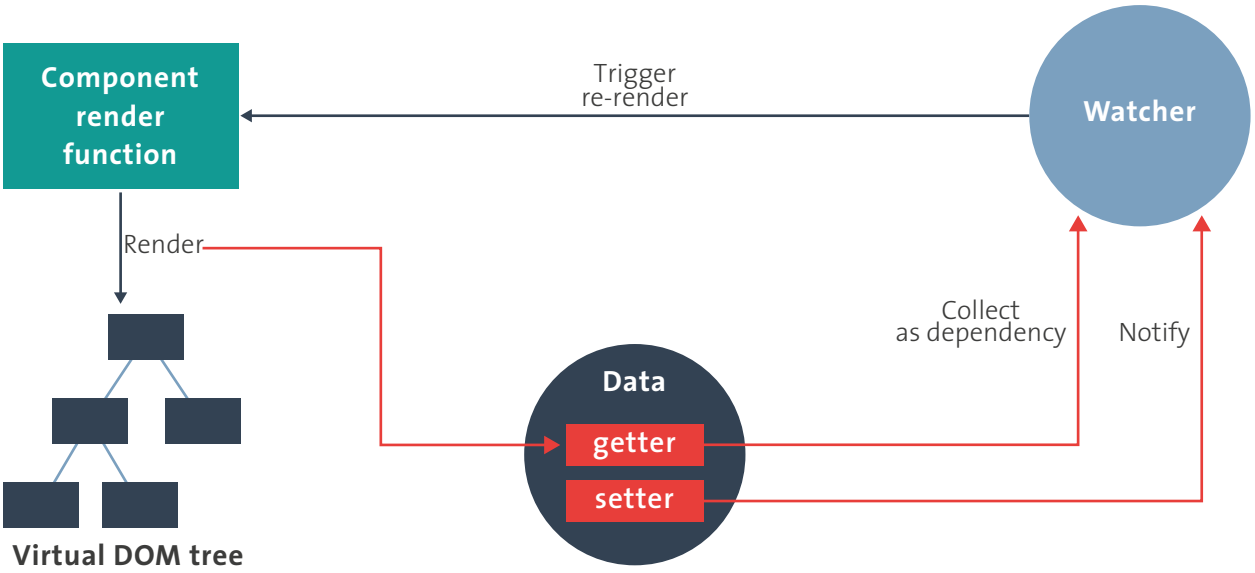


Figure 22 Change detection in the reactive cycle of Vue.

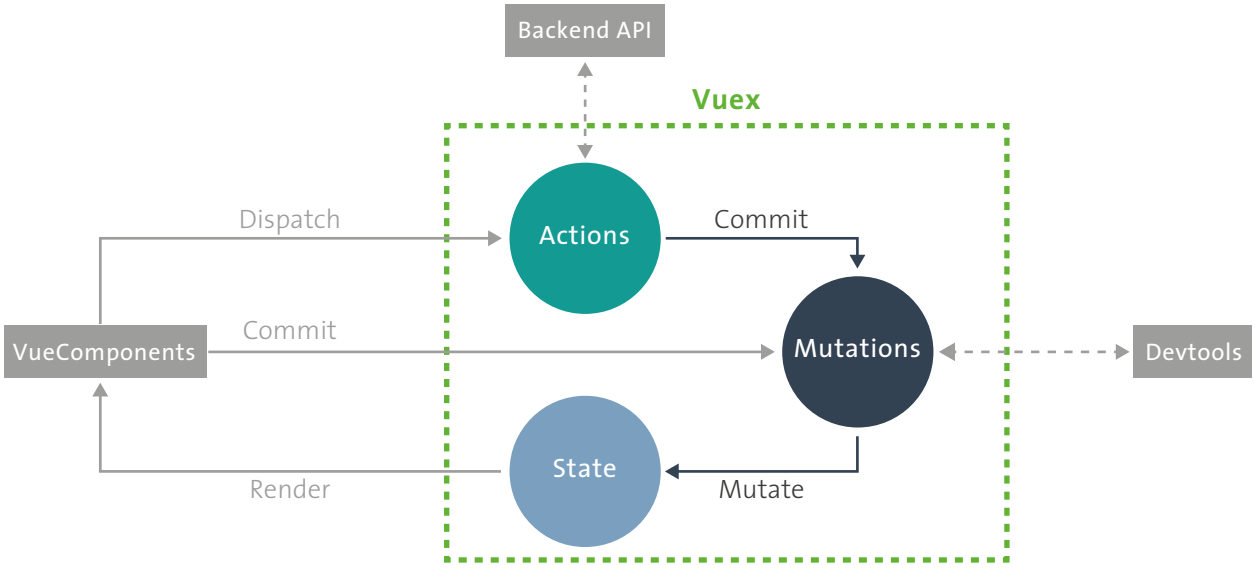


Figure 23 State management concepts of Vuex.

4.6.5 Centralized State Management with Vuex

As has been addressed in 2.5, a centralized data store is a worthwhile pattern found in many modern applications and *Vuex* is the default library to implement this architecture in *Vue*. Rather than exactly implementing *Facebook*'s *Flux* pattern (which is somewhat tailored to *React*), *Vuex* provides a subset of it, while also taking advantage of *Vue*'s granular reactivity system for efficient updates.[151]

With *Vuex*, there is an application-wide single source of truth: the so-called *store*. Any data that needs to be shared between components has to be kept there. Also, the state itself is read-only to the outside world, so components have to inform the store of their intent to change data by committing *mutations*. These commits can be meticulously retraced, thus it is easy to track down inconsistencies. In fact, the *Vue.js Devtools* browser extension even provides *time-travel debugging*, meaning that any commit can be rolled back and reapplied.[152]

Mutations have to be synchronous transactions—if they were not, one would only know the order of execution, but had no way of reconstructing the order in which they were committed. Consequently, to perform asynchronous actions like requesting data from a remote server, components have to dispatch *actions* instead. These can in turn commit mutations once they are ready (Figure 23).

Thanks to *Vue*'s reactivity system, components will automatically change their views according to state changes in the store. If the data presentation should be modified in some way, e.g. filtering a list, *getters* can be defined as to avoid duplicate code in dependent components.

Vuex additionally provides the concept of *modules*. Each module is its own set of state, mutations, actions and getters, with the store being the union of all modules. Typically, separate areas of business logic get mapped to separate modules.

While reimplementing *Codetask*, modules were written for user-related and for course-related state. All components only maintain local data which does not affect any other parts of the application—otherwise, they use the *Vuex* store and all of the techniques described above.

151 Refer to Vuex Documentation n.d.

152 Refer to Gore 2017b.

4.6.6 Near Real-Time Data Synchronization with Firebase

4.6.6.1 Notes on the "Firebase Realtime Database"

As the old server has been deprecated, the need for a new solution was apparent. The choice fell on the *Firebase Realtime Database* service, a cloud-based *NoSQL* database that seems to be a very good fit for serverless apps.

The service is part of the *Firebase* suite *Google* offers for mobile and web application development. It is a set of tools that allows developing client-side applications without any server-side programming, meeting basic requirements like authentication or database storage, while automatically taking care of scaling.[153]

The marketing-coined term *real-time* may well be called into question though, as it is slightly vague in its definition: "Real time is a level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process [...] Real time describes a human rather than a machine sense of time."[154] In the context of this study, the more neutral term *near real-time* will be used instead.

Firebase's key capabilities include:[155]

- Storing JSON data; synchronization with any number of connected clients.
- It defaults to web socket connections instead of traditional HTTP requests, providing a responsive user experience.
- Software development kits (SDK) for multiple platforms and programming languages are available.
- Some offline capabilities are included; the SDKs persist data locally, so an application remains responsive even when network connection is lost. What's more, *Firebase* implements background sync, meaning that when connection is re-established, it synchronizes the recorded local changes and merges any conflicts with remote updates automatically.
- It comes with a simple but powerful expression-based rules language to define who may read and write data.
- Within the scope of this project, all services can be used free of charge.[156]

4.6.6.2 Combining Firebase with Vuex

Firebase can be accessed on different levels. The most straightforward way is to send HTTP requests and process the responses. This may be done e.g. with *Axios*, as discussed in 4.6.4, which is in fact how data access was implemented in early stages of this project.

To make use of *Firebase*'s advanced features as introduced in the preceding section, it is worthwhile to use the *Firebase JavaScript SDK*. It offers a nice and readable API that abstracts the complex and sophisticated low-level technology it applies.

To additionally benefit from *Vue*'s reactivity system and follow the framework's best practices, the vuefire component should be used. It provides *Firebase* bindings for *Vue*—data is available as so-called *references*, which can be read and written to.

The most sophisticated approach, and the one that was ultimately applied in this project, is using the vuexfire component. Remote changes that arrive via web sockets trigger this component to commit mutations, thereby synchronizing the local state. The local application's *Vuex* actions and mutations in return get sent on the wire to synchronize with the database and all other clients in near real-time. Internally, *Vuexfire* creates and commits global mutations with a prefixed

153 Refer to Mammeri 2017.

154 Rouse 2016.

155 Refer to Firebase Docs 2018b.

156 100 simultaneous connections, 1 GB stored, 10 GB/month downloaded. As only JSON data gets stored, this is plentiful.

name, in which it is able to perform additional logic before passing on to the actual mutation.[157]

4.7 Routing with Vue-Router

As programming in *Vue* already implies structuring an application into components, *Vue-router* really just maps these components to routes. In the template, router-view elements determine where they will be rendered. Nested routes map to nested components.

Vue-router has all the features developers might expect, incl. programmatic navigation; named routes and views; redirects; aliases; handing down data from the query string to components; using the HTML5 history mode; and more. It may even be leveraged to implement *lazy loading* of routed components.[158]

Since the router-view components are essentially dynamic components, it is easy to exploit *Vue*'s native transitions to animate navigation smoothly (as will be discussed in 4.12.2).

4.8 Form Validation with Vuelidate

Vuelidate is a lightweight model-based validation library. Once imported, developers can specify which values to validate and which constraints they should comply with. Then, there will be a special object for each value which specifies its current state and validity. Collections and nested models are supported.

The library ships with a lot of the most common validations included. In addition, it allows for simple implementation of *custom validators*. For *Codetask*, a validator was written to check if an email address entered into the registration form is already taken.

It should be noted that it is quite difficult to provide a real polished user experience with form validation. The challenge is finding a balance between updating the data model after each keystroke vs. after focus leaves a given input. E.g., it is usually good to defer visual error indication until the blur or change event fires on an input. Otherwise, the user will get error messages while she is typing, as most constraints will not initially be met. Also, it is good practice to disable a form's submit button until the form as a whole is valid. Now, when a user fills in data field by field and finishes the last one—but does not press tab, enter, or click somewhere outside the input—the button confusingly is still disabled. Even software giants like *Microsoft* get this wrong, bothering with useless error messages while users enter data on login pages.

Also, with *Vuelidate* it is trivial to define validations and determine which validations fail. But to provide an ideal user experience, a tailored error message should be offered for each possible problem, adding up to a lot of boilerplate.

4.9 Authentication with Firebase

To authenticate users with only client-side code, *Firebase Authentication* has been utilized. To implement a traditional email and password authentication in this project, registration and login forms have been created to accept the credentials from the user. They get sent to *Firebase*, which verifies them and returns a token. After a successful sign in, a JavaScript object holding the user's profile information and authorization is available with the token.

Unfortunately, the data associated with a user is restricted. To implement custom features like storing users' score and settings, user data has to get stored in the database as well. This leads to a less-than-ideal parallel design where data for users is split across two sources and has to be combined using the UID (short for "Unique identifier").

157 Morote 2018.

158 Refer to Vue-Router Docs 2018.

In addition to authentication to the login process described above, users can be authenticated with *federated identity providers*, more exactly: existing accounts at *Google*, *Facebook*, *Twitter* or *GitHub*.[159]

The service integrates smoothly with the *Firebase* database and is completely free of charge. Like with the database access, the first implementation consisted of REST calls via *Axios*. Later, these have been replaced by the official SDK, which offers a highly abstracted API and takes care of the low-level work, like storing credentials in the browser's localStorage.

4.10 Reimplementing the Interactive Exercises

The original task data model is quite well composed, implementing a base task class with a subclass for each task type. As *Vue* does not support the concept of inheritance, the model has been redesigned to make use of object composition: A base component which contains the shared code and markup holds a child component of the appropriate task type.

As 4.5 has laid out, there is always at most one task on-screen at any time. To improve accessibility and ease of use, after a task is completed, the view automatically transitions to the next one. This feature can be turned off if users prefer to navigate exclusively via the navigation panel.

4.10.1 Koan Tasks

Instead of implementing an original code editor, the pre-existing code base made use of the embeddable code editor *Ace*. This is a very reasonable decision, as writing such oneself is a difficult and extensive endeavor: "[The] amount of work needed to reach such state is still huge. Even with the know-how it does not seem to be a project which can be approached by a single developer or even a medium-sized company."[160] *Ace* is a sound choice and there is even a wrapper component for *Vue* available. Some of its competitors have produced even more elaborate *Vue* components though, so for the new *koan tasks* the decision was taken in favor of the *CodeMirror* editor. *CodeMirror* is a durable and feature-rich tool that offers sophisticated solutions to the challenges of implementing an in-browser editor.[161]

To implement the *koans*' characteristic gaps, the original implementation used span elements with the contenteditable attribute set to true. While this implementation offers a simple and cheap way to make parts of HTML directly editable, there are numerous edge cases and issues with this approach.[162] Also, from a semantic point of view, the gaps should rather be input elements.

The new implementation is composed of a vue-codemirror component with all code except for the gaps set to read-only and styled in a way that it cannot be focused or selected. The gaps are input elements, with the additional capacity to grow and shrink according to the entered value, as well as some extra styling to indicate the validity of the answer. *CodeMirror* only allows for limited custom modifications, but it does offer a *bookmark* feature, by means of which the input DOM nodes get mounted into the editor, where they retain their inline position and flow correctly if the preceding code changes (i.e. if the user types).

By using inputs as child elements of a form, accessibility automatically benefits, because browsers offer optimized keyboard navigation. The task can easily be navigated with the tab and enter keys. To take it a step further, the first input automatically gains focus after a task gets loaded. After the user has typed the correct answer into an input field, the focus automatically moves to the next one. Additionally, the value cannot be changed anymore after it has been solved initially.[163]

159 Firebase Docs 2018a.

160 Koszulinski 2015a.

161 Refer to Haverbeke 2007; Haverbeke 2011.

162 Refer to Santos 2014; Koszulinski 2015a; Koszulinski 2015b.

163 This was possible in the original implementation and led to some confusing and inconsistent states.

To alleviate the content-related frustrations addressed in 3.3.4, each `input` field comes with a hint element, that shows the expected value as a tooltip on request. The justification for this feature will be discussed in 4.12.1 and 4.13.1. It can be disabled locally in each user's settings or course-wide, as determined by the course creator.

4.10.2 Code Tasks

The basic component has been created and the core functionality is implemented. It shares many traits with the *koan tasks*, e.g. an embedded *CodeMirror* editor. As the findings of the analysis proved this task type to be non-functional however, only minimal effort has been put into its new implementation. As for now, *code tasks* get filtered at the point of course creation (i.e. import).

4.10.3 Video Tasks

Video tasks are the most trivial ones. Basically, they are composed of nothing but an embedded YouTube player. Instead of using the plain player, the *Vue* component `vue-youtube-embed`, which provides a wrapper around the player, has been utilized to improve readability and follow the framework's conventions.

4.11 Enabling Interaction with Disqus

Researchers agree that peer-to-peer communication and student-lecturer interaction is an integral part of every e-learning environment (see p. 6). Therefore, there was no excuse for *Codetask* not to offer this feature: The out-of-the-box comment plug-in *Disqus* was included. It has been embedded in the chapter introduction page, so that a separate chat room is available for each chapter.

This is just a minimal solution and does not come without caveats, e.g. it only allows for very limited customization. Also, including such interaction mechanisms in a system, while being an important feature, can change the social dynamics significantly. Student-to-student interactions can, if they are not moderated, actually have a *negative* impact on the community.[164] Prospectively, a custom solution could be built for *Codetask* and made available in conjunction with appropriate moderation.

4.12 Improving the User Experience

It is difficult to consider UX in isolation. The redesign and subsequent implementation was influenced by the findings of 2.6 across the board; many crucial UX aspects have already been reviewed under the light of information architecture; content structure; data updates; enhanced data model; routing; form validation; authentication; and task redesign. In this section, the focus will be on additional aspects related to UX that have not been sufficiently covered in the preceding sections.

4.12.1 Interface Design

The global navigation in the header is kept as simple as possible and, in accordance with common suggestions, has only five elements in the submenu. All commonly used functionality is reachable with few clicks, the central dashboard is only one click away at all times. For task navigation in the course view, a consistent and minimalist panel was implemented. An indicator shows which task is currently active, so that users are always aware of their current position.

These sequences of tasks that constitute a chapter get expanded by an extra page at the beginning and the end. Owing to the serial position effect, users are most attentive at these points (see p. 23). Inspired by established platforms like *Coursera*, the introductory page is utilized to offer orientation and manage expectations by showing information about the pending chapter (count and distribution of tasks, progress, expected duration). As a shortcut, the navigation allows jumping directly to the next unsolved task (Figure 24).

164 Refer to Phirangee 2016, 13–29.

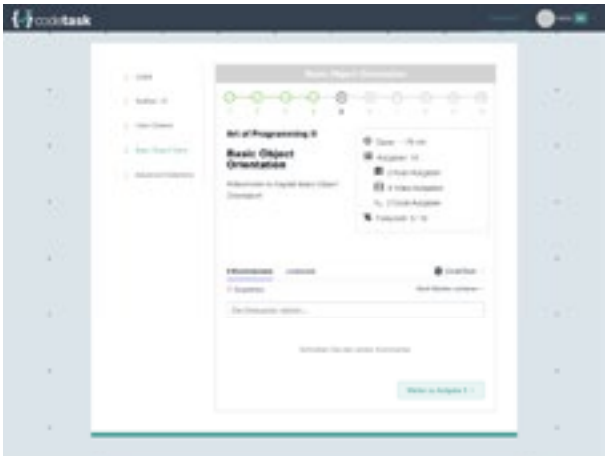


Figure 24 Chapter introduction page with information on tasks, progress, and expected duration .

The final page, on the other hand, yields closure, provides the students with a feeling of achievement and alleviation. Also, by framing the chapters like that, the structure becomes more obvious and it will get experienced as a unit even though its components are now stretched over multiple separate views.

In addition, students can rate the chapter once they reach its end. This provides multiple benefits: It increases motivation to complete chapters, indicates that the user' opinion is heard and appreciated, and offers the chance to improve course materials. Hereby, the frequent criticism of lacking quality control in e-learning is answered (see p. 5).

For reasons of usability, help should easily be available. Hence, there is an informative tooltip available per task type. In the *koan tasks*, where it is most likely for users to run into content-related frustrations, a tooltip is available for each code gap. Educators might be hesitant to give away solutions, as it enables students to cheat. It is, however, very easy to drive users away, and countless other web pages compete for their attention. By offering the solution, students might rather remain in the system, perhaps to be re-engaged in the next task. Through means of gamification, excessive usage of this feature can get discouraged, e.g. the correct solution could only get a yellow styling instead of a green one if the solution was viewed beforehand.

The application allows reversal in that students can reset or quit courses that they are enrolled in. It is possible to revisit tasks after they were solved.

Google's Pete LePage details on how forms are a major UX pain point, especially on mobile. His suggested countermeasures were taken into account, e.g. including semantic input types; using placeholder, autocomplete and autofocus attributes and input labels; pre-populating fields with known data; and visual real-time validation.[165] Also, the forms were designed to be error-proof as far as the submit button is disabled while the input is invalid and offer informative messages as to which requirements are not yet met. Where feasible, autofocus is used to streamline procedures.

When errors do occur, the system remains responsive and an in-page notification informs the user (as opposed to the browser built-ins used in the original implementation).

4.12.2 Design, Layout and Aesthetics

This section discusses the visual aspects that influenced the new UI.

4.12.2.1 Basic Styling with the Element UI Framework

There are several *Vue* libraries that wrap around front-end libraries, containing templates for typography, forms, buttons, and other UI components. Well-known packages and design languages like *Bootstrap* and *material design* are amongst them. For this project, the *Element UI* toolkit was chosen, as it is feature-rich, yet not as overused and "worn out" as its competitors mentioned above. To provide descriptive SVG (short for "Scalable Vector Graphics") icons, the popular *Font Awesome* toolkit was used in addition to *Element*'s limited built-in icons.

Using a well-grounded library like *Element* single-handedly solves a lot of problems and meets demands related to UX and visual design. *Element*

165 Refer to LePage 2018a.

is based on four design principles, each of them closely related to the theory discussed in 2.6.3:

- **Consistency:** Elements are in line with processes and logic of real life; they comply with languages and habits users are used to.
- **Feedback:** Offer visual feedback so users can clearly perceive the current state and the effect of their operations.
- **Efficiency:** Keep processes simple and intuitive; enunciate intentions clearly; make all elements easy to identify and straightforward; prefer recognition over recall.
- **Controllability:** Grant users the freedom to operate, incl. canceling, aborting, and reverting; do not make decisions for the users, give advice at most.

To a certain degree, Element allows for customization. Hereby, the primary color of the color scheme mentioned in the subsequent section could be employed as the primary theme color. Like many front-end libraries, it is implemented with *Sass* internally, so the primary color is merely a variable that can be changed, and even the computation of adjacent colors and gradients (e.g. for a button which is in hover or active state) works dynamically.

4.12.2.2 Adopting the University's Identity

In late 2016, *Hochschule Konstanz University of Applied Sciences* introduced a new corporate design, followed by a mid-2017 website relaunch—consequently, at the time of writing, this new identity is still rather new, yet it slowly gains acceptance by students and gets adopted in publications. An exhaustive style guide is available to facilitate compliant media creation (Figure 25).[166]

To exploit this novel self-image, and create a sense of credibility, belonging and community, *Codetask* follows the style guide where applicable. It uses the general layout, underlying grid system, color palette, typeface and decorative elements as specified. Framed like that, the application feels like an authorized entity of the university—which it is—and hopefully increases students' learning engagement.

Figure 25 Elements of the university's corporate design.

Figure 26 The Codetask artwork was specifically created to fit into the university's branding elements.

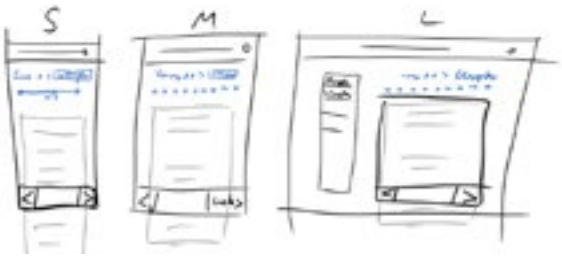


Figure 27 Wireframes outlining the layout changes in conjunction with the major breakpoints.

166 Hochschule Konstanz University of Applied Sciences 2016.

On top of that, there was the need to create a distinct identity, i.e. a logo for *Codetask*. The technical circumstances demand it—a *favicon* or even high-resolution home screen icons will be shown. More importantly though, it implies quality and communicates that it is worth the students' time. To accomplish this, a junior graphic designer was asked to create e-learning themed visuals (Figure 26). The resulting artwork was incorporated in SVG format, as it offers minimal file size and scalable, high-resolution graphics.

Other than the main colors derived from the corporate design (CD), only a limited set of colors has been used, as there is little evidence that color enhances learning, while it does potentially distract some users.[167]

4.12.2.3 Implementing Responsive Web Design

Even ten years ago, mobile devices were already ubiquitous and have shaped the way individuals learn ever since. Back in 2008 already, online education was tailored "to meet the needs of those who wish to learn 'anywhere, anytime'"[168]. This is even more true nowadays, as users browse the web on a broad range of heterogeneous devices. The intended target group of this project, tech-savvy students of computer science, can be expected to use mobile devices as well as desktop environments for learning.

Responsive web design is the endeavor to offer the best possible experience for every device. Several common techniques have been implemented in this project, e.g.:

- Setting the `initial-scale=1` and `width=device-width` attributes.
- Using relative sizes where possible.
- Dispensing large elements with fixed width.
- Adjusting styles with CSS media queries.

Several breakpoints have been defined and some built-in *Element UI* classes come into effect. They offer a grid system to conveniently adjust the content.

LePage points out that you should never hide content completely. If elements do not fit on the screen, an alternative version of the same content should be presented.[169] Consequently, e.g. the indicator of which task is active and how far the user has solved the current chapter shrinks into a simple progress bar on smaller devices (Figure 27).

4.12.2.4 Adding Animations and Transitions

Vue supports animations natively, it is easy to apply transition effects when items are inserted, updated or removed from the DOM. Under the hood, it uses the *FLIP* technique (see p. 22), thereby implicitly optimizing animations.[170]

For *Codetask*, animations from the *Animate.css* library were used. Different subtle transition types are used throughout the application, e.g.:

- Short and snappy unfolding when the main component changes.
- Left-to-right movement of tasks in the course view, mirroring western reading habits.
- A longer animation for entering the *inner application* after login, as it is a significant step semantically, and, more importantly—the animation covers up the delay caused by requesting data (resp. binding *Firebase* references).

Additionally, an animated loading spinner is shown at points where delays are expected, i.e. at the initial page load and after login.

167 Refer to Fahy 2008, 176.

168 Hutchson et al. 2008, 201–220.

169 LePage 2018b.

170 Refer to Vue.js Guide 2018a.

4.13 Gamification

As 2.7 has convincingly proved, gamification goes a long way to improve student motivation, engagement and persistence. Consequently, various game elements have been included in the redesign of the application.

4.13.1 Progress Bars and Enforced Sequential Solving of Tasks

In 4.5, the course structure has already been adjusted. In order to improve UX, only one task is displayed at a time. To create progress dynamic, the access to tasks was changed to be blocking, i.e. only after finishing a task, the following becomes available. Hence, users must *earn* them and work their way up through a chapter. On a technical level, this is implemented with route navigation guards to prevent cheating by simply accessing content via URL.

While this is a tried and tested gamification strategy, it significantly aggravates the content-related frustrations addressed in 3.3.4, turning them into system-related frustrations. In order to counteract this, a helping tooltip has been added (see p. 47). Also, courses should be thoroughly tested before they get released, to avoid unpleasant surprises like broken *YouTube* links.

To reward users after each task solved, a progress bar is always in view and adapts the progress dynamically. On the dashboard view, progress bars show the accumulated advances per course as well as per-chapter progress. Special styling and icons acknowledge completed units.

4.13.2 Points, Badges and Achievements

For each task, users score an number of points. The current score is visible in the header, right next to the profile picture. Derived from the score, students hold a badge with their current level, of which there are six (*Rookie*, *Newcomer*, *Junior*, *Expert*, *Master*, *Evangelist*). Each user has a profile page, where her achievements, points, badges and current enrollments are listed.

Prospectively, more achievements, badges etc. could be implemented for various actions. Also, the point value of each task could become adjustable to make harder tasks more rewarding.

4.13.3 Rank and Competition

Based on the students' score, they get ranked on a leaderboard. Because of how data is wired (*Vue*'s native reactivity system plus *Firebase*'s synchronization), the activities of all users of the system are mirrored in near real-time. When a user outpaces another, i.e. the list order changes, *Vue*'s list transitions make for a smooth and engaging experience.

In order not to take away control from users and respect their privacy, this feature can be disabled.

4.14 Build and Deployment Processes

4.14.1 Continuous Integration with Travis

Travis is an online continuous integration (CI) service which requires little setup and makes it easy to build and test projects hosted on *GitHub*. In the scope of this project, *Travis* could be used at no charge.

As the project is built with *NPM* and *Webpack*, a minimal *Node.js* requirement has been set up. One surprisingly tricky challenge was accessing secret variables like the *Firebase* API key. Such variables by nature cannot get checked into a version-control system (VCS), yet without them, the application cannot be built. They have to be available at run time resp. build time depending on the use case.

```
#!/bin/bash
# ...
echo “Updating env vars...”
export FIREBASE_API_KEY=”AIza5haLtqhVzjl4dAnNA4UauCHIKcBuMALKwlloVNE”
# ...
```

Listing 1 An extract of the helper script export_secret_variables.sh.

```
module.exports = {
  // ...
  FIREBASE_API_KEY: `”${process.env.FIREBASE_API_KEY}”`
  // ...
}
```

Listing 2 An extract of the prod.env.js environment configuration file.

Node.js offers the feature to define environment variables—it is even possible to define different environments, like *development*, *test*, *integration* and *production*. Hence, the solution consists of two steps:

- Export secret variables to the build environment, e.g. by entering them in *Travis*' project settings, or locally with a helper script: `source ./export_secret_variables.sh` (Listing 1).
- Access the *Node.js* environment via `process.env` and include relevant variables in the environment of the application (Listing 2).

4.14.2 Acquisition and Deployment of SSL Certificate

As it is a common web standard and required for potentially making use of service workers at some point[171], a secure HTTPS connection had to be established. Thankfully, the department makes acquiring an SSL certificate free and easy.

The subdomain codetask.in.htwg-konstanz.de had already been registered with port 80 open for access from outside the university's network. So, after setting up the SSL certificate and keys as well as requesting for port 443 to be opened, the application could consequently be served securely at https://codetask.in.htwg-konstanz.de.

4.14.3 Containerization and Shipping with Docker and Dokku

To provide a robust and predictable deployment workflow, the application has been wrapped in a lightweight *Docker* container. *Docker* is a tool to facilitate creating, deploying and running applications with all required dependencies. Containerization guarantees reproducibility—the application will behave exactly the same on the server as on the development machine.[172]

The *Codetask* container is built on top of the official *Node.js* image. The build script passes on the aforementioned secret variables, builds the application, exposes the appropriate ports and finally starts a small *Express.js* script, which in turn serves the application.

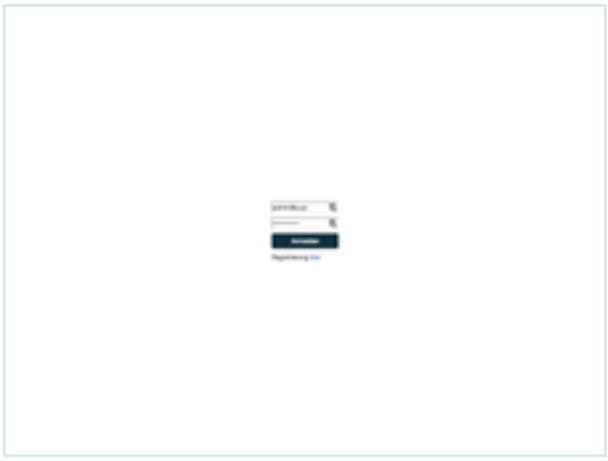
In order to easily ship the container to the Linux server available in the university's network and manage the application's lifecycle, *Dokku* was installed and configured. *Dokku* is a small, free and open source *Platform-as-a-Service* (PaaS) that aims at providing the same services as *Heroku*.[173] After installing the tool on the server and setting up a new project, deploying the *Codetask* application is as simple as executing `git push`. Additionally, *Dokku* stores the secret variables required for the build process, manages SSL certificates, and redirects HTTP traffic to HTTPS.

171 Service workers have full control of all data sent and received by the browser, so the feature is only available with secure connections.

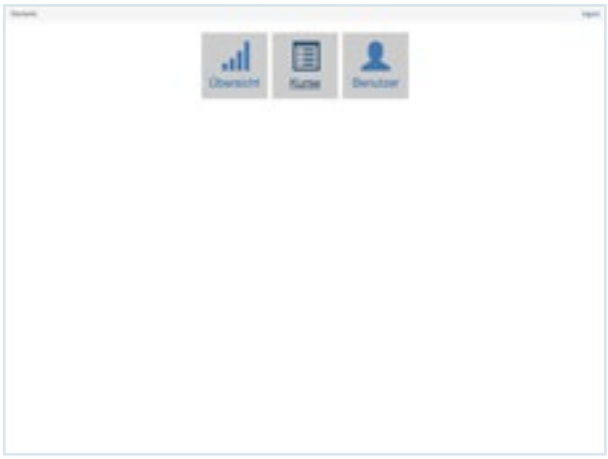
172 Refer to Docker Docs 2015.

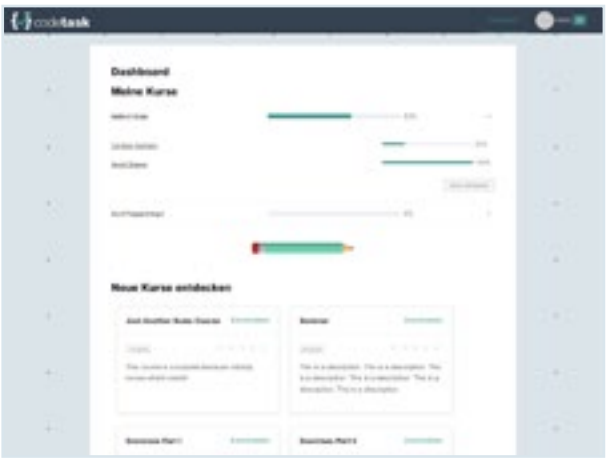
173 Refer to Dokku Docs n.d.

View 1: Login page



View 2: Main dashboard page





View 3: Learning page

Figure 28 Comparison of the original Codetask application and an early version of the new implementation (screenshots).

5 Discussion

At the end of the project, a critical review revisits the themes of this study. Then, recommendations for future work will be given.

5.1 Critical Review

In this section, some light will be shed on which decisions turned out to be beneficial and which proved to be problematic in retrospect.

5.1.1 Choice of Framework and Core Technologies

Developing with *Vue* is, subjectively, very pleasant. It offers separation of concerns in single file components. The learning curve is much less steep than the one of comparable frameworks. It is lightweight, and most notably offers an extremely intuitive reactivity system, paving the way for clean and readable code. The documentation is helpful and there are plenty of available resources and third-party components available. And to large parts, it *just works*—subjectively, a lot less time was spent debugging and struggling with the technology than when using *Angular,* for instance. Presumably, this makes for a lot of the extreme sympathy that the developers community has for the framework.

Though lightweight at its core, *Vue* is extensible, and with the appropriate additions, turns out to be a good fit for a medium-sized application. In particular, *Vuex* seems to be the connecting piece that allows for solid large-scale software. Given these points, the choice for this framework was thoroughly fit for the task at hand.

One of the biggest companies that use *Vue* is *GitLab*. Their resume also is very positive. Not only were they able to gradually introduce the framework without doing a complete rewrite, but also improved time and cost efficiency. From a sales perspective, they confirm that the UX improvements involved in the process made for a better product in general: "We can finally focus on usability and UX, where before we were focusing on little tiny things and code. Now we think about the much bigger picture."[174] The same can certainly be said about *Codetask*.

Also, *GitLab*'s front-end lead Jacob Schatz emphasizes the importance of centralized state: "While Vuex is initially more verbose, it is much more scalable, and will save you tons of time in the long run. Our mistake[s] happened when we changed the data in multiple places."[175] As for the role of *Vuex* in this project, it will be discussed in the following section.

By and large, the other packages used met their expectations as well: *Vue-router*, *Vuelidate* and *Axios* are mature and recommendable tools. *Element UI* was a little fiddly at times; while it offers a lot of built-in features and lays the groundwork for an attractive user interface, it is hard to extend and customize (at least if fine-grained control is desired). Every so often, a bug emerged, e.g. Chinese signs appeared in the view when a certain value was missing.

Webpack is extremely powerful and fuels a state-of-the-art front-end development experience. One would not want to miss all of its features, be it core functionality—transpiling *ES* to JavaScript, *Sass* to CSS, etc.—or convenience add-ons like the integrated development server with in-browser compilation error overlay. It must be noted however, that the setup and configuration are very complex and may prove time-consuming, should the need to customize the workflow arise. As long as the basic setup provided by the Vue CLI suffices, it can be fully recommended.

The integration of *TypeScript* involved several small configuration adjustments which are barely

174 Monterail and You 2017, 66.

175 Schatz 2017.

documented. Also, missing type declarations made it hard to use in combination with some other third-party packages, which slowed down the development process. If static typing is a central aspect of a project, developers are better off choosing *Angular*.

Dokku is a very young tool. At the time of writing, the latest stable version is 0.11.3. Many features are yet to be implemented, and the documentation is quite scarce. Still, for the given use case, everything worked as expected. *Dokku* integrates with *Docker* nicely, does the low-level work of managing certificates, and works reliably after an initial, slightly burdensome setup.

5.1.2 State Management, Complexity and Near Real-Time Synchronization

The benefits of both *Vuex* and *Firebase* have been discussed at length in 4.6.5 and 4.6.6. However, there are quite a few drawbacks involved. Starting with *Vuex*:

- More concepts developers have to understand.
- An additional layer of indirection.
- More boilerplate code.
- Mutations and actions get invoked by passing strings, which adds a source of errors and impedes integrated development environments' (IDE) refactoring and navigation support.
- Whereas plain *Vue* makes it extremely easy to use two-way data binding and translate a local variable into a view element, with the new approach, one has to first import the state (resp. getters) into a component to access state. Then, to modify data, a mutation has to get created, imported and commited for each piece of state.

On top of that, there are *Firebase*'s downsides:

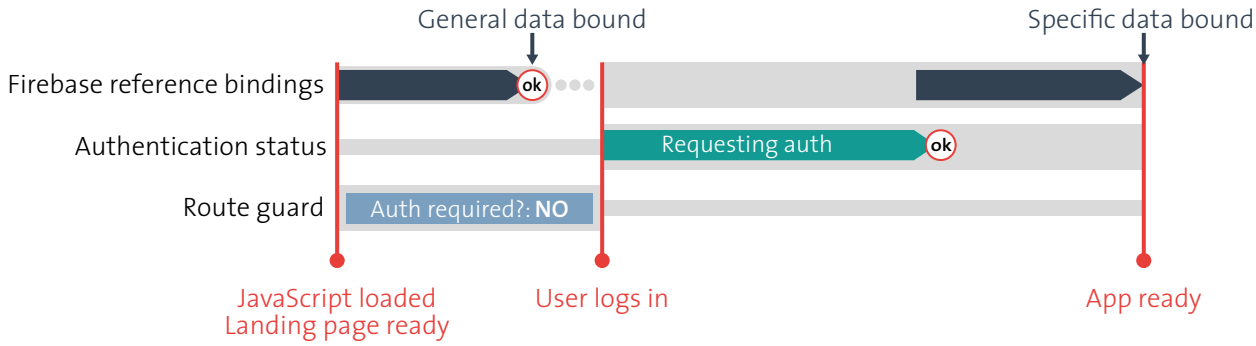
- Dependency on *Google*.
- Documentation is good, but it has a low technical depth.[176] While the high level of abstraction the SDKs and APIs provide is generally good, it is likely that developers will not understand the underlying technical base of the tools they use if documentation is not thorough.
- The *Firebase* SDKs does not implement P*romises*, only a *Promise-like* alternative. It is also a so-called *Thenable*, meaning that it has a then callback function, but there are some differences, e.g. it does not have the finally function that *Promises* have.
- While reading access does not change, writing is now done on the *Firebase* references instead of the actual data structures. This is less readable, convenient, and makes the whole process ambiguous, since there are now two different ways of doing a very similar thing.

Lastly, and most crucially, the process of binding references entails a significant increase in complexity. Some general data has to get bound (e.g. the course data) as well as some user-specific data that cannot be bound before authentication is established (e.g. the user's setting). These processes have been programmatically separated and their completion status is tracked with a flag. Only after both references have been set up, it is feasible to render the actual application, as multiple components depend on the state and given all the interconnections, an earlier partial rendering turned out to be error-prone.

What's more: Authentication, data binding and route protection add up to a complex time-dependent system. As Figure 29 illustrates, the succession is relatively easy when a user enters the application via the landing page. As it is not part of the *inner application*, its route is not protected; also, there is no need for the data bindings to be ready. The page load is not significantly delayed.

176 E.g. the documentation never even fills you in on how it uses web sockets, instead using the term "data synchronization".

Scenario 1: User journey starting at landing page



Scenario 2: Page refresh or direct navigation to internal route

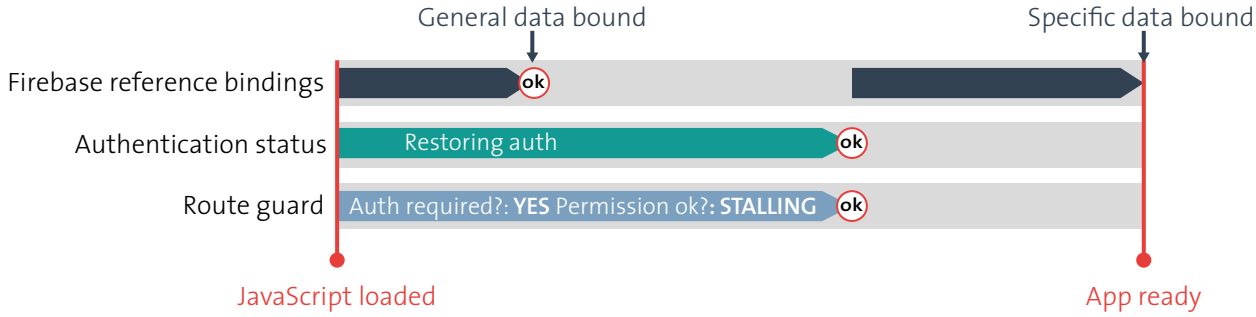


Figure 29 Timeline illustrating the steps involved in the data binding process.

It gets trickier when a user refreshes the page inside the application or directly navigates to an internal route. The immediate execution of the route navigation guard takes place before *Firebase*'s initialization finishes. It cannot determine whether or not to show the requested page, as the authentication state is not ready—so the guard has to get stalled. Even though *Firebase* automatically stores the auth token in the `localStorage`, it renews the login status at page load, which takes some time. While the binding of general data can be started as soon as possible, the binding of user-specific data has to wait for both the general data binding[177] and verified authentication status. To determine the authentication status, the *Firebase* API allows registering an observer to the auth object.[178]

The current, less-than-perfect solution is to delay the whole rendering until the process described above is finished. To take the edge off this inadequacy, a visual spinner is shown instead of just an empty page. Still, a better approach would be to immediately show some content to reduce the *time to interactive*, and subsequently load data.

As has already been pointed out in the theoretical framework, implementing centralized state management is not appropriate for all applications, rather just for large-scaled ones. This is even truer in combination with remote synchronization like *Firebase* offers. It cannot be emphasized enough

177 This is because of how the data is currently modeled, it could likely be improved.

178 Refer to Mammeri 2017.

that these tools come with a prize. While the experience was illuminating from an academic point of view and certainly represents a forward-thinking approach, frankly it is oversized for an application the size of *Codetask*.

So, yes—this setup adds a lot of complexity and boilerplate. However, it also proved to work reliably once it was installed and provided astonishingly reliable synchronization across multiple clients connected at the same time. In a way, it takes the spirit of *reactive programming* and extends it to distributed systems, incl. secure and trackable transactions. Really, it is blurring the line between client and server remarkably.

5.1.3 User Experience and Gamification

Evaluating the effect of the supposed UX improvements is difficult, as it is a very subjective matter. The best way to assess it is by means of A/B testing. Hence, a small number of user tests on a random basis have been made and yielded very positive results. The improvements in usability and design are mostly not consciously perceived, but apparently manifest in a generally positive attitude towards the application.

Going back to Morville's *Honeycomb* model (see p. 19), *Codetask* seems to "tick all the boxes" making for a valuable experience:

- **Useful:** The site allows students to learn a programming language efficiently.
- **Usable:** Usability has been significantly optimized.
- **Desirable:** A professional logo, consistent color palette, fonts, icons, animations etc. get utilized to provide an attractive appearance.
- **Findable:** Information structure is simple; navigation is intuitive.
- **Accessible:** The markup is semantically verbose; layout adjusts to devices responsively; tasks can be navigated using the keyboard only.
- **Credible:** The visual design builds on the university's credibility; users can rate content.

Revisiting the *Hierarchy of needs* pyramid (see p. 20), it is fair to say that the new interface has raised the bar. It is measurably functional, reliable and usable. On top of that, one might argue that it provides a convenient and pleasurable experience. To reach the top of the pyramid and make it *meaningful*, it would be worthwhile to create more diverse original content and improve the platform in a degree beyond the scope of this study.

5.2 Suggestions for Future Work

While research and implementation have been carried out to the best of the author's knowledge, naturally, the scope of this study is limited. There are many possible improvements that could further enhance the *Codetask* application in the future.

5.2.1 Conduct of Usability Tests

In this thesis, a major emphasis has been put on improving UX and usability. Owing to time constraints, the implementation mostly followed theoretical models—only a minimal amount of practical usability testing was conducted.

Testing is supposed to be a crucial aspect of UX however. It may comprise interviews with stake holders, lightning talks, user interviews, ethnographic field research, wireframing, storyboarding, and prototyping.[179] While the resulting insights are most valuable in early development stages, it would still be interesting e.g. to compare the two implementations, determine which improvements turned out as expected and which did not, and further promote usability in general.

179 Refer to Krug 2013, 135–162; Kurtuldu 2018.

5.2.2 Measuring and Improving Performance

Performance is a critical issue when it comes to web frameworks, as we have discussed in 2.3.1. With modern tooling like *Webpack*, it is feasible to implement *code-splitting* and *lazy loading*, thereby improving general performance and in particular the initial page load. Other than that, an SSR library or a service worker could further improve results.

5.2.3 Improving Accessibility and Offline Capability

For reasons already explained, accessibility and offline capability are substantial enablers on the way towards "education for all". These findings were only rudimentarily put to practice by now.

Concerning offline capability, some interesting innovations are currently being developed. PWAs promise to enhance regular websites with features that resemble native apps, incl. sophisticated caching strategies and background sync. The data model has actually been tailored with this perspective in mind—The course data, which changes very rarely, is stored separately from the users' advances that change frequently, allowing for the static data to be aggressively cached.[180] After initially fetching the course data and authenticating, the application could be completely functional without network connection.

Push notifications, as a side note, are another PWA feature that could re-engage users and improve retention rates. Basically, it facilitates sending messages to users even if the application tab is closed (desktop) or the browser app is not running in the foreground (mobile). A 2012 study in fact revealed that students who had received persuasive SMS messages subsequently improved their self-regulated learning.[181] Educators could target the students enrolled in a specific course, or automatically send push notifications when certain conditions are met (e.g. a student's performance is dropping, or she was inactive for a number of consecutive days).

5.2.4 Enhancement of Analysis and Monitoring Capabilities

Tracking the student's activity in more detail would open the door to an improved experience for teachers as well as students:

- Teachers could track advances in a fine-grained manner.
- Students could receive warnings if their activity is insufficient; on the other hand, the reward system could improve motivation by offering achievements for multiple days of activity in a row.

5.2.5 Back-end Repair Work and Reintegration

The findings in the analysis can be used as a guideline to improve the original *Codetask*. It would be relatively straightforward to reintegrate the new front-end with the *Play* back-end afterwards. Of course, if user and data management would completely move back to the local *MySQL* server, all of the special features *Firebase* introduced would be withdrawn. It may therefore be better to leave that part as is.

Being able to directly use the *Scala* interpreter and parser, however, would erase the indirection in the creation of new learning materials. Also, the application would benefit a lot if the *code tasks* were functional, as only two types of tasks quickly get repetitive.

180 It has to be noted however that service workers cannot intercept web socket traffic, so the current Vuexfire-based implementation would have to get adapted.

181 Refer to Panigrahi 2017, 204.

5.2.6 Adding Further Task Types

To elaborate on the point made above: More variety in the content would greatly improve the user experience, and the *koan tasks* are very restrictive due to them being actual unit tests. The data model allows for easy extension—the new types could even be really straightforward. E.g., a non-interactive page that offers explanatory text and visuals would loosen up the current drill. Simply using some educational visual imagery would upgrade the application's appeal and efficiency, as vision is the dominant sense and the most direct way to perception.[182] Research suggests that graphics and visual images play an important role in learning, lead to a better understanding of the material and can better convey ideas than textual descriptions alone.[183]

Furthermore, quizzes could be added—either as self-contained tasks or in the context of other tasks, to test the students' understanding of the content. In principle, it is also possible to enhance the application to teach other languages than *Scala*.

182 Refer to Soueidan 2013.

183 Refer to David and Glore 2010.

6 Summary and Conclusion

This study has explored the question of which factors determine learning success and how they can get employed in the context of online education. The theoretical considerations were put into practice in the implementation of a web-based e-learning application that comprises interactive programming exercises. In the closing chapter, all points that have been made will briefly be revisited, before a final conclusion is drawn.

6.1 Summary

In Chapter 2, a comprehensive theoretical groundwork was laid. The key terms and definitions regarding online education were introduced and the distinction between intrinsic and extrinsic motivation was shortly addressed. Next, the opportunities and challenges of e-learning were debated: Most importantly, e-learning provides education regardless of geographic location, time or individual qualification. On the downside, it faces several challenges, e.g. by imposing technical and organizational requirements on both students and teachers. Balancing strengths and weaknesses, broad consensus is that e-learning fundamentally improves students' learning outcomes.

Following, technical and organizational aspects were discussed: In the process of developing online courses, it is most important to consider course structure, content presentation, interaction and feedback. Web accessibility and offline capability bear a significant potential to overcome educational barriers of different kinds.

In two more technical sections, the current state of web development was investigated, and the three most promising front-end frameworks were analyzed and compared: *React* is clearly dominant, with *Vue* being "the biggest threat to *React's* crown" and receiving unparalleled sympathy from the developer community. The significance of *Angular* on the contrary seems to be fading.

To lay some more technical groundwork, cookie- and token-based authentication were juxtaposed and the concept of centralized state management was introduced.

The ensuing two sections elaborated on concepts and tools that can increase engagement and motivation in general and have particular relevance in the e-learning context. For a start, a definition and multiple models to measure user experience were given: The *Honeycomb* model e.g. argues that in order to be meaningful and valuable, information must be useful, usable, desirable, findable, accessible, and credible. Visual design influences whether users intuitively understand an interface or if cognitive load gets in the way of achieving the actual objectives. Animations can be used as a means of adding life to the page, but they come with some caveats in respect of performance. Regarding the structural layout of interfaces or content, less is more, as *Miller's law* highlights. When designing an interface, many rules and guidelines should be followed, which according to Steve Krug can be summarized in the sentence "Don't make me think".

Afterwards, a review of gamification ensued: Through applying game mechanics in a non-gaming context, it is possible to promote engagement and intrinsic motivation in students. The underlying psychological principles of sequencing, challenges, status, and achievements translate into elements like toolbars and rewards like points, badges or trophies. Studies confirm the efficacy of such methods.

In Chapter 3, an exhaustive analysis of the pre-existing application *CodeTask* was undertaken: First, the requirements, circumstances and business logic were illustrated. Then, the technical shortcomings were evaluated. Most severely, the application was not properly deployed, broke a number of architectural conventions and turned out to be malfunctioning in parts. From the perspective of usability, the general visual bleakness, broken UI

conventions and content-related deficiencies were most frustrating.

In a comparative study, some of the most successful real-world e-learning platforms were analyzed and compared. Many similarities in design and structure were identified, as well as elements of good user experience and applied gamification, as discussed above.

Chapter 4 gave a full account of implementing the new application. After cumbersome first steps, shaped by the glitches of the pre-existing code base, it was decided to decouple the new front-end from the old back-end and go serverless with the help of the *Firebase* suite. To build the application, a large technology stack was employed, incl. *Webpack*, *Vue-CLI*, *Vue-Router*, *Vuelidate*, *Vuex*, *Vuexfire*, *Firebase* (for database as well as authentication), *Disqus*, *Travis*, *Docker* and *Dokku*. At last, an SSL certificate was acquired and used to deploy the containerized application via HTTPS.

Concerning the information architecture, which is directly mirrored in the component layout, the contents were broken down into smaller units than before. At the heart of the application, the new course view was built. The existing types of exercises were recreated, improving them where possible.

To add type-safety and provide a backbone for the now serverless data, as it were, *TypeScript* was added to the project. Data changes within the application get propagated automatically by *Vue*'s simple yet powerful reactivity system. On top of that, *Vuex* was added and introduced an additional level of indirection with a centralized store. To be able to synchronize data with the database and other clients in near real-time, *Firebase* was configured and *Vuexfire* was set up.

Numerous major and minor supposed improvements of usability and interface design were made. Visually, the interface redesign strived for consistency, feedback, efficiency and controllability. The university's identity has been adopted and augmented with an original brand logo and e-learning themed artwork. Additionally, the general layout was implemented so as to adopt responsively to various devices, and some animations were added. These animations are mostly semantic-decorative, e.g. completed exercises will horizontally slide out of view, making room for the next one. Some animations are also covering up delays from fetching data.

Gamification elements in the form of progress bars, enforced task sequence, scoring system and a leaderboard were leveraged to increase engagement.

Seemingly, these improvements in terms of user experience and gamification succeeded in their aim: The theoretical models that have been introduced in Chapter 2 were drawn on in order to measure the user experience, suggesting a significant improvement. A small number of user tests on a random basis yielded positive results in regard of the game mechanics employed.

Looking back, Chapter 5's verdicts ranged from excellent (*Vue*, *Vue-Router*, *Vuelidate*, *Axios*, *Docker*) to recommendable with some reservations (*Webpack, TypeScript, Element UI, Firebase* resp. *Vuexfire, Dokku*). The core framework keeps its promise of a quick start, offers a very intuitive reactivity system and implicitly promotes clean code. The documentation, available resources and components are more than enough to build small to medium-sized applications. *Vuex* state management seems to be the connecting piece that enables large-scale applications as well. Companies which use *Vue* in production report that its simplicity and ease-of-use enable them to think about the big picture and spend their energy on improving the user experience instead of debugging and struggling with technology.

Finally, some suggestions for future work on the *Codetask* application and similar endeavors were given.

6.2 Conclusion

Building an e-learning system makes for an exciting exercise in software engineering, as it strives to solve a significant real-world problem and touches many aspects that go beyond technology. Numerous factors that determine learning success were identified and put into practice in the process of implementing a web-based e-learning application. While the project was focused on a relatively small application with a restricted user base, the key ideas and outcomes may well be applied in larger-sized real-world applications.

The approachability of modern tools and frameworks allows developers to focus on the underlying psychological factors and common themes that connect user experience, learning motivation, commitment and engagement. When trying to answer the central question of which factors determine learning success in an online context, it seems they are not so different from traditional educational requirements: a distraction-free working environment, interaction with peers and instructors, well-structured materials, and appropriate challenges and rewards.

To facilitate successful learning, a computer-based learning environment has to satisfy the same "basic needs" of the users by providing a reliable and unobtrusive technical base. On top of that, this context allows to exploit certain learning motivations to a much higher degree than a traditional classroom-setting, e.g. the gratification of achievements and social comparison translate into native gamification elements like progress bars, scoring systems, and leaderboards. Refining the user experience of the learning process can significantly change the attitude students have towards an e-learning system and hence also improve engagement and efficiency—if an activity is rated positively and classified as meaningful and valuable, students will naturally spend more time learning of their own accord.

All things considered, to provide an engaging, efficient learning experience, e-learning has to get designed and developed with the user in mind. As Davis et al. put it: "Ultimately, as with any educational system, online learning is fundamentally a human endeavor, with technology available to support the agreed-upon principles and goals, not vice-versa."[184]

184 Davis et al. 2008, 140.

List of References

Anderson, Stephen P. Anderson. 2009. In Defense of Eye Candy. Available at <http://alistapart.com/article/indefenseofeyecandy>. Accessed 17 February 2018.

Anderson, Stephen P. Anderson. 2011. *Seductive Interaction Design: Creating Playful, Fun, and Effective User Experiences*. 1st ed. Berkeley, CA: New Riders.

Andrade, Mário R. 2013. The Gutenberg Diagram in Web Design. *User Experience*. Available at <https://medium.com/user-experience-3/the-gutenberg-diagram-in-web-design-e5347c172627>. Accessed 13 February 2018.

Bali, Maha. 2014. MOOC pedagogy: Gleaning good practice from existing MOOCs. *MERLOT Journal of Online Learning and Teaching* 10 (1): 44–56.

Banna, Jinan C., Meng-Fen Grace Lin, Maria Stewart, and Marie K. Fialkowski. 2015. Interaction matters: Strategies to promote engaged learning in an online introductory nutrition course. *Journal of Online Learning and Teaching* 11. Available at <http://scholarspace.manoa.hawaii.edu/handle/10125/39586>. Accessed 15 February 2018.

Bataev, A. V. 2017. Practice-oriented online-courses as a factor of increasing the quality of engineering and economic education. In *2017 IEEE VI Forum Strategic Partnership of Universities and Enterprises of Hi-Tech Branches (Science. Education. Innovations) (SPUE)*, 107–110.

BCIT Learning and Teaching Centre. 2001. Ensuring Usability for Online Courses. British Columbia Institute of Technology.

Berners-Lee, Tim. 1999. Glossary - Weaving the Web. Available at <https://www.w3.org/People/Berners-Lee/Weaving/glossary.html>. Accessed 20 February 2018.

Bernik, A., D. Radoševic, and G. Bubaš. 2017. Introducing gamification into e-learning university courses. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 711–716.

Betts, Kristen, Alex H. Cohen, Daniel P. Veit, Henry C. Alphin Jr., Chanel Broadus, and Dan Allen. 2013. Introduction to the Special Section on Integrating Accessibility Into Online Learning. In *Journal of Asynchronous Learning Networks*, 3:. Vol. 3. Online Learning 17. Available at <https://olj.onlinelearningconsortium.org/index.php/olj/article/view/378>. Accessed 20 February 2018.

Betts, Kristen, Mark Riccobono, and Bill Welsh. 2013. Introduction to the Special Section on Integrating Accessibility Into Online Learning. In *Journal of Asynchronous Learning Networks*, 3:. Vol. 3. Online Learning 17. Available at <https://olj.onlinelearningconsortium.org/index.php/olj/article/view/378>. Accessed 20 February 2018.

Bright, Sarah. 2015. How to Successfully "Gamify" Your Online Course. *E-Learning Software*. Available at <https://www.ispringsolutions.com/blog/how-to-successfully-gamify-your-online-course/>. Accessed 17 February 2018.

Caplan, Dean, and Roger Graham. 2008. The Development of Online Courses. In *The Theory and Practice of Online Learning*, edited by Terry Anderson. Athabasca University Press.

Chen, Frank. 2016. Frank Chen's Answer to 'What technology platform are Coursera, edX, Khan Academy or any other MOOC platforms built on?' - Quora. *Quora*. Available at <https://www.quora.com/What-technology-platform-are-Coursera-edX-Khan-Academy-or-any-other-MOOC-platforms-built-on>. Accessed 12 February 2018.

Cole, Michele T., Daniel J. Shelley, and Louis B. Swartz. 2014. Online instruction, e-learning, and student satisfaction: A three year study. *The International Review of Research in Open and Distributed Learning* 15 (6). Available at <http://www.irrodl.org/index.php/irrodl/article/view/1748>. Accessed 20 February 2018.

Croxton, Rebecca A. 2014. The Role of Interactivity in Student Satisfaction and Persistence in Online Learning. *Journal of Online Learning and Teaching* 10 (2).

Damphousse, Robert. 2015. Token Based Authentication for Single Page Apps (SPAs). *Stormpath User Identity API*. Available at <https://stormpath.com/blog/token-auth-spa>. Accessed 13 February 2018.

David, Alicia, and Peyton Glore. 2010. The Impact of Design and Aesthetics on Usability, Credibility, and Learning in an Online Environment. *Online Journal of Distance Learning Administration* 13 (4).

Davids, Mogamat Razeen, Usuf M. E. Chikte, and Mitchell L. Halperin. 2014. Effect of improving the usability of an e-learning resource: a randomized trial. *Advances in Physiology Education* 38 (2): 155–160.

Davidson, Jeff. 2017. The Most Important Rule in UX Design that Everyone Breaks. *Prototypr*. Available at <https://blog.prototypr.io/the-most-important-rule-in-ux-design-that-everyone-breaks-1c1cb188931>. Accessed 17 February 2018.

Davis, Alan, Paul Little, and Brian Stewart. 2008. Developing an Infrastructure for Online Learning. In *The Theory and Practice of Online Learning*, edited by Terry Anderson. Athabasca University Press.

Dimandt, Dmitrii. 2017. The broken promise of Web Components. *A Place Where Even Mammoths Fly*. Available at <https://dmitriid.com/blog/2017/03/the-broken-promise-of-web-components/>. Accessed 13 February 2018.

DIN Deutsches Institut für Normung. 2000. *Ergonomic requirements for office work with visual display terminals (VDTs) - Part 12: Presentation of information (ISO 9241-12:1998); German version EN ISO 9241-12:1998*.

DIN Deutsches Institut für Normung. 2017. *Ergonomics of human-system interaction - Part 11: Usability: Definitions and concepts (ISO/DIS 9241-11.2:2016); German and English version prEN ISO 9241-11:2016*. DIN Deutsches Institut für Normung.

DIN Deutsches Institut für Normung. 2011. *Ergonomics of human-system interaction - Part 210: Human-centred design for interactive systems (ISO 9241-210:2010); German version EN ISO 9241-210:2010*. DIN Deutsches Institut für Normung.

Docker Docs. 2015. What is Docker? Available at <https://www.docker.com/what-docker>. Accessed 14 February 2018.

Dokku Docs. n.d. Dokku - The smallest PaaS implementation you've ever seen. Available at <http://dokku.viewdocs.io/dokku/getting-started/installation/>. Accessed 14 February 2018.

Elgamal, A. F., H. A. Abas, and E.-S. M. Baladoh. 2013. An interactive e-learning system for improving web programming skills. *Education and Information Technologies* 18 (1): 29–46.

Facebook. 2014. Flux | Application Architecture for Building User Interfaces. Available at <http://facebook.github.io/flux/index.html>. Accessed 14 February 2018.

Fahy, Patrick J. 2008. Characteristics of Interactive Online Learning Media. In *The Theory and Practice of Online Learning*, edited by Terry Anderson. Athabasca University Press.

Fang, Xiaonan, Huaxiang Zhang, and Yunchen Sun. 2014. A Programming Related Courses' E-learning Platform Based on Online Judge. In *Frontier and Future Development of Information Technology in Medicine and Education*, 3419–3423. Lecture Notes in Electrical Engineering. Springer, Dordrecht. Available at <https://link.springer.com/chapter/10.1007/978-94-007-7618-0_443>. Accessed 17 February 2018.

Ferati, Mexhid, Njomza Mripa, and Ridvan Bunjaku. 2016. Accessibility of MOOCs for Blind People in Developing Non-English Speaking Countries. In *Advances in Design for Inclusion*, 519–528. Advances in Intelligent Systems and Computing. Springer, Cham. Available at <https://link.springer.com/chapter/10.1007/978-3-319-41962-6_46>. Accessed 20 February 2018.

Firebase Docs. 2018a. Firebase Authentication. Available at <https://firebase.google.com/docs/auth/>. Accessed 16 February 2018.

Firebase Docs. 2018b. Firebase Realtime Database | Firebase Realtime Database. Available at <https://firebase.google.com/docs/database/>. Accessed 14 February 2018.

Fowler, Martin, Kent Beck, John Brant, William Opdyke, Don Roberts, and Erich Gamma. 1999. *Refactoring: Improving the Design of Existing Code*. 1 edition. Reading, MA: Addison-Wesley Professional.

Gaunt, Matt. 2018. The Web Push Protocol | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/push-notifications/web-push-protocol>. Accessed 13 February 2018.

Gilbert, Brittany. 2015. Online Learning Revealing the Benefits and Challenges. *Education Masters*. Available at <https://fisherpub.sjfc.edu/education_ETD_masters/303>.

Gore, Anthony. 2017a. 4 AJAX Patterns For Vue.js Apps. Available at <https://vuejsdevelopers.com/2017/08/28/vue-js-ajax-recipes/>. Accessed 15 February 2018.

Gore, Anthony. 2017b. WTF is Vuex? A Beginner's Guide To Vue's Application Data Store. *Vue.js Developers*. Available at <http://vuejsdevelopers.com/2017/05/15/vue-js-what-is-vuex/>. Accessed 14 February 2018.

Greeff, Jacob, Reolyn Heymann, Murray Heymann, and Heymann, Carl. 2017. Codebreakers: Designing and Developing a Serious Game fortheTeachingofInformationTheory. In *Advances in Web-Based Learning – ICWL 2017: 16th International Conference, Cape Town, South Africa, September 20-22, 2017, Proceedings*, edited by Haoran Xie, Elvira Popescu, Gerhard Hancke, and Baltasar Fernández Manjón. Information Sys-

tems and Applications, incl. Internet/Web, and HCI. Springer International Publishing. Available at <// www.springer.com/de/book/9783319667324>. Accessed 17 February 2018.

Greif, Sacha. 2017. I just asked 23,000 developers what they think of JavaScript. Here's what I learned. *freeCodeCamp*. Available at <https://medium.freecodecamp.org/i-just-asked-23-000-developers-what-they-think-of-javascript-heres-what-i-learned-9a06b61998fa>. Accessed 12 February 2018.

Greif, Sacha, Raphaël Benitte, and Michael Rambeau. 2017. The State of JavaScript 2017. Available at <http://stateofjs.com>. Accessed 12 February 2018.

Growth Engineering. 2016. 10 Uses for Gamification in Online Learning. Available at <http://www.growthengineering.co.uk/10-uses-for-gamification-in-online-learning/>. Accessed 18 February 2018.

Haverbeke, Marijn. 2007. Implementing a Syntax-Highlighting JavaScript Editor In JavaScript. Available at <http://codemirror.net/1/story.html>. Accessed 13 February 2018.

Haverbeke, Marijn. 2011. (Re-) Implementing A Syntax-Highlighting Editor in JavaScript. Available at <http://codemirror.net/doc/internals.html>. Accessed 13 February 2018.

Herald, Baye, and Janice Thorpe. 2016. User Experience (UX), Learner Experience (LX) and Usability in Online Learning.

Hewitt, Jim. 2015. E-Learning. *Encyclopedia of Science Education*. Springer, Dordrecht. Available at <https://link.springer.com/referenceworkentry/10.1007/978-94-007-2150-0_36>. Accessed 15 February 2018.

Hochschule Konstanz University of Applied Sciences. 2016. Markenportal der HTWG | Hochschule Konstanz. Available at <https://onlinemanual.htwg-konstanz.de/>. Accessed 15 February 2018.

Hutchson, Maureen, Tony Tin, and Cao Yang. 2008. 'In-your-pocket' and 'On-the-fly:' Meeting the Needs of Today's New Generation of Online Learners with Mobile Learning Technology. In *The Theory and Practice of Online Learning*, edited by Terry Anderson. Athabasca University Press.

Iglesias Cancio, Iñaki. 2014. The application of gamification methodologies to e-learning contexts in order to enhance its student's performance, motivation, and collaboration. Available at <http://openaccess.uoc.edu/webapps/o2/handle/10609/32621>. Accessed 17 February 2018.

Indiana University. n.d. Developing an Online Course. Available at <https://canvas.ucdavis.edu/courses/34528/pages/developing-an-online-course>. Accessed 20 February 2018.

Iñigo, Emilia. 2015. Gamification: A Better Way of Reaching Online Learners. Available at <https://www.shiftelearning.com/blog/gamification-a-better-way-of-reaching-online-learners>. Accessed 17 February 2018.

Iravanian, Sina. 2012. Programming Koans: One of the best ways to learn a new language or framework. *Logging Experiences*. Available at <https://sinairv.wordpress.com/2012/05/27/programming-koans-one-of-the-best-ways-to-learn-a-new-language-or-framework/>. Accessed 12 February 2018.

Jeffels, Peter. 2011. Usability, in relation to e-learning projects.

Jehle, Daniel. 2016. Entwicklung einer modernen interaktiven Lernplattform für Scala mit Scala und Polymer.

Kaplan, Andreas M., and Michael Haenlein. 2016. Higher education and the digital revolution: About MOOCs, SPOCs, social media, and the Cookie Monster. *Business Horizons* 59 (4): 441–450.

Katambur, Deepa. 2017. Gamification of Learning: A Strategy for Online Course Completion. *Custom Training and E-learning, Anywhere Anytime!* Available at <https://blog.commlabindia.com/elearning-design/gamification-for-online-course-completions>. Accessed 17 February 2018.

Kearns, Lorna R. 2012. Student Assessment in Online Learning: Challenges and Effective Practices. *Journal of Online Learning and Teaching* 8 (3).

Kearns, Lorna R., Barbara A. Frey, and Gabriel McMorland. 2013. Designing Online Courses for Screen Reader Users. In *Journal of Asynchronous Learning Networks*, 3:. Vol. 3. Online Learning 17. Available at <https://olj.onlinelearningconsortium.org/index.php/olj/article/view/378>. Accessed 20 February 2018.

Kiget, Nicholas Kipkurui, G. Wanyembi, and Anselemo Ikoha Peters. 2014. Evaluating Usability of E-Learning Systems in Universities. *International Journal of Advanced Computer Science and Applications* 5 (8).

Korotya, Eugeniya. 2017. 5 Best JavaScript Frameworks in 2017. *Hacker Noon*. Available at <https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282>. Accessed 14 February 2018.

Koszulinski, Piotrek. 2015a. ContentEditable — The Good, the Bad and the Ugly. *Content Uneditable*. Available at <https://medium.com/content-uneditable/contenteditable-the-good-the-bad-and-the-ugly-261a38555e9c>. Accessed 13 February 2018.

Koszulinski, Piotrek. 2015b. Fixing ContentEditable. *Content Uneditable*. Available at <https://medium.com/content-uneditable/fixing-contenteditable-1a9a5073c35d>. Accessed 13 February 2018.

Krajka, Bartosz. 2015. The difference between Virtual DOM and DOM. *React Kung Fu*. Available at <http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/>. Accessed 18 February 2018.

Krug, Steve. 2013. *Don't Make Me Think: A Common Sense Approach to Web Usability*. Revised edition. Berkeley, Calif.: New Riders.

Kukic, Ado. 2016. Cookies vs Tokens: The Definitive Guide. *Autho - Blog*. Available at <https://autho.com/blog/cookies-vs-tokens-definitive-guide/>. Accessed 13 February 2018.

Kurtuldu, Mustafa. 2018. Basics of UX | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/ux-basics/>. Accessed 15 February 2018.

Lane, Kenneth E., and Stephen Hull. 2013. Infrastructure for Web-Based Learning. In *Web-Based Learning: Theory, Research, and Practice*, edited by Harold F. O'Neil and Ray S. Perez. Routledge.

Law, Kris M. Y., Victor C. S. Lee, and Y. T. Yu. 2010. Learning motivation in e-learning facilitated computer programming courses. *Computers & Education* 55 (1): 218–228.

Lawson, Bruce. 2017. World Wide Web, Not Wealthy Western Web (Part 1). *Smashing Magazine*. Available at <https://www.smashingmagazine.com/2017/03/world-wide-web-not-wealthy-western-web-part-1/>. Accessed 18 February 2018.

Lee-Post, Anita, and Holly Hapke. 2017. Online Learning Integrity Approaches: Current Practices and Future Solutions. *Journal of Online Learning and Teaching* 21 (1). Available at <https://olj.onlinelearningconsortium.org/index.php/olj/article/view/843>. Accessed 15 February 2018.

Lehr, Andreas. 2012. Usability von eLearning-Anwendungen. *CHECK.point eLearning*. Available at <https://www.checkpoint-elearning.de/corporate-elearning/interviews/usability-von-elearning-anwendungen>. Accessed 16 February 2018.

LePage, Pete. 2018a. Create Amazing Forms | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/input/forms/>. Accessed 20 February 2018.

LePage, Pete. 2018b. Responsive Web Design Basics | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/responsive/>. Accessed 15 February 2018.

Lewin, Tamar. 2012. Universities Reshaping Education on the Web. *The New York Times*.

Lewis, Paul. 2018a. Animating Between Views | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/animations/animating-between-views>. Accessed 18 February 2018.

Lewis, Paul. 2018b. Animations | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/animations/>. Accessed 18 February 2018.

Lewis, Paul. 2015a. FLIP Your Animations. Available at <https://aerotwist.com/blog/flip-your-animations/>. Accessed 13 February 2018.

Lewis, Paul. 2018c. The Basics of Easing | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/animations/the-basics-of-easing>. Accessed 18 February 2018.

Lewis, Paul. 2015b. The Cost of Frameworks. Available at <https://aerotwist.com/blog/the-cost-of-frameworks/>. Accessed 13 February 2018.

Lewis, Paul, and Sam Thorogood. 2018. Animations and Performance | Web Fundamentals. *Google Developers*. Available at <https://developers.google.com/web/fundamentals/design-and-ux/animations/animations-and-performance>. Accessed 18 February 2018.

Lister, Meaghan. 2014. Trends in the Design of E-Learning and Online Learning. *Journal of Online Learning and Teaching* 10 (4).

Lopes, Rui Pedro, and Cristina Mesquita. 2015. Evaluation of a Gamification Methodology in Higher Education. *EDULEARN15 Proceedings*: 6996–7005.

López, Leticia Lafuente. 2016. Usability And User Experience In Training. *eLearning Industry*. Available at <https://elearningindustry.com/usability-and-user-experience-training>. Accessed 16 February 2018.

Lynch, Patrick. 2009. Visual Decision Making. Available at <http://alistapart.com/article/visual-decision-making>. Accessed 17 February 2018.

MacKenzie, Lydia, and Kim Ballard. 2015. Can Using Individual Online Interactive Activities Enhance Exam Results? *Journal of Online Learning and Teaching* 11 (2). t.

Mammeri, Anas. 2017. Vue 2 + Firebase: How to build a Vue app with Firebase authentication system in 15 minutes. Available at <https://medium.com/@anas.mammeri/vue-2-firebase-how-to-build-a-vue-app-with-firebase-authentication-system-in-15-minutes-fdce6f289c3c>. Accessed 18 February 2018.

Marco, Félix Albertos, Víctor M. R. Penichet, and José A. Gallud. 2015. What Happens when Students Go Offline in Mobile Devices? In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, 1199–1206. MobileHCI '15. New York, NY, USA: ACM. Available at <http://doi.acm.org/10.1145/2786567.2801609>. Accessed 20 February 2018.

Mendoza-Gonzalez, Ricardo. 2016. *User-Centered Design Strategies for Massive Open Online Courses (MOOCs)*. IGI Global.

Miller, G. A. 1956. The magical number seven plus or minus two: some limits on our capacity for processing information. *Psychological Review* 63 (2): 81–97.

Milovanovic, Slavoljub. 2010. Opportunities and Challenges of Electronic Learning. *Economics and Organization* 2 (7).

Monterail, and Evan You. 2017. *State of Vue.js 2017*. Available at <https://www.monterail.com/state-of-vuejs-report>. Accessed 15 February 2018.

Morelli, Brandon. 2017. Angular vs. React: Which Is Better for Web Development? *codeburst*. Available at <https://codeburst.io/angular-vs-react-which-is-better-for-web-development-e0dd1fefab5b>. Accessed 14 February 2018.

Morote, Eduardo San Martin. 2018. *vuexfire: Firebase bindings for Vuex (Vue.js)*. JavaScript. Available at <https://github.com/posva/vuexfire>. Accessed 14 February 2018.

Morris, Libby V. 2013. MOOCs, Emerging Technologies, and Quality. *Innovative Higher Education* 38 (4): 251–252.

Morville, Peter. 2006. Information Architecture 3.0. *Semantic Studios*. Available at <http://semanticstudios.com/information_architecture_30/>. Accessed 17 February 2018.

Morville, Peter. 2004. User Experience Design. *Semantic Studios*. Available at <http://semanticstudios.com/user_experience_design/>. Accessed 16 February 2018.

Morville, Peter, and Louis Rosenfeld. 2006. *[Information Architecture For The World Wide Web] By Morville, Peter (Author) Dec-2006 [Paperback] Information Architecture for the World Wide Web*. O'Reilly Media, Inc, USA.

Neuhaus, Jens. 2017. Angular vs. React vs. Vue: A 2017 comparison. *unicorn.supplies*. Available at <https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176>. Accessed 14 February 2018.

Nguyen, Tuan. 2015. The Effectiveness of Online Learning: Beyond No Significant Difference and Future Horizons. *Journal of Online Learning and Teaching* 11 (2).

Panigrahi, Ritanjali. 2017. Online Learning: Improving the Learning Outcomes. In *Proceedings of the 2017 ACM SIGMIS Conference on Computers and People Research*, 203–204. SIGMIS-CPR '17. New York, NY, USA: ACM. Available at <http://doi.acm.org/10.1145/3084381.3084434>. Accessed 20 February 2018.

Papa, John. 2017. Vue.js with TypeScript. Available at <https://johnpapa.net/vue-typescript/>. Accessed 18 February 2018.

Penfold, Steve. 2016. 3 Simple ways to gamify your online learning. Available at <https://www.docebo.com/2016/06/02/3-simple-ways-gamify-online-learning/>. Accessed 17 February 2018.

Phirangee, Krystle. 2016. Students' Perceptions of Learner-Learner Interactions that Weaken a Sense of Community in an Online Learning Environment. *Online Learning* 20 (4). Available at <https://olj.onlinelearningconsortium.org/index.php/olj/article/view/1053>. Accessed 20 February 2018.

Rauschmayer, Axel. 2015. What happened to Web Components? *2ality*. Available at <http://2ality.com/2015/08/web-component-status.html>. Accessed 13 February 2018.

Renz, J., A. Shams, and C. Meinel. 2017. Offline-Enabled Web-based E-Learning for Improved User Experience in Africa. In *2017 IEEE AFRICON*, 736–742.

Robinson, Rhonda, Michael Molenda, and Landra Rezabek. 2013. Facilitating Learning. In *Educational Technology: A Definition with Commentary*, edited by Al Januszewski and Michael Molenda. Routledge.

Rouse, Margaret. 2016. What is real time? - Definition from WhatIs.com. *WhatIs.com*. Available at <http://whatis.techtarget.com/definition/real-time>. Accessed 14 February 2018.

Sanchez-Gordon, Sandra, and Sergio Luján-Mora. 2014. Web Accessibility Requirements for Massive Open Online Courses. 529–534. Available at <http://desarrolloweb.dlsi.ua.es/moocs/web-accessibility-requirements-massive-open-online-courses>. Accessed 20 February 2018.

Santos, Nick. 2014. Why ContentEditable is Terrible. *Medium Engineering*. Available at <https://medium.engineering/why-contenteditable-is-terrible-122d8a40e480>. Accessed 13 February 2018.

Schae, Jacek. 2017. A Real-World Comparison of Front-End Frameworks with Benchmarks. *freeCodeCamp*. Available at <https://medium.freecodecamp.org/a-real-world-comparison-of-front-end-frameworks-with-benchmarks-e1cb62fd526c>. Accessed 14 February 2018.

Schatz, Jacob. 2017. How we do Vue: one year later. *GitLab*. Available at <https://about.gitlab.com/2017/11/09/gitlab-vue-one-year-later/>. Accessed 15 February 2018.

Shank, Patti. 2009. Usability Issues That Impact Online Learning| Higher Ed Teaching & Learning. *Faculty Focus | Higher Ed Teaching & Learning*. Available at <https://www.facultyfocus.com/articles/distance-learning/usability-issues-that-impact-online-learning/>. Accessed 16 February 2018.

Shneiderman, Ben. 1987. Designing the User Interface Strategies for Effective Human-computer Interaction. *SIGBIO Newsl.* 9 (1): 6–.

Sidorenko, Vladimir. 2017. 7 Best Frameworks For Web Development in 2017. Available at <https://gearheart.io/blog/7-best-frameworks-for-web-development-in-2017/>. Accessed 14 February 2018.

Sinha, Shantanu. 2012. Motivating Students and the Gamification of Learning. *Huffington Post*. Available at <https://www.huffingtonpost.com/shantanu-sinha/motivating-students-and-t_b_1275441.html>. Accessed 19 February 2018.

Soueidan, Sara. 2013. Lessons from the "Seductive Interaction Design" Book. Available at <https://sarasoueidan.com>. Accessed 17 February 2018.

Staubitz, Thomas, Christian Willems, Christiane Hagedorn, and Christoph Meinel. 2017. The gamification of a MOOC platform. In *2017 IEEE Global Engineering Education Conference (EDUCON)*, 883–892.

Stifterverband. 2012. Jörn Loviscach: E-Learning - Chancen und Grenzen - YouTube. Available at <https://www.youtube.com/watch?v=kWk8joW5Cug>. Accessed 17 February 2018.

Stroh, Maximilian. 2017. Web Development in 2017. *Maximilian Stroh*. Available at <https://medium.com/@Hisako1337/web-development-in-2017-e106ec18662>. Accessed 17 February 2018.

Tarchichi, Simon. n.d. Use Vue.js to create custom web components. *Vue.js Tips*. Available at <http://vuetips.com/vue-web-components>. Accessed 13 February 2018.

techsith. 2017. *Which JavaScript Framework to learn Angular, React, Vue or Ember?* Available at <https://www.youtube.com/watch?v=Q6S7m4jQO7c>. Accessed 14 February 2018.

Tolle, Troy. 2014. 8 Benefits of Gamification in eLearning | DigitalChalk Blog. *DigitalChalk*. Available at <https://www.digitalchalk.com/blog/8-benefits-gamification-elearning>. Accessed 17 February 2018.

U.S. Department of Health & Human Services. 2014. User Experience Basics. Available at <https://www.usability.gov/what-and-why/user-experience.html>. Accessed 16 February 2018.

Vue.js Guide. 2018a. Enter/Leave & List Transitions — Vue.js. Available at <https://vuejs.org/v2/guide/transitions.html>. Accessed 18 February 2018.

Vue.js Guide. 2018b. Reactivity in Depth — Vue.js. Available at <https://vuejs.org/v2/guide/reactivity.html#How-Changes-Are-Tracked>. Accessed 20 February 2018.

Vue-Router Docs. 2018. vue-router. Available at <https://router.vuejs.org/en/>. Accessed 16 February 2018.

Vuex Documentation. n.d. What is Vuex? · Vuex. Available at <https://vuex.vuejs.org/en/intro.html>. Accessed 14 February 2018.

Wong, Euphemia. 2018. User Interface Design Guidelines: 10 Rules of Thumb. *The Interaction Design Foundation*. Available at <https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb>. Accessed 17 February 2018.

You, Evan. 2016. Retiring vue-resource. *The Vue Point*. Available at <https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4>. Accessed 14 February 2018.

Yu, Jiyuan, and Zi Hu. 2016. Is online learning the future of education? Available at <https://www.weforum.org/agenda/2016/09/is-online-learning-the-future-of-education/>. Accessed 15 February 2018.