Technische Universität München
Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Kartographie
Prof. Dr.-Ing. Liqiu Meng

Cartography Playground

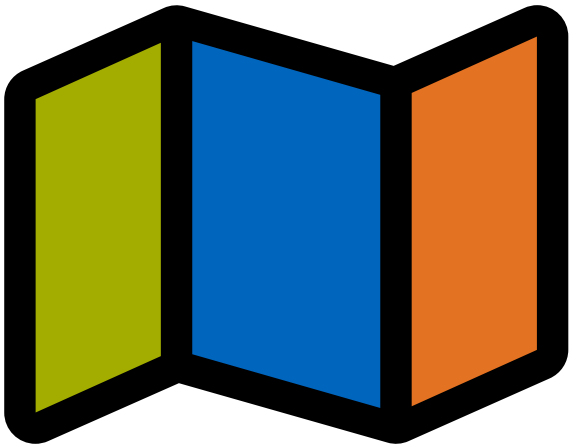
Interaktive, webbasierte Applikationen zur Erläuterung komplexer kartographischer Sachverhalte

Moritz Brunnengräber

Master Thesis

Bearbeitungszeit	01.02.2018 – 31.07.2018
Studiengang	Geodäsie und Geoinformation
Betreuer	Dr.-Ing Holger Kumke

2018



CARTOGRAPHY PLAYGROUND

Kurzfassung

Diese Masterarbeit beschreibt wie die Lehre in der Kartographie mit einer e-Learning Webseite, namens CARTOGRAPHY **PLAYGROUND** ergänzt werden kann.
Zunächst wird dazu ein Überblick zu bestehenden e-Learning Webangeboten gegeben. Im Anschluss wird auf die theoretischen Grundlagen der Seite eingegangen. Dazu zählen neben der Abgrenzung der Zielgruppe, allgemeine Erfolgsfaktoren des e-Learnings und eine kurze Beschreibung der ausgewählten Themen.
Die Webseite richtet sich vor allem an Studenten der Kartographie. Der Inhalt der Webseite umfasst Algorithmen, generelle kartographische Probleme und gestalterische Grundlagen. Ziel ist es, den Nutzern durch visuelle und interaktive Anwendungen die Inhalte näher zu bringen. Zur Implementierung wird der static-site-generator Jekyll in Kombination mit dem Webframework Bootstrap verwendet. Die einzelnen Playgrounds basieren auf Open Source `JavaScript`-Libraries. Das gesamte Projekt ist Open Source auf GitLab verfügbar, wo auch die Webseite gehostet ist, sodass sie in Zukunft auch von Anderen gepflegt und erweitert werden kann. Die Webseite kann aufgerufen werden unter
`https://moritzbru.gitlab.io/cartography-playground/`.

Abstract

This master thesis describes how the teaching in cartography can be complemented with an e-learning website called CARTOGRAPHY **PLAYGROUND**.
First, an overview of existing e-learning web offerings is given. Afterwards the theoretical basics of the site will be discussed. In addition to the demarcation of the intended target group, this also includes general success factors of e-learning and a short description of the selected topics.
The website is mainly aimed at students of cartography. The content of the website includes algorithms, general cartographic problems and design basics. The aim is to bring the content closer to the users through visual and interactive applications. For the implementation the static site generator Jekyll is used in combination with the web framework Bootstrap. The individual playgrounds are based on open source `JavaScript` libraries. The entire project is available open source on GitLab, where the website is also hosted, so that it can be maintained and expanded by others in the future. The website can be accessed at `https://moritzbru.gitlab.io/cartography-playground/`.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

A

API - Application Programming Interface ..... 23, 24, 32, V

B

BMBF - Bundesministeriums für Bildung und Forschung 1

C

CI/CD - Continuous Integration & Deployment 9, 10, 12
CMS - Content-Management-System 11
CPU - Central Processing Unit 30
CSS - Cascading Style Sheets 7, 11, 13, 17, 29

D

DBSCAN - Density-based spatial clustering of applications with noise 6, 26

E

eLML - eLesson Markup Language 3

G

GITTA - Geographic Information Technology Training Alliance ..... 3, IV

H

HTML - Hypertext Markup Language 7, 10, 11, 15–17, 20, 29, 30

J

JSON - JavaScript Object Notation 20

P

PWA - Progressive Web App 30, 31

S

Sass/SCSS - Syntactically Awesome StyleSheets/Sassy CSS 7, 11, 13, 15, 30
SGK - Schweizerische Gesellschaft für Kartografie 27
SSL/TLS - Secure Sockets Layer/Transport Layer Security 10
SVC - Swiss Virtual Campus 3
SVG - Scalable Vector Graphics 22, 29

T

TUM - Technische Universität München 8, 25

U

URL - Uniform Resource Locator 18, 32

W

1 Einleitung und Motivation

In diesem Kapitel wird die Forschungsfrage vorgestellt, welche im Rahmen der Arbeit beantwortet werden soll. Des Weiteren wird auf die aktuelle Relevanz des Themas eingegangen.

1.1 Forschungsfrage

In der Geodäsie und speziell in der Kartographie gibt es viele komplexe Themen und Algorithmen. Im Rahmen dieser Masterarbeit soll es darum gehen, diese Inhalte anschaulich darzustellen um deren Vermittlung in der Lehre zu vereinfachen und zu verbessern. Hierbei profitieren die lehrenden Dozenten und die lernenden Studenten gleichermaßen von der interaktiven, visuellen Unterstützung zum Durchdringen der komplexen Sachverhalte.
Diese Masterarbeit versucht konkret folgende Frage zu beantworten:

„Wie kann die Lehre in der Kartographie
mit Hilfe moderner, digitaler Technologien
verbessert und interaktiver gestaltet werden?“

Im Rahmen diese Masterarbeit wird eine e-Learning Webseite konzipiert und umgesetzt. In Kapitel 2 wird kurz auf bereits bestehende e-Learning Angeboten eingegangen. Ab Kapitel 3, welches sich mit der Definition der Zielgruppe, der Feststellung der Faktoren erfolgreichen e-Learnings sowie der Definition der Themen befasst, werden die Grundlagen für das Projekt CARTOGRAPHY **PLAYGROUND** geschaffen. In Kapitel 4 werden dann die gestalterischen Grundlagen, in Kapitel 5 die technischen Grundlagen besprochen. Der Hauptteil der Arbeit ist jedoch die Umsetzung von CARTOGRAPHY **PLAYGROUND** als Internetseite, welche in Kapitel 6 beschrieben wird. Die Masterarbeit schließt mit einem Fazit und Ausblick in Kapitel 7 ab.

1.2 Digitale Technologien als Lernmotivation

Im Rahmen des *ZunkunftsForum* des BMBF (Bundesministeriums für Bildung und Forschung) wurden zwischen 2015 und 2017 Bürgerdialoge und Umfragen unter anderem zum Thema „Lehren, Lernen und Leben in der digitalen Welt“ (Informationsbüro ZukunftsForum 2018) durchgeführt. Die Ergebnisse beinhalten, dass „die große Mehrheit [der Befragten anerkennt], dass der Einsatz digitaler Technologien in der Bildung unabdingbar ist“ (Informationsbüro ZukunftsForum 2018) und dass „digitale Technologien die Lust am Lernen steigern können“. Abbildung 1 zeigt einen Ausschnitt der Ergebnisse dieser Umfragen. Wie auch verschiedene Umfragen in sogenannten iPad-Klassen (Eckert und Staatliche Realschule Gauting 2018) zeigen, ist die Akzeptanz und positive Annahme der digitalen

Technologien in der Altersgruppe der 14- bis 19-jährigen mit über 80 Prozent am größten. Auch in der Gruppe der 20- bis 39-jährigen geben zwei Drittel der Befragten an, mit digitalen Technologien zum Lernen motiviert zu werden. Diese Altersgruppe ist auch der Kernbereich der Zielgruppe von CARTOGRAPHY **PLAYGROUND**, welche in Kapitel 3.1 näher erläutert wird.

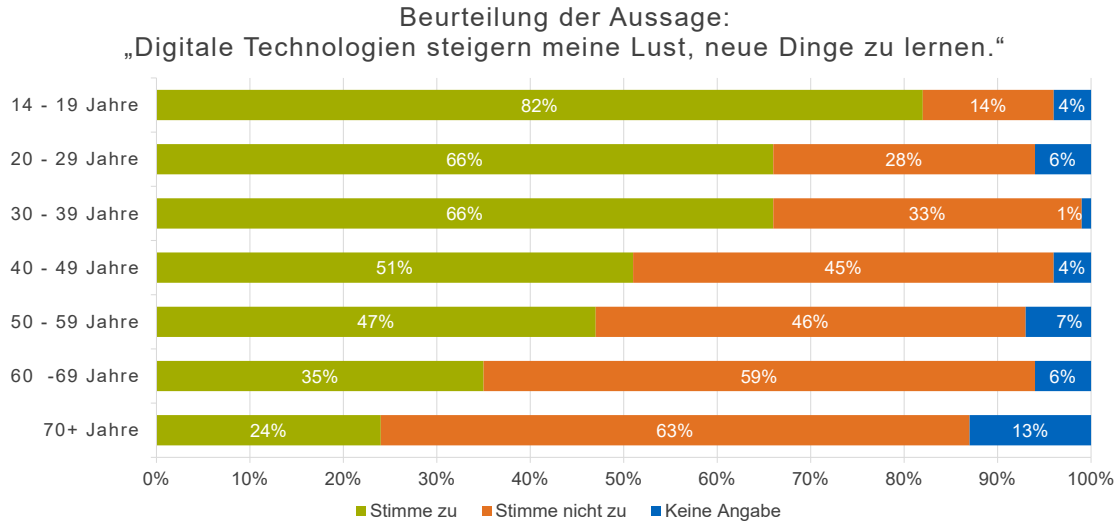


Abbildung 1 Digitale Technologien motivieren zum Lernen (nach Informationsbüro ZukunftsForum (2018))

2 Bestehende e-Learning Webangebote

In diesem Kapitel sollen kurz bereits bestehende e-Learning Angebote vorgestellt werden.
Sicherlich eine der bekanntesten und am weitesten verbreitete e-Learning Plattform ist Moodle (Moodle Pty Ltd und Open Source Community 2018). Moodle kann für die schnelle, asynchrone Kommunikation zwischen Studenten untereinander, aber auch zwischen Studenten und Lehrpersonen und zur einfachen Bereitstellung von Lehrmaterialien verwendet werden. Moodle wird überall auf der Welt von unterschiedlichsten Fachrichtungen verwendet.
GITTA (Fisler, Bleisch und Weibel 2006) und CartouCHe (Schnabel, Stopper und Hurni 2007) sind frei zugängliche eLearning-Lektionen des SVC (Swiss Virtual Campus) zu denen man sich jedoch anmelden muss. Für sie wurde eigens die Markup Sprache eLML (eLesson Markup Language) entwickelt. Es scheint aber so als würden die Projekte seit einigen Jahren nicht mehr gepflegt werden.
Auch ESRI bietet Online Kurse zum Thema Kartographie an (Environmental Systems Research Institute, Inc. (Esri) 2018). Diese sind jedoch auf die Verwendung der Produkte der ESRI ArcGIS Familie spezialisiert. Desweiteren sind diese Kurse nur nach einer Anmeldung einsehbar und teilweise kostenpflichtig.
Im Gegensatz zu den anderen e-Learning Angeboten soll CARTOGRAPHY **PLAYGROUND** kostenlose und ohne Anmeldung zugängliche, allgemeine Informationen und Playgrounds zum Thema Kartographie bereitstellen und auch einfach erweiter- und fortführbar sein.

Abbildung 2 Logos von GITTA (Fisler, Bleisch und Weibel 2006), CartouCHe (Schnabel, Stopper und Hurni 2007), und ESRI(Environmental Systems Research Institute, Inc. (Esri) 2018)

3 Grundlagen der e-Learning Webseite

In diesem Kapitel wird auf die theoretischen Grundlagen der Webseite eingegangen. Dazu zählen neben der Abgrenzung der Zielgruppe, allgemeine Erfolgsfaktoren des e-Learnings und eine kurze Beschreibung der ausgewählten Themen.

3.1 Zielgruppe des Projekts

Vor der Umsetzung der Webseite ist zunächst eine Definition der Zielgruppe notwendig, um die Seite an deren Anforderungen anzupassen. CARTOGRAPHY **PLAYGROUND** richtet sich an Personen, die Vorwissen im Bereich Geodäsie und Kartographie besitzen. Es müssen also nicht die Hintergründe der einzelnen Algorithmen und kartografischen Probleme im kleinsten Detail erläutert werden. Des weiteren wird davon ausgegangen, dass überwiegend Studenten CARTOGRAPHY **PLAYGROUND** besuchen. Sie sind „digital natives“ - also mit dem Internet und Computern aufgewachsen und kennen sich damit gut aus. Außerdem wissen Studenten woher sie zusätzliche Informationen zu den einzelnen Themen beziehen können. Wie schon erwähnt wird von technisch versierten Nutzern ausgegangen, die vermutlich moderne und aktuelle Geräte und Browser verwenden. Die unterstützten Browser sind in Kapitel 6.4 gelistet. Nicht zuletzt wird davon ausgegangen, dass die Nutzer Englisch sprechen und verstehen, da die Seite in englischer Sprache umgesetzt werden soll um die Zielgruppe zu vergrößern. Die Umsetzung von CARTOGRAPHY **PLAYGROUND** soll als Ergänzung zu klassischen Vorlesungen im Kartographie-Studium dienen. L. Sun u. a. (2003) beschreiben drei verschiedene Lernmethoden: auditiv, visuell und taktil.

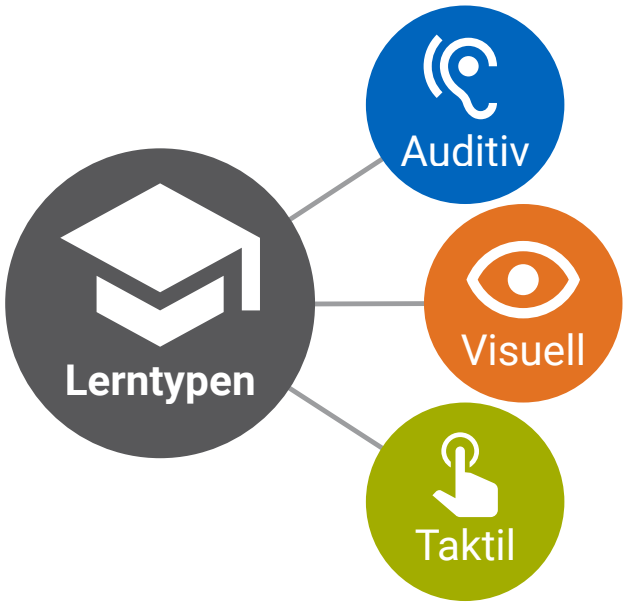


Abbildung 3 Verschiedene Lerntypen

Visuelle Lernende lernen vor allem durch beobachten des Lehrenden, das Lesen von Texten und das ansehen von Diagrammen, Animationen und Illustrationen. Auditive Lernende lernen am Besten durch das Zuhören von Vorträgen und Diskussionen. Taktile Lernende bevorzugen das aktive Experimentieren, um das Gelernte bestmöglich zu verstehen. In klassischen Vorlesungen werden vor allem auditive aber auch visuelle Lernreize geboten. CARTOGRAPHY **PLAYGROUND** setzt als Ergänzung auf visuelle und taktile Lernmöglichkeiten.

3.2 Erfolgsfaktoren des e-Learning

Laut Webster und Sudweeks (2006) ist es für Studenten unerlässlich autonomer zu werden, wobei e-Learning eine große Rolle spielen kann. E-Learning setzt im Vergleich zu traditionellem „face-to-face“ (Kahiigi u. a. 2008) Lernen, bei dem ein Lehrer die Inhalte direkt an Schüler vermittelt, auf digitale Technologien wie Computer und Internet (Kahiigi u. a. 2008). Hierbei kommen die Vorteile des asynchronen Lernens zum Tragen. Beim asynchronen Lernen müssen im Gegensatz zum synchronen Lernen die Lehrenden und Lernenden nicht gleichzeitig tätig werden (Hrastinski 2008). Beim asynchronen e-Learning können die Lehrenden also Inhalte erstellen und zum Beispiel online bereitstellen, auf welche die Lernenden wo und vor allem wann sie wollen zugreifen können. Sherry und Wilson (1997) behaupten, dass dieser Paradigmenwechsel weg von der Fokussierung auf den Lehrenden hin zur Fokussierung auf den Lernenden, einen positiven Einfluss auf Lernenden im Hochschulbereich hat, da diese nun mehr Verantwortung für ihren Lernerfolg tragen. Ausschlaggebend für den Lernerfolg ist dabei die eigenständige Auseinandersetzung der Lernenden mit den Lerninhalten.
P.-C. Sun u. a. (2008) tragen viele Informationen zusammen, die zu einem erfolgreichen e-Learning Angebot beitragen. Abbildung 4 zeigt was den Erfolg und die Zufriedenheit beim e-Learning maßgeblich beeinflussen kann.
Der Lernende mit seinen Vorkenntnissen im Computer- und Internetbereich und besonders seine Selbstdisziplin im Internet - also die Fokussierung auf den Lerninhalt ohne Ablenkungen - spielen hier eine große Rolle. Der Inhalt des e-Learning Angebots mit den behandelten Themen, deren Vielfalt und die Qualität der Inhalte haben ebenfalls Einfluss auf den Erfolg der e-Learning Webseite. Ein weiterer Erfolgsfaktor ist die Technologie - sowohl des e-Learning Angebots als auch des zugreifenden Lernenden. Hierunter fallen beispielsweise die Servergeschwindigkeit des Anbieters aber auch die Internetanbindung des Zugreifenden oder die unterstützten und verwendeten Geräte und Browser zum Zugriff. Der letzte aber nicht unwichtige Faktor ist das Design der Plattform. Es sollte einfach, zweckmäßig aber auch ästhetisch gehalten werden um die größtmögliche Zufriedenheit bei den Nutzern zu erreichen.

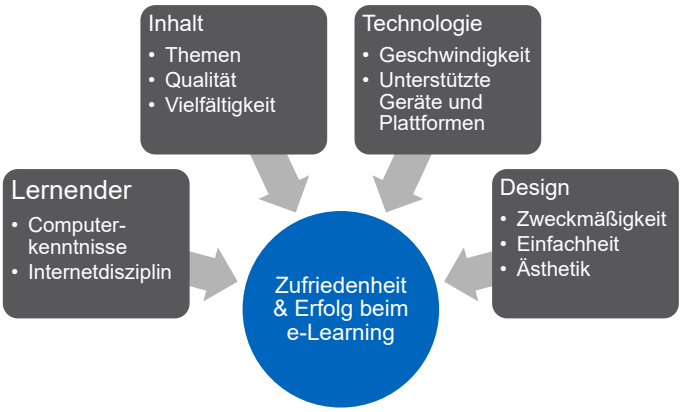


Abbildung 4 Erfolgsfaktoren e-Learning (nach P.-C. Sun u. a. (2008))

3.3 Umgesetzte Themen

Bei der Wahl der umgesetzten Themen wurde darauf geachtet, ein möglichst großes Spektrum der vielfältigen kartographischen Themen abzudecken. Die dargestellten Themen bei CARTOGRAPHY **PLAYGROUND** sollen eine Mischung aus Algorithmen, generellen kartographischen Problemen, kartographisch-gestalterischen Prozessen und einem spielerischen Wettbewerb sein.

Als Algorithmen werden der Kurvenglättungsalgorithmus Ramer-Douglas-Peucker (Douglas und Peucker 1973; Ramer 1972) und ein Vergleich der Clusteringverfahren k-Means (MacQueen 1967) und DBSCAN (Density-based spatial clustering of applications with noise) (Ester u. a. 1996) umgesetzt. Die Darstellung der generellen kartographischen Probleme beinhaltet die Interpretation von Höhenlinien und Geländeprofil, sowie Methoden der kartographischen Generalisierung. Ein „Spielplatz“ zum Kartendesign soll als Umsetzung für kartographisch-gestalterische Prozesse und ein Quiz als Umsetzung für einen spielerischen Wettbewerb dienen.

4 Gestalterische Grundlagen

In diesem Kapitel werden die gestalterischen Grundlagen vorgestellt, auf welchen die Umsetzung dieser Arbeit basiert.
Das Framework Bootstrap liefert durch Userinterface-Bausteine die Basis der Webseite. Desweiteren werden die Material Design Prinzipien und das Konzept der Farbwahl beschrieben, nach welchen die Seite gestaltet wurde.

4.1 Bootstrap

Bootstrap ist ein kostenloses Open Source Frontend-Framework zur Gestaltung von responsiven Websiten. Responsives Webdesign verfolgt das Paradigma sich auf die jeweiligen Eigenschaften, des zur Betrachtung verwendeten Endgerätes anzupassen. Hierunter fallen zum Beispiel Kriterien wie die Größe, Auflösung und Orientierung des Bildschirms und unterschiedliche Eingabemöglichkeiten mit Maus und Tastatur oder Touchgesten. Bootstrap bietet Vorlagen auf Basis von `CSS (Cascading Style Sheets)` beziehungsweise `Sass/SCSS (Syntactically Awesome StyleSheets/Sassy CSS)` und `HTML (Hypertext Markup Language)` Codefragmenten wie Navigationsleisten, Gridlayouts, Überschriften, Buttons, Tabellen und vieles mehr. Zusätzlich bietet Bootstrap optionale `JavaScript` Komponenten wie zum Beispiel Carousel Slideshows, Modal Fenster, Dropdowns, Tooltips und Scroll-basierte Aktionen (Otto u. a. 2017).
`Sass/SCSS` ist ein `CSS`-Präprozessor und erweitert `CSS` um Variablen, Schleifen, verschachtelte Regeln, Importe und vieles mehr (Catlin u. a. 2018). Bootstrap lässt sich unter anderem über diese `Sass/SCSS` Variablen sehr gut an die eigenen Designvorstellungen anpassen. Beispielsweise die primäre Akzentfarbe kann über eine Variable festgelegt werden, sodass alle Komponenten diese Farbe verwenden.

4.2 Material Design Prinzipien

Material Design ist eine von Google entwickelte Designsprache, die klassische Designprinzipien mit moderner Technologie verbinden soll. Material Design basiert auf drei grundlegenden Prinzipien (Google LLC 2017).

Material als Metapher

Inspiriert durch einfaches Design auf Papier wird versucht die Beziehung von Objekten und Oberflächen oder physikalische Erscheinungen zum Beispiel durch den Einsatz von Licht und Schatten darzustellen.

Kräftig, anschaulich, bewusst

Durch den bewussten Einsatz von Typographie, kräftigen Farben oder der Platzierung der Elemente werden deren Hierarchie und Bedeutung verdeutlicht und

Interaktionsmöglichkeiten aufgezeigt.

Bewegung bringt Bedeutungen

Die von Benutzerinteraktion ausgelöste Bewegung transformiert Objekte und ordnet sie neu an ohne die Kontinuität des Designs zu zerstören um Auswirkungen der Handlung aufzuzeigen und den Fokus des Nutzers zu lenken.

Für diese Arbeit wurden diese Prinzipien berücksichtigt. Zum Beispiel wurde das Prinzip der *Cards* für die Darstellung der Playouground-Übersicht und der Interaktiven Teile der Playgrounds eingesetzt und mit einem Schatten versehen. Vor allem aber wurden die von Google, Austin Andrews und der Open Source Community entworfenen Material Design Icons (Andrews, Google LLC und Open Source Community 2018) verwendet. In Abbildung 5 werden beispielhaft einige dieser Symbole gezeigt. Die Symbole sind bewusst schlicht gewählt, da sie dadurch sehr anschaulich sind. Die Smybole werden als Ergänzung zu Text zum Beispiel auf Buttons verwendet, um die Webseite noch intuitiver und visuell anschaulicher zu gestalten.

Abbildung 5 Beispiel von Material Design Icons (nach Andrews, Google LLC und Open Source Community (2018))

4.3 Farbwahl

Die TUM (Technische Universität München) unterhält ebenfalls eine Styleguide für die Umsetzung ihres Corporate Designs auf Webseiten (Technische Universität München 2018). Hierin werden unter anderem Empfehlungen zum Aufbau der Seite, der Typografie, dem Seitenverhältnis von Bildern und dem Aussehen von Tabellen und Formularen gegeben. Insbesondere für diese Arbeit wurden die Empfehlungen für die Farbgestaltung, wie in Abbildung 6 übernommen.

Abbildung 6 Verwendete Farben angelehnt an das TUM Corporate Design

Texte werden nicht komplett schwarz sondern dunkelgrau dargestellt, da dieser weichere Kontrast auf weißem Hintergrund angenehmer für die Augen zu lesen ist und trotzdem noch eine AAA Kontrast-Bewertung der WCAG (Web Content Accessibility Guidelines) erhält. Die Akzentfarben blau, grün und orange ergänzen sich nahezu zu einer Farbtriade - drei Farben die im Farbkreis jeweils möglichst weit entfernt voneinander sind - was besonders ausgewogen und harmonisches aber dennoch dynamisch wirkt.

5 Technische Grundlagen

Im folgenden Kapitel werden die technischen Grundlagen, die zur Realisierung von CARTOGRAPHY **PLAYGROUND** beigetragen haben, erläutert.
Zunächst wird Git vorgestellt welches in Form vom Onlineservice GitLab zur Code und Dateiverwaltung verwendet wurde. Auch andere Features von GitLab werden beschrieben. Schließlich wird der static-site-generator Jekyll, welcher zur Implementierung der Webseite verwendet wurde besprochen.

5.1 Code- und Dateiverwaltung mit Git

Der Quellcode und die binären Dateien (z.B. Raster-Bilder), die zur Erstellung der Webseite verwendet werden, werden mit Hilfe von Git (Torvalds u. a. 2018) verwaltet. Git ist eine freie und quelloffene Software zur Versionsverwaltung von - vorwiegend textbasierten - Dateien. Das Git-Projekt wurde 2005 von Linus Torvalds initiiert. Mit Git lassen sich beispielsweise mehrere parallele Entwicklungszweige, genannt `branches` (z.B. production, experimental, v1.0, v2.0, etc.) eines Softwareprojekts verwalten und einfach zusammenfügen oder aufspalten. Auch wenn Git keinen zentralen Server benötigt, da die gesamte Versionsgeschichte, genannt `history`, lokal bei jedem Nutzer gespeichert ist, bietet es sich an das gesamte Projekt, genannt `repository`, auf einem Server zu hosten und mit diesem zu synchronisieren. Dies bietet mehrere Vorteile. Zum Einen besitzt man dann ein Backup des Projekts, zum Anderen können dann mehrere Personen auf das Projekt zugreifen und an diesem arbeiten. Für diese kollaborative Versionsverwaltung gibt es einige bekannte Anbieter wie GitHub (GitHub, Inc. 2018) oder GitLab (GitLab, Inc. 2018).

5.2 Online Versionsverwaltung mit GitLab

Für diese Arbeit wurde das online Versionsverwaltungstool GitLab verwendet. Ein solches online Git bietet neben den in Kapitel 5.1 genannten Vorteilen auch weitere Funktionen wie beispielsweise die direkte Ausführung von Quellcode und das Hosten von Webseiten. Gegenüber Konkurrenten wie GitHub die bei kostenloser Nutzung nur öffentlich einsehbare `repositories` erlauben, bietet GitLab auch die Möglichkeit kostenlos private `repositories` anzulegen. Ebenso bietet GitLab CI/CD (Continuous Integration & Deployment) die Möglichkeit der automatisierten Code-Ausführung und Bereitstellung der Ergebnisse.

In Abbildung 7 ist ein schematischer Ablauf der CI/CD pipeline gezeigt.
Beispielsweise kann durch einen `commit`, also das Hinzufügen von Änderungen zu einem bestehenden `repository` der `build` Prozess eines static-site-generators wie Jekyll (siehe Kapitel 5.3) gestartet werden. Nach dessen Abschluss kann wiederum der `deploy` Prozess

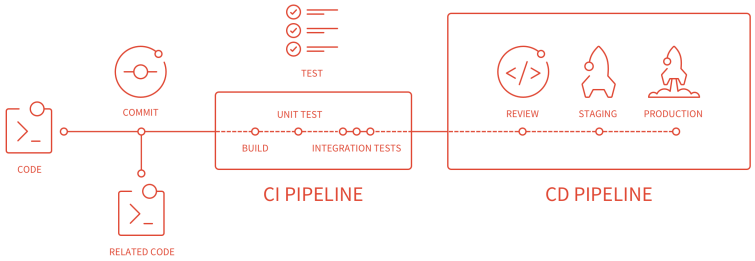


Abbildung 7 GitLab CI/CD flow (GitLab, Inc. 2018)

starten, der die Ergebnisse des `build` Prozess nun beispielsweise zum Hosting auf einem GitLab Pages Server bereitstellen kann. Was genau der CI/CD-process durchführt ist in der `.gitlab-ci.yml`-Datei festgelegt. Wie der Inhalt einer solchen Datei aussehen kann ist in Kapitel 6.1 gezeigt.
GitLab bietet mit GitLab Pages auch die Möglichkeit kostenlos statische Webseiten aus einem `repository` zu hosten. Hierbei steht eine `.gitlab.io`-Subdomain mit SSL/TLS (Secure Sockets Layer/Transport Layer Security) Zertifikat zur Verfügung. Es lassen sich aber auch eigene Domains und Zertifikate verwenden. Serverseitige Programmiersprachen wie `PHP` werden nicht unterstützt.
Um den Einstieg zu erleichtern bietet GitLab schon Beispielprojekte für gängige static-site-generators wie Jekyll, Middleman oder Hugo, die für die Verwendung von GitLab CI/CD und GitLab Pages vorkonfiguriert sind.

5.3 Statische Webseitengenerierung mit Jekyll

Für diese Arbeit wurde auf Jekyll, den bekanntesten und am weitesten verbreiteten static-site-generator mit über 34.000 GitHub stars gesetzt. Es gibt aber eine ganze Reihe an ähnlichen Projekten wie Hugo (25.000 GitHub stars), Gatsby (21.000 GitHub stars) oder Nuxt (12.000 GitHub stars). Die Philosophie von Jekyll ist im `README` des Jekyll GitHub Repositories wie folgt beschrieben:
"Jekyll does what you tell it to do — no more, no less. It doesn't try to outsmart users by making bold assumptions, nor does it burden them with needless complexity and configuration. Put simply, Jekyll gets out of your way and allows you to concentrate on what truly matters: your content." (Jekyll und Open Source Community 2018)
"Jekyll tut, was man ihm sagt - nicht mehr und nicht weniger. Es versucht weder die Benutzer durch gewagte Annahmen zu verwirren, noch belastet es sie mit unnötiger Komplexität und Konfiguration. Einfach ausgedrückt, Jekyll geht einem aus dem Weg und erlaubt es, sich auf das zu konzentrieren, was wirklich wichtig ist: der Inhalt."
Jekyll ist ein kostenloser Open Source static-site-generator, der auf der Programmiersprache `Ruby` basiert. Static-site Generatoren vereinfachen das Erstellen von statischen Webseiten da sie aus „einfachen“ Textdateien zum Beispiel im `markdown`-Format `HTML`-Seiten

generieren. Hierzu werden ein oder mehrere sogenannte *layouts*, also Vorlagen für Inhalt und dessen Anordnung verwendet. Diese *layouts* können beispielsweise mit Frontend-Frameworks wie Bootstrap (siehe Kapitel 4.1) erstellt werden. Außerdem bietet Jekyll einige erweiterte Programmierstrukturen die pures `HTML` nicht bietet, wie beispielsweise `for`-Schleifen oder `if`-Abfragen.
Jekyll verwendet die Markup-Sprache Liquid (Shopify und Open Source Community 2018). Liquid kann in *objects*, *filters* und *tags* kategorisiert werden. *Objects* werden in doppelt geschweiften Klammern geschrieben und geben Inhalt als Text zurück. *Filters* werden an *objects* durch einen senkrechten Strich getrennt angehängt und verändern deren Ausgabe. *Tags* werden in geschweiften Klammern mit Prozentzeichen geschrieben und erstellen Logik- und Kontrollblöcke wie zum Beispiel `if`-Abfragen oder `for`-Schleifen.

```
<!-- object -->
{{ page.title }}
<!-- Output: Cartography Playground -->

<!-- filter -->
{{ page.title | upcase | prepend: "Hello " }}
<!-- Output: Hello CARTOGRAPHY PLAYGROUND -->

<!-- tag -->
{% for playground in site.playgrounds %}
  <h1>{{ playground.title }}</h1>
  <p>{{ playground.description }}</p>
{% endfor %}
<!-- Loop through all playgrounds and print their title as heading
↪  and description as paragraph -->
```

Quellcode 1: Beispiel für Jekylls Markup-Sprache Liquid

Um eine Webseite zu erstellen muss Jekyll den auf `HTML`, `Liquid` und `Sass/SCSS` basierenden Quellcode erst prozessieren und in `HTML` und `CSS` Dateien umwandeln welche dann auf einen Server transferiert werden können. Im Gegensatz zu dynamischen CMSs (Content-Management-Systems) wie WordPress (WordPress Foundation, Mullenweg und Open Source Community 2018) bei denen eine Datenbank und serverseitige Sprachen wie `PHP` benötigt werden, wird bei Jekyll lediglich ein einfacher Server benötigt der statische `HTML`-Dokumente ausliefert, dessen Einrichtung weniger komplex ist. Da keine Datenbankabfragen und serverseitigen Prozesse beim Aufruf einer Webseite stattfinden, sind statische Seiten schneller und sicherer als dynamische Seiten.
Jedoch bringen static-site-generators auch einige Nachteile im Vergleich zu klassischen CMSs wie beispielsweise die komplexere erste Einrichtung und Einarbeitung, das Nichtvorhandensein einer zentralen Administrationsoberfläche mit WYSIWYG (“What You See Is What You Get")-Editor sowie das Fehlen einer Datenbank mit deren Möglichkeiten zum Suchen und Sortieren von Inhalten.

6 Umsetzung

Dieses Kapitel beschreibt die Umsetzung von CARTOGRAPHY **PLAYGROUND**. Zunächst wird darauf eingegangen, wie das Projekt auf GitLab angelegt wird. Anschließend wird die gestalterische und inhaltliche Umsetzung erläutert. Hierbei wird jeder erstellte Playground einzeln beschrieben. Abschließend werden die an der Seite durchgeführten Tests und Optimierungen dargelegt.

6.1 Anlegen des Projektes

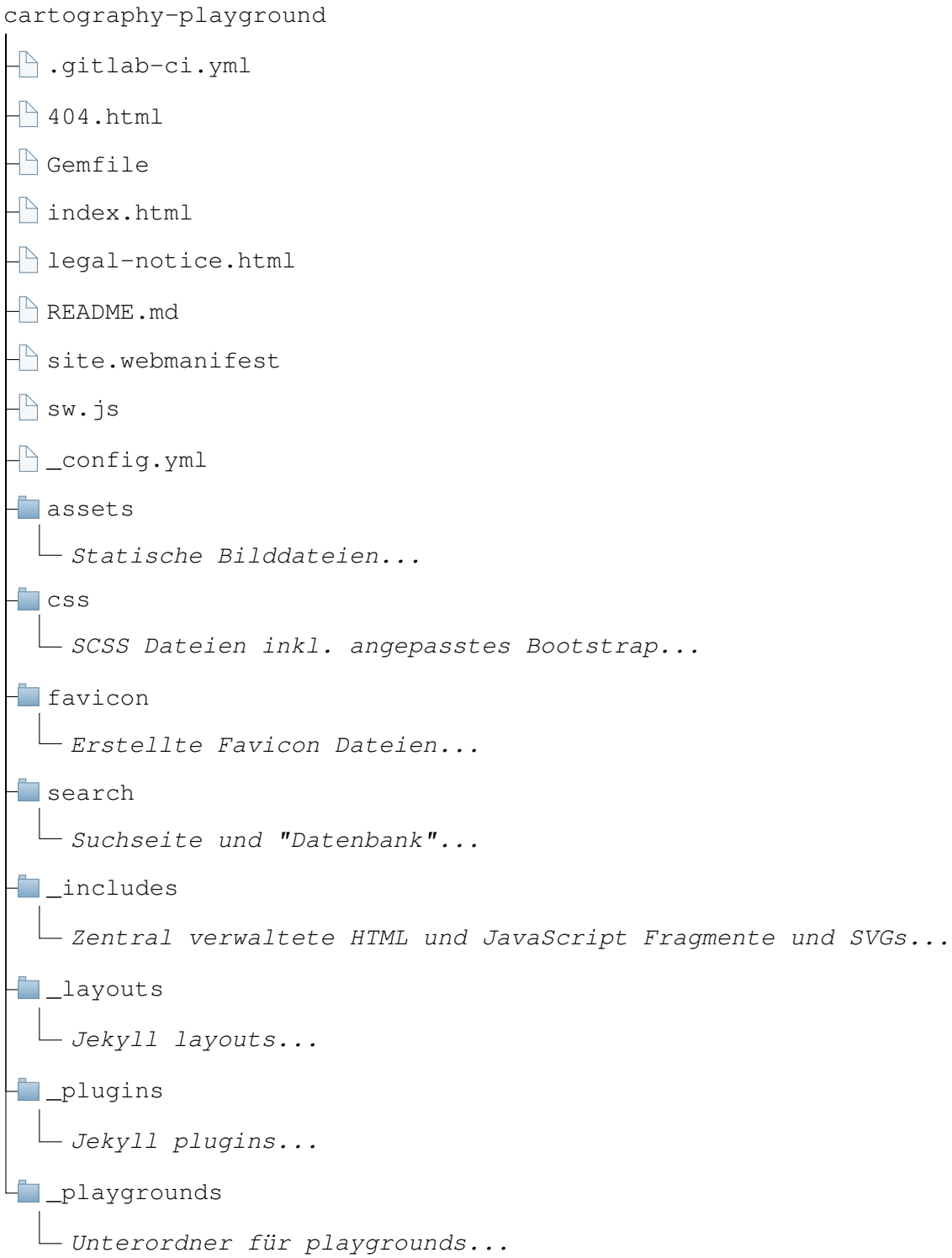
Zur Erstellung und Bereitstellung der Webseite wurde ein Git `repository` in GitLab angelegt. Es wurde das Jekyll Beispielprojekt für GitLab Pages `geforkt` um die Einrichtung von GitLab Pages und Jekyll zu erleichtern. Die CI/CD Pipeline wurde in der `.gitlab-ci.yml`-Datei wie in Quellcode 2 konfiguriert.

```
1  image: ruby:2.4
2
3  variables:
4    JEKYLL_ENV: production
5
6  before_script:
7    - export LC_ALL="C.UTF-8"
8    - export LANG="en_US.UTF-8"
9    - export LANGUAGE="en_US.UTF-8"
10   - curl -sL https://deb.nodesource.com/setup_8.x | bash
11   - apt-get install -y nodejs
12   - bundle install
13
14 pages:
15   stage: deploy
16   script:
17   - bundle exec jekyll build -d public
18   artifacts:
19     paths:
20     - public
21   only:
22   - master
```

Quellcode 2: Beispiel für eine `.gitlab-ci.yml`-Datei

Zeile 1 gibt an welches Docker Image zur Ausführung des gesamten Quellcodes verwendet werden soll. Hier wird ein Linux System mit vorinstalliertem `Ruby` in Version 2.4 geladen. Die Zeilen 6 bis 12 listen Befehle die immer vor der eigentlichen Quellcodeausführung ausgeführt werden. Zeilen 7 bis 9 setzen die Sprache und das Encoding fest. Zeilen 10

und 11 downloaden und installieren Node.js auf dem `Ruby`-Image und Zeile 12 installiert alle `Ruby` Pakete und deren Abhängigkeiten die im separaten Gem-File festgelegt sind. In diesem Fall sind im Gem-File die aktuelle *jekyll* Version, der *jekyll-autoprefixer*, welcher bei der `Sass/SCSS`-Kompilierung die nötigen Vendorprefixes vor die `CSS`-Klassen setzt, der *uglifier*, welcher für die Minifizierung von `JavaScript` Code benötigt wird und *jekyll-gzip*, welches für die Komprimierung der Datein verwendet wird, gelistet. Ab Zeile 14 folgen die Befehle für die eigentliche Codeausführung des *pages*-Jobs. In Zeile 17 wird der build-Prozess von Jekyll ausgeführt, dessen Ergebnisse im Ordner `public` landen sollen. Zeile 18 bis 20 weisen GitLab Pages an den Ordner `public` zum Darstellen der Webseite zu verwenden.
Die Ordnerstruktur des Wurzelordners des Projekts sieht wie folgt aus:



Das `Gemfile` enthält `RubyGems`, also die `Ruby` Pakete die für das Projekt benötigt werden. Diese werden beim `build` Prozess automatisch mit Hilfe von *Bundler* - ein `RubyGem` Paketmanager - installiert.
Eine weitere wichtige Datei ist die `_config.yml`. In dieser Datei wird Jekyll konfiguriert und globale Variablen wie zum Beispiel der Seitenname werden hierin festgelegt. Außerdem können in ihr die Parameter einiger Jekyll Plugins gesetzt werden.

6.2 Gestalterische Umsetzung

Zur Gestaltung der Seite wurde das Frontend-Framework Bootstrap verwendet. Es werden die `Sass/SCSS`-Files von Bootstrap anhand der Gestaltungsgrundlagen aus Kapitel 4 angepasst und um eigene `Sass/SCSS`-Dateien erweitert. Beispielsweise werden die Schriftarten *Roboto* und *Roboto Mono* geladen und die Farb-Variablen anhand der Farben wie in Abbildung 6 gesetzt. Hinzugefügt werden unter anderem Styles für den animierten *Hamburger*-Menübutton in der Navigationsleiste oder die animierte 404-Error Seite.
Des Weiteren wurde ein Logo erstellt, welches in Abbildung 8 dargestellt ist.

Abbildung 8 Cartography Playground Logo

Als Basis für das Logo dient das Material Design Karten Symbol und die Farben aus Abbildung 6. Die Logo Schriftarten sind *Roboto Condensed* und *Roboto Condensed Bold* wobei das „O“ in Playground als Kreis mit einer Kompassnadel dargestellt ist. Das Kerning - also der Buchstabenabstand - wurde so angepasst, dass „Cartography“ und „Playground“ die Gleiche Breite erhalten.
Als Favicon, also als Symbol das zum Beispiel auf Browsertabs oder -lesezeichen angezeigt wird, dient das Kartensymbol ohne die Schrift.
Auf Basis von Bootstrap wurden für Jekyll ein *default layout*, welches als allgemeines Standardlayout für alle Seite verwendet wird und ein *playground layout* für die Seiten der Anwendungen erstellt. In diese *layouts* wurden verschieden *includes* importiert. Beispielsweise gibt es ein *head include* das den `HTML-head` Teil enthält. Der Vorteil eines *includes* ist es, dass man es nur einmal zentral anlegt und in die entsprechenden Seite dann importieren kann. Dadurch wird redundanter Code vermieden und die Wart- und Anpassbarkeit der Seite erhöht. Möchte man beispielsweise etwas im `head` anpassen und hat dafür ein *include*, so muss man es nicht in zahlreichen Unterseiten anpassen sondern nur einmal zentral im entsprechenden *include*. Diese *includes* können wiederum dynamische Variablen der Seiten, in denen sie importiert sind enthalten. Codebeispiel 3 zeigt den `head` *include* wobei der `title-tag` den jeweiligen Title der Unterseite sofern er existiert und den Titel der gesamten Seite als Variablen enthält. Für den Playground „Douglas-Peucker-Algorithmus“ würde der Titel in folgender Form dargestellt werden: „Douglas-Peucker-Algorithmus | Cartography Playground“. Es gibt auch noch weitere *includes* für alle Elemente die mehrfach in verschiedenen oder allen Unterseiten von CARTOGRAPHY **PLAYGROUND** auftreten wie den Navigationsbereich, den Footer und verschiede `JavaScript` Bibliotheken.

```
1  <head>
2
3    <meta charset="utf-8">
4
5    <!-- Title -->
6    <title>
7      {% if page.title %}
8        {{ page.title | escape }} |
9      {% endif %}
10     {{ site.title | escape }}
11   </title>
12
13   <!-- Favicon -->
14   <link rel="icon" type="image/png" sizes="32x32" href="{{
     ↪  '/favicon/favicon-32x32.png' | absolute_url }}">
15   <link rel="icon" type="image/png" sizes="16x16" href="{{
     ↪  '/favicon/favicon-16x16.png' | absolute_url }}">
16   <link rel="shortcut icon" href="{{ '/favicon/favicon.ico' |
     ↪  absolute_url }}">
17   <!-- ... -->
18
19   <!-- CSS -->
20   <link rel="stylesheet" href="{{ '/css/main.css' | absolute_url }}">
21
22   <!-- ... -->
23   <!-- ... -->
24
25 </head>
```

Quellcode 3: *head include* als Beispiel für eine Jekyll *include* `html`-Datei

Quellcode 4 zeigt exemplarisch das *default layout*. In den Zeilen 4, 8, 14 und 15 sieht man wie die *includes head* für den `HTML-head` Tag, *nav* für die Navigationsleiste, *footer* für den Footer der Seite sowie *js* für die JavaScripte, die immer am Ende der Seite importiert werden sollten, um den Aufbau der Seite nicht zu behindern, importiert werden. Der eigentliche Inhalt der in das Layout eingefügt wird, erscheint innerhalb des semantischen `main` Tags in Zeile 11.

```
1  <!DOCTYPE html>
2  <html>
3
4    {% include head.html %}
5
6    <body>
7
8      {% include nav.html %}
9
10     <main role="main">
11       {{ content }}
12     </main>
13
14     {% include footer.html %}
15     {% include js.html %}
16
17   </body>
18
19 </html>
```

Quellcode 4: *default layout* als Beispiel für eine Jekyll *layout* `html`-Datei)

Es wurde darauf geachtet die Seite möglichst responsiv umzusetzen. Responsive Webdesign ist ein Ansatz der darauf Wert legt, dass Inhalte sich an das jeweilige Betrachtungsgerät, insbesondere an Bildschirmgröße, -auflösung und -orientierung anpassen. Dies wird unterstützt durch die Verwendung von Bootstrap, welches eine Vielzahl von `CSS`-Klassen für ein responsives Design bietet. Hervorzuheben ist ein zwölfgeteiltes Gridsystem. In Zeile 12 von Quellcode 5 sind die Klassen `col-lg-4` und `col-md-6` zu sehen, welche in Abhängigkeit der Bildschirmgröße die Anzahl der Spalten regeln. Bei großen (*large/lg*) Bildschirmgrößen werden $\frac{12}{4} = 3$ Spalten, bei mittleren (*medium/md*) Größen werden $\frac{12}{6} = 2$ und Standardmäßig wird nur eine Spalte von Playground-Auswahlcards auf der Startseite angezeigt.

6.3 Umsetzung der Inhalte

Die technische Umsetzung der Seite erfolgt auf Basis des erstellten *default layouts*. Es wurden die `HTML`-Seiten `index`, `404`, und `legal-notice` angelegt.
`legal-notice` enthält notwendige rechtliche Hinweise zum Beispiel zur Erfassung von persönlichen Daten.
`404` ist eine Seite die angezeigt wird wenn der `http`-Antwortcode 404 beim Versuch eines Seitenaufrufs zurückgegeben wird. `404` ist einer der häufigsten Fehlerantwortcodes im Internet. Die `http`-Antwortcodes sind dreistellige Ziffernkombinationen die in 5 Kategorien unterteilt werden können welche an der ersten Ziffer erkennbar sind (Internet Engineering Task Force (IETF) 2014).

- **1xx** (Informational): Die Anfrage ist eingegangen und wird verarbeitet
- **2xx** (Successful): Die Anfrage wurde erfolgreich empfangen, verstanden und akzeptiert
- **3xx** (Redirection): Weitere Maßnahmen müssen ergriffen werden, um die Anfrage zu bearbeiten
- **4xx** (Client Error): Die Anfrage enthält eine fehlerhafte Syntax oder kann nicht erfüllt werden
- **5xx** (Server Error): Der Server kann eine scheinbar gültige Anfrage nicht erfüllen

Der Code `404` „Not Found“ besagt, dass die angefragte Seite zum Beispiel auf Grund eines kaputten oder toten Links oder eines Schreibfehlers in der Adressleiste nicht gefunden werden konnte. Die 404-Fehlerseite stellt groß den Fehlercode dar, wobei die Null als Kompass mit Kompassnadel dargestellt ist. Diese Nadel ist animiert und spielt als Anlehnung an den nicht zu findenden Inhalt der der Seite verrückt.
Die `index`-Seite ist die Startseite, die beim Aufruf der CARTOGRAPHY **PLAYGROUND** URL (Uniform Resource Locator) automatisch angezeigt wird. Sie stellt eine Übersicht über alle verfügbaren Playgrounds dar. Das besondere hieran ist, dass die Vorzüge von Jekyll - wie beispielsweise Schleifen - voll zum tragen kommen.

```
1  ---
2  layout: default
3  ---
4
5  <!-- Other stuff like introduction ... -->
6
7  <section>
8    <div class="container">
9      <div class="row">
10
11       {% for playground in site.playgrounds %}
12         <div class="col-lg-4 col-md-6 d-flex align-items-stretch">
13           <div class="card bg-info flex-fill shadow my-3">
14               <img class="card-img-top bg-white" src="{{ 'card.svg' |
                 ↪  prepend: playground.url | absolute_url }}" alt="{{
                 ↪  playground.title }}">
15             <div class="card-body">
16               <h4 class="card-title">{{ playground.title }}</h4>
17               <p class="card-text">{{ playground.description }}</p>
18             </div>
19             <div class="card-footer border-top-0 bg-transparent">
20               <a href="{{ playground.url | absolute_url }}"
                 ↪  class="btn btn-outline-primary btn-block
                 ↪  text-uppercase">{{ playground.title }}</a>
21             </div>
22           </div>
23         </div>
24       {% endfor %}
25
26     </div>
27   </div>
28 </section>
```

Quellcode 5: Ausschnitt der `index` `html`-Datei

Zeilen 1 bis 3 sind das sogenannte *Frontmatter* von Jekyll. Hierin wird Jekyll angewiesen das *default layout* zu verwenden. In den Zeilen 7 bis 30 werden die einzelnen Playgrounds dargestellt wobei das besondere ist, dass nicht jeder Playground einzeln hinzugefügt wird, sondern alles in einer Schleife geschieht. Zeile 11 besagt, dass die Schleife alle Playgrounds der Seite durchläuft. Für den jeweils aktuellen Playground wird eine `card` angelegt, die zum Playground mit der `playground.url` verlinkt. Neben dem Bild in Zeile 14 werden auf der `card` auch der `playground.title` in Zeile 16 und die `playground.description` in Zeile 17 angezeigt, die im *Frontmatter* des jeweiligen Playgrounds angelegt werden. So muss im Ordner für die Playgrounds nur einmal eine neue Playground-Datei angelegt und mit korrektem *Frontmatter*, wie in Quellcode 6, versehen werden und der Playground taucht automatisch auf der Startseite auf.

Außerdem wurde eine `search` Seite erstellt, die es erlaubt nach den vorhandenen

Playgrounds zu suchen. Zur Realisierung der Suchfunktion wurde die `JavaScript` Programmbibliothek *Simple-Jekyll-Search* (Fei 2018) verwendet. Hierzu wurde eine JSON (JavaScript Object Notation) Datei erstellt, in dem alle für die Suche relevanten Informationen gespeichert werden. Dies geschieht ähnlich wie in Codebeispiel 5 in dem mit einer Schleife über alle Playgrounds die Informationen `title`, `url`, `description` und `tags` ausgelesen und in das JSON-File geschrieben werden. Diese Datei kann durch Eingabe von Suchbegriffen in die Suchleiste durchsucht werden. Um die Suche zu Erleichtern wird eine sogenannte Tag-Cloud angezeigt, die die Schlagwörter der verschiedenen Playgrounds enthält. So können Anwendungen gefunden werden, selbst wenn dem Nutzer der genaue Titel oder die Beschreibung des Playgrounds nicht bekannt ist. Das Ergebnis der Suche wird als Liste mit Bild, Titel und Beschreibung der Playgrounds dargestellt und verlinkt zu den eigentlichen Playgroundseiten.

Alle Playgroundseiten sind nach einem festen Schema aufgebaut. Zuerst wird dem Nutzer in einer kurzen Einleitung ein Überblick über das Thema des Playgrounds gegeben. Anschließend wird das behandelte Thema durch textuelle Beschreibung und Grafiken näher erläutert. Den Kern jeder Seite bildet der interaktive Teil. Hier kann der Nutzer die theoretisch erlernten Inhalte im Rahmen unterschiedlicher Anwendungen selbst ausprobieren. Der Nutzer kann hier oftmals durch Veränderung einzelner Parameter direkt die Auswirkungen auf das Ergebnis beobachten. Durch diesen Aufbau soll neben dem visuellen Lernen auch das taktile Lerne (vgl. Kapitel 3.1) unterstützt werden. Am Ende jeder Seite werden Quellen und weiterführende Links angegeben. Für die Erstellung der Playgrounds wurde eine neue Collection names *playgrounds* in Jekyll angelegt. Hierzu wurde in der `_config.yml` unter `collections` der Eintrag *playgrounds* hinzugefügt und im Wurzelverzeichnis des Projekts ein Ordner names `_playgrounds` erstellt. Alles innerhalb dieses Ordner gehört zur Collection, welche dann beispielsweise in Jekyll mit for-Schleifen durchlaufen werden kann.

Jeder Playground in der Collection befindet sich aus Gründen der Übersichtlichkeit wiederum in einem eigenen Ordner dessen Namen dem `HTML`-Namen des Playgrounds entspricht.

Jedes Playground-`HTML` sollte im *Frontmatter* seinen Titel *titel*, eine kurze Beschreibung *description* und ein Veröffentlichungsdatum *date* enthalten, nach dem sortiert wird und kann optional noch Schlagwörter *tags* die bei der Suche helfen, Quellen *references* sowie Zusätze für den `HTML`-head und -footer enthalten. Quellcode 6 zeigt exemplarisch das *Frontmatter* des Douglas-Peucker Playgrounds.

```yaml
---
title: "Douglas-Peucker algorithm"
description: "Play with the algorithms parameter to reduce ..."
date: 2018-02-21
tags: douglas peucker ramer line smoothing simplification
references:
  - author: "Urs Ramer"
    title: "An iterative procedure ..."
    date: 1972
    link: https://www. ...
  - author: "David Douglas, Thomas Peucker"
    title: "Algorithms for the reduction ..."
    date: 1973
    link: https://www. ...
foot_additional: >
  {% include mathjax.html %}
  {% include svgjs.html %}
---
```

Quellcode 6: Exemplarisches *Frontmatter* des Douglas-Peucker Playgrounds

6.3.1 Douglas-Peucker Algorithmus

Für die Erläuterung des Douglas-Peucker Algorithmus (Douglas und Peucker 1973; Ramer 1972) wurden zunächst Vektorgrafiken im `SVG (Scalable Vector Graphics)` Format wie in Abbildung 9 für die einzelnen Schritte des Algorithmus in den Themenfarben erstellt. Zur Darstellung der mathematischen Formel wird die Open Source `JavaScript` Library *MathJax* (MathJax Consortium 2018) verwendet. Diese sucht beim Rendern der Seite Latexbefehle und wandelt sie in korrekt dargestellte mathematische Formeln um.

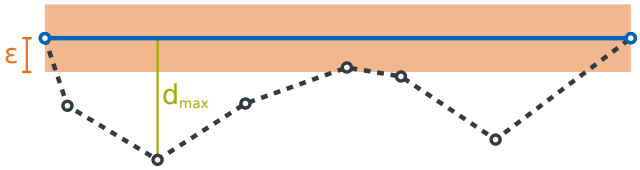


Abbildung 9 Erläuterungsbild zum Douglas-Peucker Algorithmus

Die `JavaScript` Implementierung des Douglas-Peucker Algorithmus für den interaktiven Teil der Seite basiert auf *Simplify.js* (Agafonkin 2018). Die Visualisierung wurde mit Bootstrap-Komponenten sowie der `JavaScript` Bibliothek *SVG.js* (Fierens und Open Source Community 2018) umgesetzt. Wie in Abbildung 10 dargestellt, lässt sich der Parameter ε durch einen Schieberegler verändern und somit die gezeichnete Polylinie mehr oder weniger stark vereinfachen. Es können entweder vorgefertigte Beispiel Polylinien geladen, aber auch eigene Polylinien gezeichnet werden.

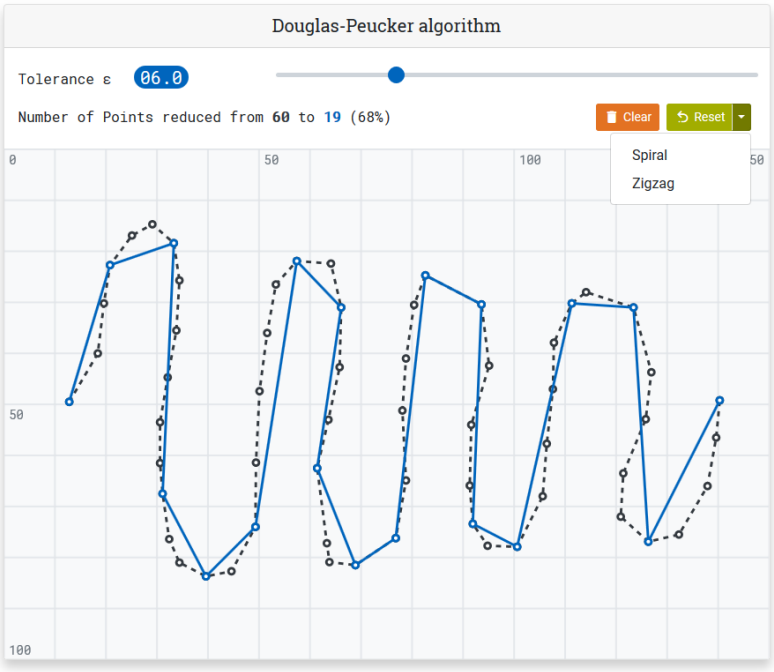


Abbildung 10 Hands-on zum Douglas-Peucker Algorithmus

6.3.2 Interpretation von Höhenlinien und Geländeprofil

Im Rahmen dieses Playgrounds wird dem Nutzer der Zusammenhang zwischen Geländesteigung und der Kartendarstellung durch Höhenlinien nähergebracht. Zur Einführung wurden dazu Vektorgrafiken in den Themenfarben erstellt. Hiermit wurden einerseits die unterschiedlichen Typen von Höhenlinien beschreiben und andererseits der Zusammenhang zwischen Geländeprofil und Höhenlinien veranschaulicht, welcher auch in Abbildung 11 dargestellt ist.

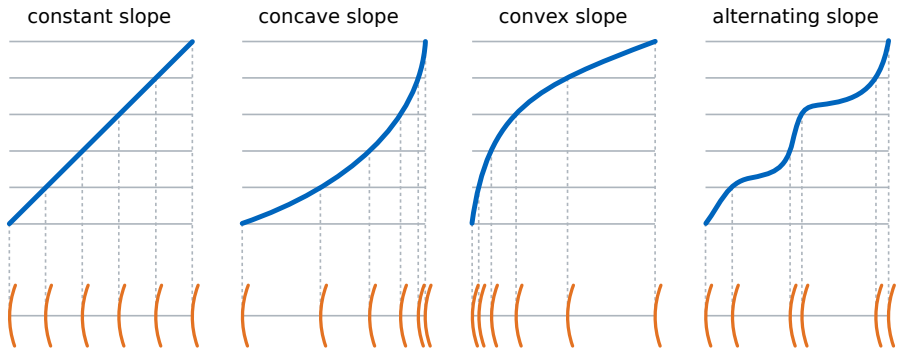


Abbildung 11 Zusammenhang zwischen Geländeprofil und Höhenlinien

Für die interaktive Kartendarstellung wie in Abbildung 12 wird das Kartenframework *Mapbox GL JS* (Mapbox Inc. und Open Source Community 2018a) verwendet. Für die kostenlose Verwendung von Mapbox wird ein API Schlüssel benötigt. Der Kartenstil mit dargestellten Höhenlinien wurde in Mapbox Studio erstellt und basiert auf dem Mapbox Style „Outdoors“. Die Profillinie lässt sich durch Klicken in die Karte erstellen, wobei bei jedem Klick WGS84 (World Geodetic System 1984)-Punkte als Breiten- und Längengrade extrahiert werden. Diese Punkte werden der Reihe nach miteinander verbunden und die Länge der resultierenden Profillinie wird mit Hilfe der Mapbox `JavaScript` Erweiterung *Turf.js* (Mapbox Inc. und Open Source Community 2018b) berechnet. Die Höhe entlang der Profillinie wird mit der *Google Maps Elevation API* (Google LLC 2018b) abgefragt, da mit Mapbox eine Höhenabfrage entlang einer Linie nur auf komplexen Umwegen möglich wäre. Auch für die Verwendung der Google API wird ein API-Schlüssel benötigt. Die API-keys besitzen in der verwendeten kostenlosen Variante ein Anfragenlimit von 50.000 Kartenaufrufen pro Monat bei Mapbox und 40.000 API-Anfragen pro Monat bei Google, was aber für eine normale Nutzung ausreichend sein sollte. Für die Visualisierung des Höhenprofils wird die Bibliothek *Chart.js* (Chart.js und Open Source Community 2018) verwendet.

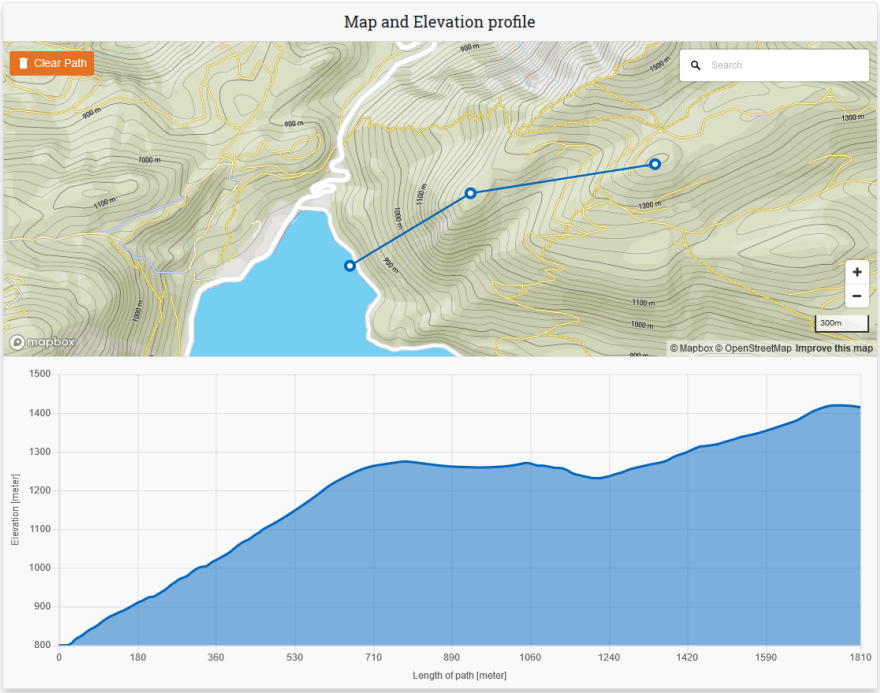


Abbildung 12 Interaktive Karte mit Höhenlinien und Geländeprofil

Die Schwierigkeit bei der Umsetzung dieses Playgrounds lag darin, geeignete APIs vor allem für die Abfrage der Höhenwerte zu finden und alle verwendeten APIs zu integrieren und miteinander zu verbinden. Die Ergebnisse der Tests der APIs von Google Maps, Bing Maps und Open-Elevation sind in Tabelle 1 dargestellt. Leider bietet Mapbox keine Höhenwertabfrage über eine API an. Es könnten lediglich die Höhenwerte anhand der Vektordaten der zugrundeliegenden Karte abgefragt werden. Diese Abfrage ist jedoch komplex in der Implementierung und vermutlich nicht sehr genau und schnell.

	Google Maps	Bing maps	Open-Elevation
Geschwindigkeit	Sehr schnell	Schnell	Sehr langsam
Genauigkeit	Gut	Akzeptabel	Schlecht
API Key	Ja	Ja	Nein
Open Source	Nein	Nein	Ja
Anfragelimit kostenlos	40 000 $^{\text{requests}}/_{\text{month}}$	125 000 $^{\text{requests}}/_{\text{year}}$	1 024 $^{\text{bytes}}/_{\text{request}}$
Stabilität	Keine Probleme	Keine Probleme	Vereinzelt Probleme

Tabelle 1 Test der Elevation APIs

6.3.3 Kartendesign

Wie schon Eduard Imhof sagte: „Die Karte ist eine künstlerische Kreation. Auch wenn sie durch wissenschaftliche Zwecke stark konditioniert ist, kann sie sich den grafischen Gesetzen nicht entziehen.“ (Imhof 2007). Zur Erläuterung des Kartendesigns wurde eine ausführliche Einleitung unter anderem auf Basis verschiedener Kartographie-Vorlesungen an der TUM sowie Hake, Grünreich und Meng (2013) angelegt und wieder mit selbst erstellten Grafiken verbildlicht. Für den interaktiven Teil, wie in Abbildung 13, kommt erneut eine Mapbox Karte in Verbindung mit Bootstrap Komponenten zum Einsatz, die durch die Farbauswahl `JavaScript` Library *Spectrum* (Grinstead und Open Source Community 2018) ergänzt wird. Die verschiedenen Schriftarten für die Karte werden von *Google Fonts* (Google LLC 2018a) geladen. Der Nutzer soll im Rahmen dieses Playgrounds entdecken, welche Auswirkungen die Veränderung von Farbe und Kartenschrift auf die Wirkung der Kartendarstellung hat.

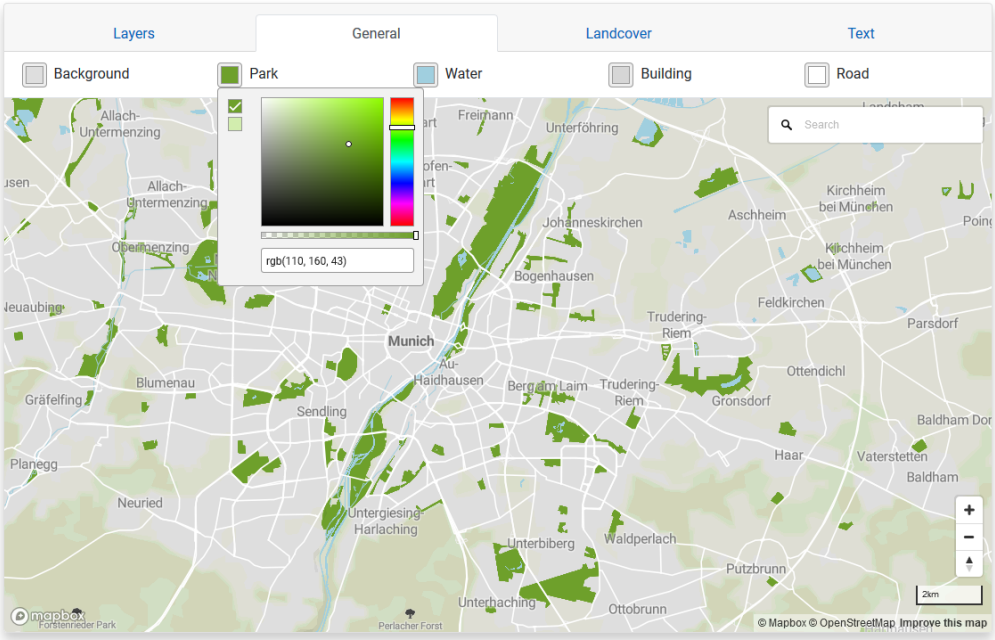


Abbildung 13 Karte mit Farbauswahl

6.3.4 Vergleich von Clusteringverfahren

Für den Vergleich des centroid-basierten Clusteringverfahrens *k-Means* (MacQueen 1967) mit dem dichte-basierten Verfahren DBSCAN (Ester u. a. 1996) wurden - wie beim Douglas-Peucker Algorithmus - Grafiken in den Themenfarben zur Erläuterung der Verfahren erstellt und deren mathematischen Grundlagen mit *MathJax* dargestellt.

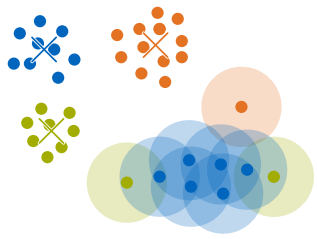


Abbildung 14 Visualisierung der Clusteringverfahren k-Means und DBSCAN

Der interaktive Teil wird aufbauend auf bestehenden Open Source `JavaScript` Implementierungen der beiden Verfahren (Młokosiewicz und Open Source Community 2018; Sugar und Open Source Community 2018), sowie mit *SVG.js* und Bootstrap Komponenten umgesetzt. Es lassen sich vorgefertigte Datensätze laden, die die Vor- und Nachteile der Verfahren hervorheben. Alternativ lassen sich auch eigene Datenverteilungen kreieren. Des weiteren kann bei k-Means die Initialisierung der Centroiden zufällig oder manuell gesetzt werden. Der Algorithmus kann in Einzelschritten oder durchgehend ausgeführt werden. Bei DBSCAN können die beiden Parameter ε und $minPts$ variiert werden. Bei beiden Verfahren kann zusätzlich das Distanzmaß zwischen Euklidischer und Manhatten Distanz gewählt werden.

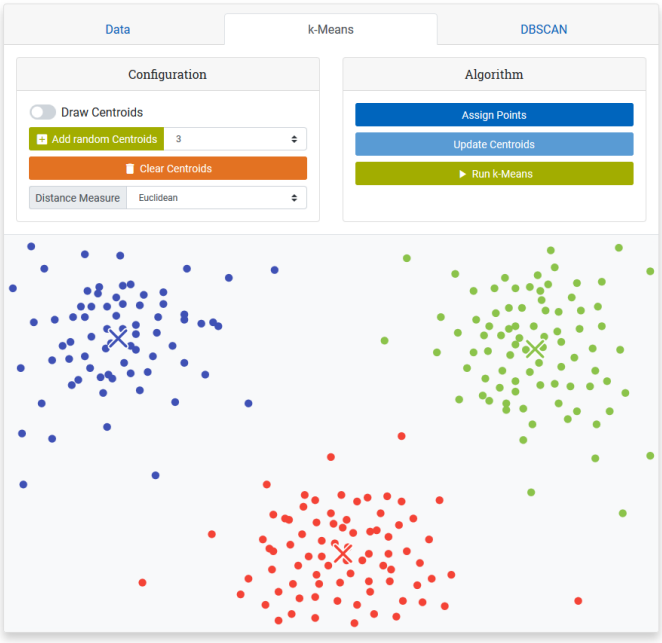


Abbildung 15 Interaktives Clusteringverfahren k-Means

6.3.5 Kartographische Generalisierung

Dieser Playground veranschaulicht dem Nutzer die Auswirkung verschiedener Generalisierungsmethoden auf die Kartendarstellung. Beim Playground zur Generalisierung wurde besonderes Augenmerk auf die Erstellung von eingängigen Symbolbildern zur Veranschaulichung ausgewählter Methoden der Generalisierung gelegt. Auch hier wurde auf den Einsatz von wenigen Themenfarben und simplen Formen geachtet. Abbildung 16 zeigt exemplarisch die Methode *Auswahl*.

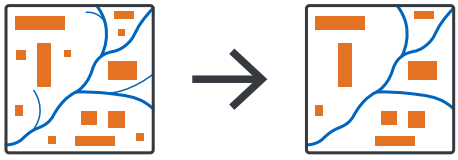


Abbildung 16 Generalisierungsmethode *Auswahl*

Inspiriert durch Publikationen der SGK (Schweizerische Gesellschaft für Kartografie) (Schweizerische Gesellschaft für Kartographie 1990) und ESRI (Environmental Systems Research Institute, Inc. (Esri) 1996) wurden die Methoden *Auswahl*, *Vereinfachung*, *Glättung*, *Aggregation*, *Kollapse der Dimension*, *Übertreibung*, *Verdrängung* und *Klassifizierung* dargestellt. Für den interaktiven Teil wurden alle Methoden zusammen in einer erstellten Grafik visualisiert und können, wie in Abbildung 17 dargestellt, einzeln an- oder abgewählt werden um die Karte zu generalisieren.

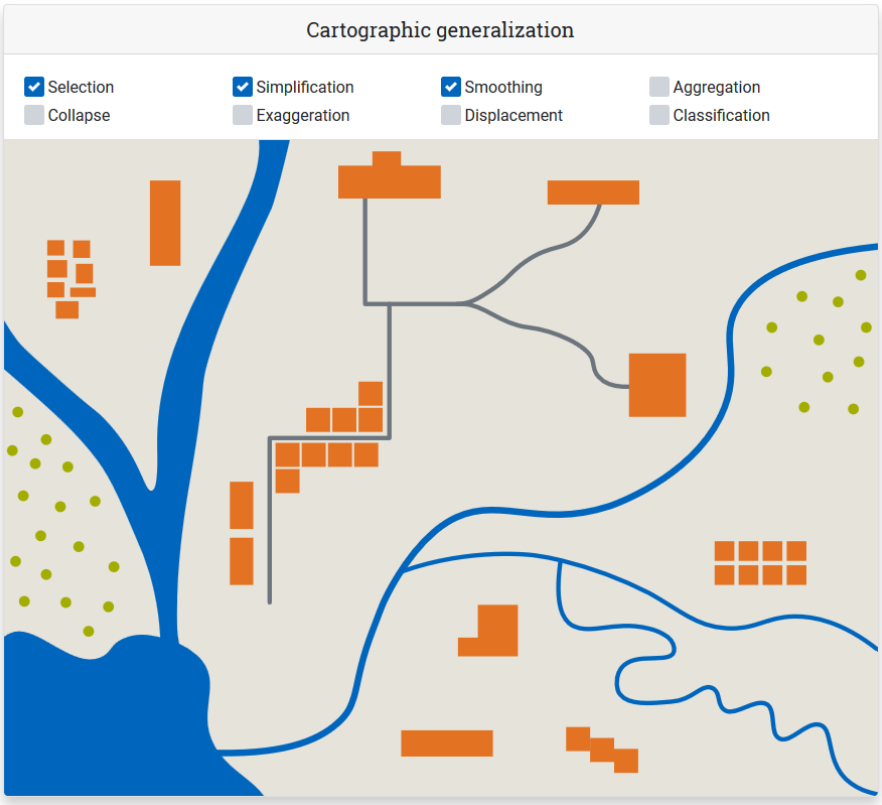


Abbildung 17 Schematische Karte zur Darstellung der Generalisierungsmethoden

6.3.6 Quiz

Mit dem Quiz kann der Nutzer sein erlerntes Wissen aus den bisherigen Playgrounds spielerisch überprüfen. Für das Quiz wurden Fragen und Aufgaben, die alle anderen Playgrounds auf der CARTOGRAPHY **PLAYGROUND** Seite betreffen angelegt. Hierfür wurde der externe Service *H5P* (Joubel und Open Source Community 2018) verwendet und in die Seite eingebettet. Es wurde ein *H5P* Quiz mit Multiple Choice Fragen, Lückentexten, Wahr/Falsch Fragen und Bildzuordnung per Drag&Drop erstellt, welches die zuvor erstellten Texte und Bilder der Einführungen in die anderen Playgrounds verwendet. Abbildung 18 zeigt die Erstellung des Quiz mit H5P.

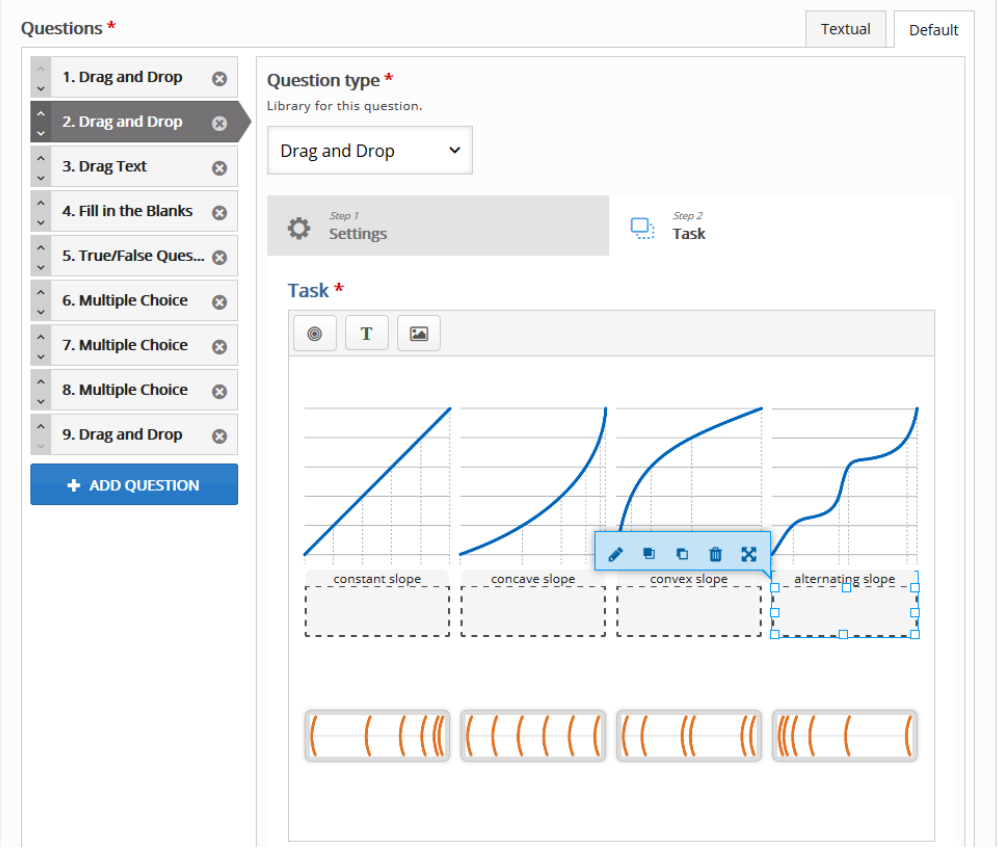


Abbildung 18 Ausschnitt der H5P Quiz-Erstellung

6.4 Test und Optimierung

Die Umsetzung der Seite wurde umfangreich getestet und optimiert. Es wurden die in Tabelle 2 gelisteten Systemen und Browser verwendet um alle Unterseiten aufzurufen und visuell auf volle Funktionsfähigkeit in der jeweils verwendeten Auflösung und Bildschirmgröße zu prüfen. Die Browser wurden aufgrund der von *StatCounter* erhobenen Statistik zur Browserverteilung in Europa (StatCounter 2018) aus Abbildung 19 ausgewählt.

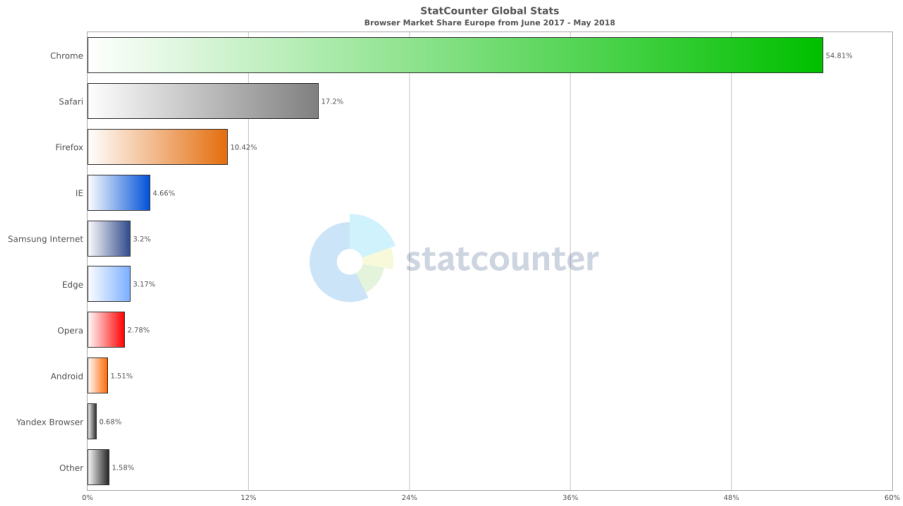


Abbildung 19 Browser Marktanteil in Europa von Juni 2017 - Mai 2018 (StatCounter 2018)

Leider standen keine *Apple* Produkte mit dem Browser *Apple Safari* zum Testen zur Verfügung. Da *Apple Safari* auf der Webkit Engine basiert und ähnlich wie *Google Chrome* und *Mozilla Firefox* nahezu alle modernen `HTML`-, `CSS`- und `SVG`-Spezifikationen des W3C (World Wide Web Consortium) umsetzt, wird davon ausgegangen, dass CARTOGRAPHY **PLAYGROUND** auf *Apple Safari* ordnungsgemäß funktioniert. Mit dem Dienst *Browsershots* (Browsershots und Open Source Community 2018) welcher Screenshots von Webseiten auf verschiedenen Betriebssystemen und Browsern bereitstellt, kann zumindest die optisch korrekte Darstellung der Seite bestätigt werden.

	Desktop Computer	**Notebook**	**Tablet**	**Smartphone**
Betriebssystem	Windows 10	Ubuntu 18.04	Android 7.1	Android 8.0
Bildschirmgröße	22 Zoll	16 Zoll	10 Zoll	5.5 Zoll
Bildschirmauflösung	1680×1050 Pixel	1366×768 Pixel	2560×1600 Pixel	1920×1080 Pixel
getestet Browser	Mozilla Firefox 60, Google Chrome 67, Opera 53, Microsoft Edge 41, Microsoft Internet Explorer 11	Mozilla Firefox 60	Mozilla Firefox 60, Google Chrome 67	Mozilla Firefox 60, Google Chrome 67, Samsung Internet

Tabelle 2 Testgeräte und Browser

Die Tests verliefen überwiegend zufriedenstellend. Lediglich die Browser *Microsoft Internet Explorer* und *Samsung Internet* haben Darstellungsprobleme oder implementieren nicht alle `HTML`-Spezifikationen und unterstützen somit nicht alle modernen Browserfunktionen. Aufgrund des geringen Marktanteils dieser Browser, der vermutlich in der Zielgruppe der technisch versierten Nutzer von CARTOGRAPHY **PLAYGROUND** noch geringer ausfällt als im europäischen Durchschnitt, wurden keine großen browserspezifischen Optimierungen durchgeführt, sondern ein Hinweis auf der Startseite von CARTOGRAPHY **PLAYGROUND** platziert, der die Nutzer anweist unterstützte Browser zu verwenden.
Um die Webseite hinsichtlich ihrer Geschwindigkeit und anderer Optimierungsmöglichkeiten zu evaluieren wurde das Open Source Tool *Lighthouse* (Google Chrome und Open Source Community 2018) verwendet, welches in die Entwicklertools des Browsers Google Chrome integriert ist. Außerdem wurden Test mit Hilfe der Internetseiten *GTmetrix* (GT.net 2018) und *Richard's Toolbox* (Oosterhof 2018) durchgeführt.
Die aus den Tools erhaltenen Tipps wurden umgesetzt, indem die von Jekyll generierten `HTML`-Seiten mit dem Jekyll Layout *compress* (Broder und Open Source Community 2018) bereinigt werden. Das heißt, es werden alle unnötigen Leerzeichen und Zeilenumbrüche und alle Kommentare entfernt. Das reduziert zum Beispiel die Größe der Datei `index.html` von 25kb auf 21kb, was einer Ersparnis von ca. 16% entspricht. Ebenso wurde in der Jekyll `config`-Datei die `Sass/SCSS`-Kompilierung auf *compressed* gestellt. Dies bringt Einsparungen um ca. 23%. `JavaScript`-Dateien werden mit Hilfe eines selbstgeschriebenen Jekyll Plugins und der Bibliothek *UgilfyJS* (Bazon und Open Source Community 2018) verkleinert. Beim `JavaScript` für die Seite des Clusteringvergleichs beispielsweise hat die Komprimierung eine Einsparung um ca. 29% zur Folge.
Anschließend werden alle Dateien noch mit Hilfe des Plugins *Jekyll::Gzip* mit dem *gzip*-Verfahren komprimiert. Da der Flaschenhals bei der Webseiten-Geschwindigkeit die Übertragung der Dateien ist und moderne Wiedergabegeräte über leistungsfähige CPUs (Central Processing Units) verfügen, unterstützen moderne Browser das Anfragen der Webseiteninhalte in komprimierten Formaten, da das Übertragen und Entpacken kleiner, komprimierter Dateien zumeist schneller als das Übertragen großer Dateien ist. Die Komprimierung bringt je nach Datei eine weitere Größeneinsparung von 60% bis 80%. Um die Kompatibilität mit Browsern zu wahren, die keine Komprimierung unterstützen liegen alle Dateien als mit *gzip* komprimierter und Original-Version auf dem Server vor.
Auch die anderen Empfehlungen des Analysetools *Lighthouse* wurden größtenteils umgesetzt. Zum Beispiel wurden zur Verbesserung der Accessibility - also dem barrierefreien Zugang zum Web - `titel` und `label` Attribute zu Bildlinks oder Buttons, die nur Symbole enthalten hinzugefügt. Außerdem wurden Fokusstile angelegt, die es Ermöglichen nur mit Tastatureingaben ohne Maus auf der Seite zu navigieren.
Auch ein `Web App Manifest` und ein `Serviceworker` wurden angelegt um mit Hilfe der `JavaScript`-Bibliothek *Workbox* (Google LLC und Open Source Community 2018) ein Caching der Seite vorzunehmen und die Seite Offline- und PWA (Progressive Web App) fähig zu machen. Mit einer PWA kann die Webseite zum Beispiel unter Android aus dem Chrome Browser als eigene App gespeichert werden und funktioniert dank der gecachten

Inhalte auch ohne Internetverbindung. Auch auf Desktop Browsern können PWAs genutzt werden, jedoch wird diese neue Technologie noch nicht vollständig von den großen Browsern unterstützt.
Nach den Optimierungen gibt *GTmetrix* einen sehr guten *Google PageSpeed Score* von A(94%) und einen ebenfalls sehr guten *Yahoo YSlow Score* von A(96%) aus. Auch das Tool *Lighthouse* gibt sehr gute Ergebnisse aus, die in Abbildung 20 begutachtet werden können.

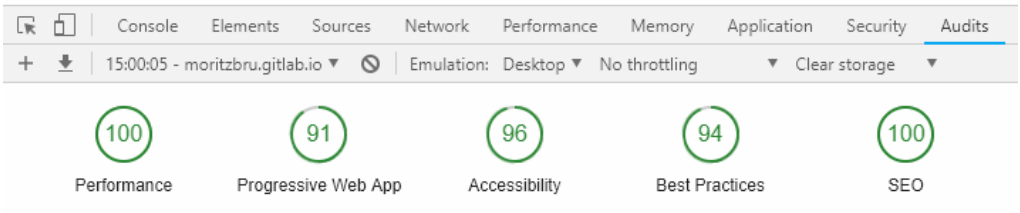


Abbildung 20 Ergebnisse des *Lighthouse* Audits

In allen verwendeten Tools finden sich auch noch weitere Tipps zur Verbesserung der Seite, die aber angesichts der schon sehr guten Ergebnisse nicht mehr berücksichtigt wurden. Hier kommt das *Pareto-Prinzip* - auch 80/20-Regel genannt - zum Tragen, was bedeutet, dass die letzten 20% der Optimierung 80% des Aufwandes ausmachen würden.

7 Fazit und Ausblick

„Wie kann die Lehre in der Kartographie mit Hilfe moderner, digitaler Technologien verbessert und interaktiver gestaltet werden?“ Diese anfängliche Frage kann nun beantwortet werden. Die Lehre in der Kartographie kann mit der e-Learning Webseite CARTOGRAPHY **PLAYGROUND** ergänzt werden. Diese ist unter der URL

`https://moritzbru.gitlab.io/cartography-playground/`

zu erreichen. Die Seite setzt auf ein schlichtes und modernes Design. Die umgesetzten Themen sind möglichst vielfältig gewählt. Es wurde darauf geachtet, dass deren Inhalte nach Möglichkeit visuell und interaktiv gestaltet sind. Auch auf einen schnellen und responsiven Seitenaufbau wurde Wert gelegt.
Wie an der URL zu erkennen ist, wird die Seite auf GitLab gehostet. Auch der Quellcode der Seite ist Open Source auf GitLab verfügbar. Für die Höhenwertabfrage des Playgrounds zur Interpretation der Höhenlinien wurden auch Closed Source APIs - Google Maps Elevation API und Bing Maps Elevations API - verwendet. Leider kommen die Open Source Alternativen in Schnelligkeit, Qualität und Benutzerfreundlichkeit nicht an die genannten Closed Source APIs heran. Es wäre jedoch wünschenswert, dass das Projekt in Zukunft nur auf Open Source Abhängigkeiten basiert.
Es gibt noch weitere Verbesserungsmöglichkeiten, vor allem für die Anzeige auf kleinen Disyplays. Zum Beispiel können zu lange Formeln nicht umgebrochen werden und müssen horizontal gescrollt werden. Es müssen zu viele horizontale Tabs bei geringem Platz übereinander dargestellt werden, was die Ästhetik etwas stört.
Auch an anderen Stellen gib es Verbesserungspotential. Beispielsweiße wurde das Quiz von H5P extern auf der H5P Webseite erstellt und dann per *iframe* in CARTOGRAPHY **PLAYGROUND** eingebunden. Es wäre besser, wäre man hier nicht auf einen externen Service angewiesen und könnte das Quiz direkt vom selben Server ausliefern. Hierzu könnte das Projekt H5P-Standalone (Tunapanda und Open Source Community 2018) verwendet werden. Der Quiz-Playground könnte zum Beispiel noch um eine Bestenliste erweitert werden.
Aufgrund von Zeitmangel konnten im Rahmen dieser Masterarbeit nur sechs Playgrounds implementiert werden. Im Idealfall sollten diese sechs Playgrounds nicht die einzigen bleiben. Es könnten weitere Playgrounds zum Beispiel zu den Themen Kernel Density Estimation (mit Hilfe der `JavaScript`-Bibliothek Plotly.js), Kartennetzprojektionen (basierend auf map-projections.net (Jung 2018)) oder dem Vergleich von verschiedenen Darstellungsmethoden der Thematischen Kartographie erstellt werden. Ebenfalls könnte ein weiteres Spiel implementiert werden, bei dem Pins möglichst nah an vorgegeben Orten oder dem Aufnahmeort eines gezeigten Fotos, auf der Karte platziert werden müssen (mit Hilfe von Mapbox (Mapbox Inc. und Open Source Community 2018a)).
Um auch auditive Lerner ansprechen zu können, könnten Audio-Files in die Seite eingebunden werden, welche den Inhalt auditiv wiedergeben. Auch das Einbinden von

Video-Tutorials ist denkbar.
Da der Quellcode Open Source auf GitLab verfügbar ist, können die genannten Verbesserungen, aber auch andere Änderungen und Erweiterungen des Projekts CARTOGRAPHY **PLAYGROUND** von Anderen talentierten und motivierten Entwicklern umgesetzt werden. Um den Einstieg und die Einarbeitung zu erleichtert, steht auf GitLab auch ein Guide zur Verfügung, der das Projekt und vor allem die Erstellung neuer Playgrounds beschreibt. Dieser Guide ist auch im Anhang der Arbeit zu finden.
Es wäre erfreulich wenn CARTOGRAPHY **PLAYGROUND** viel genutzt und selbst zu einem interaktiven Projekt mit vielen weiteren Playgrounds anderer Nutzer wird.

Literatur

Agafonkin, Vladimir (13. 06. 2018). *Simplify.js*. URL: https://github.com/mourner/simplify-js.

Andrews, Austin, Google LLC und Open Source Community (16. 03. 2018). *Material Design Icons*. URL: https://materialdesignicons.com/.

Bazon, Mihai und Open Source Community (18. 06. 2018). *UglifyJS*. URL: http://lisperator.net/uglifyjs/.

Broder, Anatol und Open Source Community (18. 06. 2018). *jekyll-compress-html*. URL: http://jch.penibelst.de/.

Browsershots und Open Source Community (18. 06. 2018). *Browsershots*. URL: http://browsershots.org/.

Catlin, Hampton, Natalie Weizenbaum, Chris Eppstein und Open Source Community (15. 03. 2018). *Sass - Syntactically Awesome Stylesheets*. URL: https://sass-lang.com/.

Chart.js und Open Source Community (14. 06. 2018). *Chart.js*. URL: http://www.chartjs.org/.

Douglas, David H und Thomas K Peucker (12/1973). „Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature“. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2, S. 112–122. DOI: 10.3138/fm57-6770-u75u-7727.

Eckert, Peter und Staatliche Realschule Gauting (03. 07. 2018). *Das iPad im Unterricht an der RSG - 90% der Schüler würden wieder in die iPad-Klasse gehen*. URL: https://rsgipad.wordpress.com/2016/06/21/90-der-schueler-wuerden-wieder-in-die-ipad-klasse-gehen/.

Environmental Systems Research Institute, Inc. (Esri) (1996). *Automation of Map Generalization*. URL: http://downloads.esri.com/support/whitepapers/ao_/mapgen.pdf.

– (23. 07. 2018). *Esri Academy*. URL: https://www.esri.com/training/.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander und Xiaowei Xu (1996). „A density-based algorithm for discovering clusters in large spatial databases with noise“. In: *Kdd*. Bd. 96. 34, S. 226–231. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.121.9220.

Fei, Christian (17. 05. 2018). *Simple-Jekyll-Search*. URL: https://github.com/christian-fei/Simple-Jekyll-Search.

Fierens, Wout und Open Source Community (13. 06. 2018). *SVG.js*. URL: http://svgjs.com/.

Fisler, Joël, Susanne Bleisch und Robert Weibel (2006). „Das e-Learning-Projekt GITTA: Frei zugängliche Inhalte für die akademische Ausbildung in Geoinformation“. In: *Lernen mit Geoinformation (AGIT proceedings - Themenschwerpunkt Geoinformation in der*

Schule). Hrsg. von Thomas Jekel, Alfons Koller und Josef Strobl. URL: http://www.elml.org/website/en/download/publications/agit_gitta.pdf.

GitHub, Inc. (08. 02. 2018). *GitHub*. URL: https://github.com/.

GitLab, Inc. (08. 02. 2018). *GitLab*. URL: https://gitlab.com/.

Google Chrome und Open Source Community (18. 06. 2018). *Lighthouse*. URL: https://developers.google.com/web/tools/lighthouse/.

Google LLC (17. 11. 2017). *Material Design*. URL: https://material.io/.

– (14. 06. 2018a). *Google Fonts*. URL: https://fonts.google.com/.

– (13. 06. 2018b). *Google Maps Elevation API*. URL: https://developers.google.com/maps/documentation/elevation/start.

Google LLC und Open Source Community (18. 06. 2018). *Workbox*. URL: https://developers.google.com/web/tools/workbox/.

Grinstead, Brian und Open Source Community (14. 06. 2018). *Spectrum*. URL: https://bgrins.github.io/spectrum/.

GT.net (18. 06. 2018). *GTmetrix Performance Report*. URL: https://gtmetrix.com/reports/moritzbru.gitlab.io/iSjDzk4p.

Hake, Günter, Dietmar Grünreich und Liqiu Meng (12. 03. 2013). *Kartographie*. Gruyter, Walter de GmbH. 617 S. URL: https://www.degruyter.com/view/product/12592.

Hrastinski, Stefan (2008). „Asynchronous and synchronous e-learning“. In: *Educause quarterly* 31.4, S. 51–55. URL: https://er.educause.edu/articles/2008/11/asynchronous-and-synchronous-elearning.

Imhof, Eduard (2007). *Cartographic Relief Presentation*. URL: https://books.google.bg/books?id=cVy1Ms43fFYC&printsec=frontcover#v=onepage&q&f=false.

Informationsbüro ZukunftsForum (28. 04. 2018). *ZukunftsMonitor III „Lehren, Lernen und Leben in der digitalen Welt“*. Hrsg. von Bundesministerium für Bildung und Forschung. URL: https://www.zukunft-verstehen.de/application/files/7814/7636/3024/BMBF_ZF_III_ZukunftsMonitor_Ergebnisse.pdf.

Internet Engineering Task Force (IETF) (2014). *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. URL: https://tools.ietf.org/pdf/rfc7231.pdf.

Jekyll und Open Source Community (14. 03. 2018). *Jekyll*. URL: https://github.com/jekyll/jekyll/blob/master/README.markdown.

Joubel und Open Source Community (14. 06. 2018). *H5P*. URL: https://h5p.org/.

Jung, Tobias (17. 07. 2018). *Compare Map Projections*. URL: https://map-projections.net/index.php.

Kahiigi, Evelyn Kigozi, Love Ekenberg, Henrik Hansson, Francis F. Tusubira und Mats Danielson (2008). „Exploring the e-Learning State of Art“. In: *The Electronic Journal of e-Learning* 6 (2), S. 77–88. URL: http://ejel.org/issue/download.html?idArticle=67.

MacQueen, James (1967). „Some methods for classification and analysis of multivariate observations“. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, S. 281–297. URL: https://projecteuclid.org/euclid.bsmsp/1200512992.

Mapbox Inc. und Open Source Community (13. 06. 2018a). *Mapbox GL JS*. URL: https://www.mapbox.com/mapbox-gl-js/api/https://www.mapbox.com/mapbox-gl-js/api/.

– (13. 06. 2018b). *Turf.js*. URL: http://turfjs.org/.

MathJax Consortium (13. 06. 2018). *MathJax*. URL: https://docs.mathjax.org/en/latest/index.html.

Młokosiewicz, Jakub und Open Source Community (14. 06. 2018). *k-means-visualization*. URL: https://github.com/hckr/k-means-visualization.

Moodle Pty Ltd und Open Source Community (18. 07. 2018). *Moodle*. URL: https://moodle.org/.

Oosterhof, Richard (18. 06. 2018). *Richard's Toolbox*. URL: https://richardstoolbox.com/.

Otto, Mark, Jacob Thornton, Twitter Inc. und Open Source Community (17. 11. 2017). *Bootstrap*. URL: https://getbootstrap.com/.

Ramer, Urs (11/1972). „An iterative procedure for the polygonal approximation of plane curves“. In: *Computer Graphics and Image Processing* 1.3, S. 244–256. DOI: 10.1016/s0146-664x(72)80017-0.

Schnabel, O., R. Stopper und L. Hurni (2007). „New modular approach for knowledge-transfer in multimedia cartography: The e-learning project CartouCHe“. In: *Proceedings of the International Cartographic Conference*. URL: http://www.e-cartouche.ch/material/paper_schnabel_stopper_hurni_cartouche.pdf.

Schweizerische Gesellschaft für Kartographie, Hrsg. (1990). *Kartographisches Generalisieren*. SGK Publikationen. Bd. 10. URL: http://kartografie.ch/publikationen/sgk-publikationen/.

Sherry, Lorraine und Brent Wilson (1997). „Transformative communication as a stimulus to Web innovations“. In: *Web-based instruction*, S. 67–73. URL: https://books.google.de/books?hl=de&lr=&id=natcmen0J_gC&oi=fnd&pg=PA67&dq=Transformative+communication+as+a+stimulus+to+Web+innovations&ots=ftHFrD0kKp&sig=iLqtTbO6hT90dAXfEykQeozjnjg#v=onepage&q=Transformative%20communication%20as%20a%20stimulus%20to%20Web%20innovations&f=false.

Shopify und Open Source Community (18. 05. 2018). *Liquid*. URL: https://shopify.github.io/liquid/.

StatCounter (18. 06. 2018). *Browser Market Share Europe*. URL: http://gs.statcounter.com/browser-market-share/all/europe/#monthly-201706-201805-bar.

Sugar, Corneliu und Open Source Community (14. 06. 2018). *jDBSCAN*. URL: https://github.com/upphiminn/jDBSCAN.

Sun, Lily, Shirley Williams, Khadidjatou Ousmanou und Jude Lubega (2003). „Building Personalised Functions into Dynamic Content Packaging to Support Individual Learners“. In: *Proceedings of the 2nd European Conference on e-Learning, Glasgow*, S. 439–448. URL: http://www.ais.reading.ac.uk/papers/con41-building%20personalised.pdf.

Sun, Pei-Chen, Ray J. Tsai, Glenn Finger, Yueh-Yang Chen und Dowming Yeh (05/2008). „What drives a successful e-Learning? An empirical investigation of the critical factors influencing learner satisfaction“. In: *Computers & Education* 50.4, S. 1183–1202. DOI: 10.1016/j.compedu.2006.11.007.

Technische Universität München (27. 03. 2018). *Corporate Design der Technischen Universität München*. URL: https://portal.mytum.de/corporatedesign/index_html/.

Torvalds, Linus, Junio C. Hamano, Jeff King, Shawn O. Pearce, Johannes Schindelin, Duy Nguyen und Open Source Community (08. 02. 2018). *Git*. URL: https://git-scm.com/.

Tunapanda und Open Source Community (17. 07. 2018). *h5p-standalone*. URL: https://github.com/tunapanda/h5p-standalone.

Webster, R. und F. Sudweeks (2006). „Teaching for e-Learning in the knowledge society: Promoting conceptual change in academics' approaches to teaching“. In: *Current Developments in Technology-Assisted Education*, S. 631–635. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.1799&rep=rep1&type=pdf.

WordPress Foundation, Matt Mullenweg und Open Source Community (07. 05. 2018). *WordPress.org*. URL: https://wordpress.org/.

Anhang: Guide zur Erstellung neuer Playgrounds

Der folgende Guide ist auch online auf der GitLab Seite des Projekts unter `https://gitlab.com/MoritzBru/cartography-playground/blob/master/HOW-TO-CONTRIBUTE.md` verfügbar.

CARTOGRAPHY **PLAYGROUND**

How to contribute

Requirements

- GitLab account [*optional*]
- Git GUI (e.g. GitKraken) [*optional*]
- Node.js
- Ruby
- Text editor (e.g. Atom)

Working with git

If you do **not** want to work with Git and GitLab you can just download the project and continue working just with your local copy offline. However, this has two drawbacks:

1. You have to pull the latest updates manually and probably override your local changes
2. You cannot send a pull request and contribute to the project by adding your playground

Now that you are convinced to work with Git, it is probably best to download a Git GUI like for example GitKraken.

If you don't have one yet, now is the time to create a GitLab account.

Now go to the Cartography Playground GitLab Repository and `Fork` the project.

In GitKraken *Clone Repo*, log in to GitLab and clone your Fork of Cartography Playground to a local folder on your computer.

Create a new *branch* from the lastest commit in the `master` *branch* and name it like your new playground.

Preparing a local build environment

If it is not already installed, install Node.js and Ruby.

Open a terminal in the root folder `cartography-playground`.

> Hint: On Windows this can be done by pressing `shift` + right-clicking in an empty spot in the folder and selecting *open command/powershell window here* from the menu.

Run the command `bundle` to download all dependencies (e.g. Jekyll etc.) that are needed for the project.

Now you can build the project and run it on a local server by typing `bundle exec jekyll serve`. Amongst other things, the output should print somethings like `Server address: http://127.0.0.1:4000/cartography-playground/` so go to that address in your preferred Browser (Internet Explorer is not supported). You should now see the Cartography Playground website served locally. As long as the server is running it updates automatically when you change some source files.

> Hint: If you do not see your most recent changes you might have to force reload (= clear cache and reload) the page by pressing `ctrl` + `F5` in your browser.

Getting to know the project

The folder structure of the project should look something like this:

```
cartography-playground
│   404.html
│   Gemfile # includes Ruby dependencies like Jekyll
│   Gemfile.lock
│   index.html
│   legal-notice.html
│   README.md
│   site.webmanifest
│   sw.js
│   _config.yml # Jekyll configuration
│
├──assets # contains image assets
│       cartography-playground-logo-line.svg
│       cartography-playground-logo-square.svg
│       og-image.png
│
├──css
│   │   style.scss
│   │   topography-dark.svg
│   │   topography-light.svg
│   │
│   └──_scss
│       ├──bootstrap
│       │       bootstrap.scss
│       │       ... # bootstrap style scss files
│       │
│       └──custom
│               ... # custom style scss files
│
├──favicon
│       favicon.ico
│       ... # favicons for different platforms
│
├──search
│       search.html
│       search.js
│       search.json # search database
│
├──_includes # Parts that are included elsewhere
│   │   head.html
│   │   footer.html
│   │   js.html
│   │   ... # Jekyll includes (HTML)
│   │
│   └──svg
│           404.svg
│           logo.svg
│           ... # Jekyll includes (Inline SVG)
│
├──_layouts # Layout templates for Jekyll
│       compress.html
│       default.html
│       playground.html
│
├──_plugins # Ruby plugins
│       expand_nested_variable_filter.rb
│       js_compressor.rb
│
└──_playgrounds # Each playground has its own subfolder here
    ├──cartographic-generalization
    │       ... # Playground files
    │
    ├──clustering-comparison
    │       ... # Playground files
    │
    ├──contour-lines-to-profile
    │       ... # Playground files
    │
    ├──douglas-peucker-algorithm
    │       ... # Playground files
    │
    ├──map-design
    │       ... # Playground files
    │
    └──quiz
            ... # Playground files
```

Cartography Playground uses Jekyll and it's templating language Liquid, which can be categorized into *objects*, *tags*, and *filters*. *objects* are written in double curly braces `{{ ... }}` and contain variables that become the pages content. *filters* are appended to *objects* separated by `|` and change their outputs. *tags* are denoted by curly braces with percentage signs `{% ... %}` and contain control flow and logic.

```
<!-- object -->
{{ page.title }}
<!-- Output: Cartography Playground -->

<!-- filter -->
{{ page.title | upcase | prepend: "Hello " }}
<!-- Output: Hello CARTOGRAPHY PLAYGROUND -->

<!-- tag -->
{% for playground in site.playgrounds %}
  <h1>{{ playground.title }}</h1>
  <p>{{ playground.description }}</p>
{% endfor %}
<!-- Loop through all playgrounds and print their title as heading and description as paragraph -->
```

As you can see in the examples above, Liquid is used to extend `HTML` or `Markdown` markup which contains the content of the websites. Other languages used for Cartography Playground are `CSS/SCSS` for styling the content, `SVG` for displaying nice vector images, `JavaScript` for adding interactivity to the pages and `Ruby` for extending Jekyll at compilation time.

Adding a new playground

If you want to add a new playground you have to create a new folder inside the `_playgrounds` folder. Name your folder the same as your content file. The name should be clear and short and not include umlauts or other special characters and have dashes instead of spaces because that name will be the url of the new playground. The content file can either be an `HTML`-file or for easier writing a `Markdown`-file. Have a look at the Markdown-Cheatsheet for infos on how to write `Markdown`.

Frontmatter

Your content file then need some `frontmatter` which is written in `YAML` between three dashes in the first lines of the file:

- title
  - obligatory
  - nice Title for the page that gets displayed on top of the page and on the card for the playground selection on the homepage
  - use quotes `"..."`
- description
  - obligatory
  - short description that gets displayed on the card for the playground selection
  - use quotes `"..."`
- date
  - obligatory
  - date when file was created
  - format: `yyyy-mm-dd`
  - used for sorting playground on the homepage
- tags
  - obligatory
  - tags used for the search search database
  - tags separated by space, no commas between tags
  - no quotes needed
- references
  - optional
  - references for citations or further information
  - format: see example
- head_additional
  - optional
  - html to be included in the head section
  - usually used for including additional Stylesheets
- foot_additional
  - optional
  - html to be included at the end of the body section
  - usually used for including additional Scripts

Example

```
---
title: "New Playground Example"
description: "This is an awesome new playground example to display how everything works."
date: 2018-07-31
tags: example new playground contribute guide
references:
  - author: "Albert Einstein"
    title: "Can quantum-mechanical description of physical reality be considered complete?"
    date: May 1935
    link: https://journals.aps.org/pr/pdf/10.1103/PhysRev.47.777
  - author: "Wikipedia"
    title: "Cartography"
    date: July 2018
    link: https://en.wikipedia.org/wiki/Cartography
foot_additional: >
  {% include mathjax.html %}
  {% include mapbox.html %}
  <script src="{{ 'new-playground.js' | prepend: page.url | absolute_url }}"></script>
---
```

Bootstrap

Cartography Playground uses Bootstrap so you can also use all the Bootstrap components for laying out your page.

> Hint: you can also use `HTML`-snippets inside of a `Markdown`-file.

MathJax

If you want your site to have beautiful math formulas you can include MathJax by adding `{% include mathjax.html %}` to your `foot_additional` and then write LaTeX formulas between `\( ... \)` for inline math and `\[ ... \]` for display math.

Example

```
The general quadratic equation is
\[ ax^2 + bx + c = 0 \]
where \( x \) represents the unknown variable and \( a \), \( b \) and \( c \) are constants with \( a \neq 0 \).
Solved for \( x \) the quadratic formula is
\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]
```

Mapbox

For including a map in your page, you have to add `{% include mapbox.html %}` to your `foot_additional`, first. Then you need to create a container for your map by adding `<div id="map"></div>` where you want your map to appear. Finally, you have to create a `JavaScript` file in your new playground folder and include that also in your `foot_additional` like this `<script src="{{ 'NAME-OF-YOUR-JS-FILE.js' | prepend: page.url | absolute_url }}"></script>`. Inside your `.js` file you nee to initialize the map:

```
mapboxgl.accessToken = mapbox_accessToken;
var map = new mapboxgl.Map({
  container: "map",
  style: "mapbox://styles/mapbox/streets-v10"
});
```

You can of course include more options, other styles, or other interaction methods as you like.

Scalable Vector Graphics

Cartography Playground uses only `SVG`s, no raster images. You can either use SVG.js by including `{% include svgjs.html %}` in your `foot_additional` and create the `SVG`s programmatically with `JavaScript` or create the `SVG`s in your favorite graphics editor (e.g. Inkscape), save them in your new playground folder and include them in your page for example via `img`-tag like

```
<img class="img-fluid mx-auto d-block" src="NAME-OF-YOUR-SVG.svg" alt="ALTERNATIVE TITLE FOR YOUR SVG">
```

or include them inline with Jekyll like

```
{% include_relative NAME-OF-YOUR-SVG.svg %}
```

To make sure your image fits in the design properly use the following colors:

#f8f9fa	#343a40	#0065bd	#a2ad00	#e37222	#e5e3da
					

Other Libraries

Of course you can use other libraries as you like. If you include other dependencies, try to include them from a content delivery network - preferably CDNJS. Please also add the new library to the dependencies in the README.md of the project to make it easier to maintain and keep track of the dependencies.

Publishing the new playground

If you want your changes to appear on the *original* Cartography Playground `commit` and `push` your change to your fork of the project on GitLab and send a `pull/merge request`. You then have to wait for a maintainer of the project to merge your changes.

If something gets merged, you can follow the CI/CD process here.

If you want to make a local *production* build and push the files manually to another server, you first have to adapt the target url in the _cofing.yml by changing the `baseurl` and `url` parameters. Then you have to run following command (ion Windows PowerShell)

```
& { $JEKYLL_ENV = 'production'; $env:JEKYLL_ENV = 'production'; bundle exec jekyll serve }
```

You can find the minified and compressed output in the `_site` folder. The content of that `_site` folder can then just be uploaded on your desired server.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne Benutzung anderen als der angegebenen Hilfsmittel angefertigt habe. Wörtliche und sinngemäße Zitate sind durch Quellenangaben kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

München, 31.07.2018

Moritz Brunnengräber