**Untersuchung von JavaScript-Frameworks für Geoportale zur Präsentation verteilter Geoinformation**

**A study of JavaScript-Frameworks for presenting distributed geographic information in geoportal**

An der RWTH Aachen,

Fakultät für Bauingenieurwesen,

Geodätisches Institut und

Lehrstuhl für Bauinformatik & Geoinformationssysteme

**Masterarbeit**

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

vorgelegt von

**Min Xie**

geboren am 29.08.1992 in Zhejiang

Matrikel-Nr.: 371773

Erstgutachter:

Zweitgutachter:

Eingereicht am:

# Inhaltsverzeichnis

[Inhaltsverzeichnis I](#_Toc535230106)

[Abbildungsverzeichnis III](#_Toc535230107)

[Tabellenverzeichnis IV](#_Toc535230108)

[Quellcodeverzeichnis V](#_Toc535230109)

[1 Einleitung: Der Hintergrund und die Motivation 1](#_Toc535230110)

[1.1 Vorstellung des Projekts EarlyDike Geoportal und die Schwierigkeiten in der Praxis 1](#_Toc535230111)

[1.2 Allgemeine Vorteile der Implementierung von JavaScript-Frameworks 1](#_Toc535230112)

[2 Die Architekturmuster für den Aufbau einer Web-Anwendung 4](#_Toc535230113)

[2.1 Die Architekturmuster und ihre Beziehungen zu JavaScript-Frameworks 4](#_Toc535230114)

[2.2 Model-View-Controller (MVC) und Model-View-Presenter (MVP) 5](#_Toc535230115)

[2.2.1 Grundstruktur 5](#_Toc535230116)

[2.2.2 Workflow und Beispiele 6](#_Toc535230117)

[2.2.3 Fazit 7](#_Toc535230118)

[2.3 Model-View-ViewModel (MVVM) 7](#_Toc535230119)

[3 Vergleich und Auswahl von JavaScript-Frameworks 11](#_Toc535230120)

[3.1 Allgemeine Vergleichung 11](#_Toc535230121)

[3.1.1 Popularität und Geschichte 11](#_Toc535230122)

[3.1.2 Lernkurve 12](#_Toc535230123)

[3.1.3 Besonderheiten 14](#_Toc535230124)

[3.2 Praxisnahe Vergleichung: Implementierungsflexibilität 14](#_Toc535230125)

[3.3 Fazit 15](#_Toc535230126)

[4 Vorbereitung für die Implementierung des JavaScript-Frameworks Vue in das Projekt EarlyDike 17](#_Toc535230127)

[4.1 Angestrebte Ziele 17](#_Toc535230128)

[4.2 Hintergrund und Arten der Implementierung 18](#_Toc535230129)

[4.2.1 Hintergrund 18](#_Toc535230130)

[4.2.2 Arten der Implementierung 19](#_Toc535230131)

[4.3 Entscheidung der Art der Implementierung 24](#_Toc535230132)

[5 Versuch 1: Implementierung durch CDN (s. Anhang 4) 26](#_Toc535230133)

[5.1 Grundprinzip: Migration 26](#_Toc535230134)

[5.2 Grundstruktur des Projekts EarlyDike 27](#_Toc535230135)

[5.3 Technische Details der Migration 27](#_Toc535230136)

[5.3.1 Allgemeine Hinweise 27](#_Toc535230137)

[5.3.2 Ordentlicher Aufbau mit Vue Components 28](#_Toc535230138)

[5.3.3 Template Syntax von Vue 29](#_Toc535230139)

[5.3.4 Obligatorische Änderungen bei der Migration 31](#_Toc535230140)

[5.4 Fazit 31](#_Toc535230141)

[6 Versuch 2: Implementierung durch CLI (s. Anhang 5) 34](#_Toc535230142)

[6.1 Grundprinzip: Wiederaufbau 34](#_Toc535230143)

[6.2 Initialisierung des Projekts EarlyDike Geoportal mit der Hilfe von Vue CLI 3 34](#_Toc535230144)

[6.3 Grundstruktur des Projekts EarlyDike 35](#_Toc535230145)

[6.4 Technische Details 37](#_Toc535230146)

[6.4.1 Details in globaler Ebene 37](#_Toc535230147)

[6.4.2 Details in Komponenten 43](#_Toc535230148)

[6.5 Projekt übergreifend wiederverwendbare Dateien 47](#_Toc535230149)

[6.6 Fazit 48](#_Toc535230150)

[7 Zusammenfassung 49](#_Toc535230151)

[Literaturverzeichnis VII](#_Toc535230152)

[Anhang 1: Beispiel - MVC und MVP Architekturmuster X](#_Toc535230153)

[Anhang 2: Beispiel - plain JavaScript vs. JavaScript‑Framework Vue XI](#_Toc535230154)

[Anhang 3: Schritten der Migration XII](#_Toc535230155)

[Anhang 4: Codes von EarlyDike Geoportal (CDN Version) XIII](#_Toc535230156)

[Anhang 5: Codes von EarlyDike Geoportal (CLI Version) XIV](#_Toc535230157)

[Anhang 6: Grundstruktur von EarlyDike Geoportal (CLI Version) XV](#_Toc535230158)

[Anhang 7: Code Beispiel über Veux XVI](#_Toc535230159)

# Abbildungsverzeichnis

[Abbildung 2‑1: Grundstruktur MVC (Ackermann 2015, S. 13) 5](#_Toc535229524)

[Abbildung 2‑2: Grundstruktur MVP (Ackermann 2015, S. 13) 5](#_Toc535229525)

[Abbildung 2‑3: Workflow MVC (Bashir 2014) 6](#_Toc535229526)

[Abbildung 2‑4: Workflow MVP (Bashir 2014) 7](#_Toc535229527)

[Abbildung 2‑5: Grundstruktur MVVM (Ackermann 2015, S. 16) 8](#_Toc535229528)

[Abbildung 2‑6: plain JS vs. Vue (eigene Darstellung) 9](#_Toc535229529)

[Abbildung 3‑1: Numbers of stars added on GitHub for Front-end Frameworks in 2017 (Quelle: https://risingstars.js.org/2017/en/#section-framework) 11](#_Toc535229530)

[Abbildung 3‑2: npm downloads of the top four front end frameworks at 14.07.2018 (Quelle: http://www.npmtrends.com/react-vs-vue-vs-angular-vs-@angular/core) 12](#_Toc535229531)

[Abbildung 3‑3: Lernkurve Angular vs. React vs. Vue (Quelle: Schwarzmüller 2017) 14](#_Toc535229532)

[Abbildung 4‑1: Beispiel: Liste von Dependencies in package.json Datei (eigene Darstellung) 22](#_Toc535229533)

[Abbildung 4‑2: Vordefinierte Struktur von neu erstelltem Projekt mit extra Anforderungen an TypeScript- und Vuex-Unterstützung (eigene Darstellung) 24](#_Toc535229534)

[Abbildung 5‑1: Grundstruktur des Projekts EarlyDike - CDN Version (eigene Darstellung) 27](#_Toc535229535)

[Abbildung 5‑2: Von Web-Anwendung zu vernetzten Komponenten 28](#_Toc535229536)

[Abbildung 5‑3: Benutzeroberfläche der EarlyDike Geoportal (CDN Version) 33](#_Toc535229537)

[Abbildung 6‑1: Ofizielle Plugins von Vue CLI (eigene Darstellung) 35](#_Toc535229538)

[Abbildung 6‑2: Kommunikation zwischen Vue Components (eigene Darstellung) 39](#_Toc535229539)

[Abbildung 6‑3: Kommunikation zwischen Vue Components durch Vuex (eigene Darstellung) 41](#_Toc535229540)

[Abbildung 6‑4: Ablaufdiagramm: systematische Implementierung der Layers (eigene Darstellung) 44](#_Toc535229541)

[Abbildung 6‑5: Struktur der Auswahllisten von Layers (eigene Darstellung) 45](#_Toc535229542)

[Abbildung 6‑6: Aktualisierung der Daten von Informationstab mit Vuex (eigene Darstellung) 46](#_Toc535229543)

# Tabellenverzeichnis

[Tabelle 3‑1: zusätzliche Kenntnisse für jeweilige JS-Frameworks (eigene Darstellung) 13](#_Toc535229544)

[Tabelle 3‑2: Implementierungsmöglichkeiten von JS-Frameworks 15](#_Toc535229545)

[Tabelle 4‑1: Erfüllungsgrad der angestrebten Ziele von unterschiedlichen Arten der Implementierung (eigene Darstellung) 24](#_Toc535229546)

[Tabelle 6‑1: Interaktionen und Auswirkungen auf die Informationstabs 46](#_Toc535229547)

[Tabelle 6‑2: Projekt übergreifend wiederverwendbare Dateien 47](#_Toc535229548)

# Quellcodeverzeichnis

[Quellcode 4‑1: Einbindung des JavaScript-Framework Vue durch CDN Link in HTML (eigene Darstellung) 20](#_Toc535229549)

[Quellcode 4‑2: Eirichtung von Dependencies durch NPM: Vue und Vue CLI 3 (eigene Darstellung) 21](#_Toc535229550)

[Quellcode 4‑3: Einrichtung der Entwicklungsumgebung durch NPM (eigene Darstellung) 21](#_Toc535229551)

[Quellcode 5‑1: Registrierung globaler Komponente in Vue (eigene Darstellung) 29](#_Toc535229552)

[Quellcode 5‑2: Direkte DOM-Manipulationsmethoden 30](#_Toc535229553)

[Quellcode 5‑3: Template von Vue Instance / Component (eigene Darstellung) 30](#_Toc535229554)

[Quellcode 6‑1: Initialisierung eines neuen Projekts mit Vue CLI (eigene Darstellung) 34](#_Toc535229555)

[Quellcode 6‑2: Struktur eines Single‑File Component 36](#_Toc535229556)

[Quellcode 6‑3: Auf ES6 Modules basierte Implementierung von OpenLayers 5 38](#_Toc535229557)

[Quellcode 6‑4: Anwendungsbeispiel: Vuetify UI-Component Lists im EarlyDike Geoportal (eigene Darstellung) 43](#_Toc535229558)

[Quellcode 6‑5: Zusammengefasste Parameter von einem Layer in GIAGS\_config.json (eigene Darstellung) 44](#_Toc535229559)

Abkürzungsverzeichnis

C

CDN · *content delivery network or content distribution network*

CLI · *command-line interface or command language interpreter*

G

GUI · *Graphical user interface*

J

**JSX** · *JavaScript XML*

M

MVC · *Model View Controller*

MVP · *Model–view–presenter*

MVVM · *Model–view–viewmodel*

N

NPM · *Node package manager*

S

SFC · *Single-File Components*

SPA · *Single-page application*

U

UI · *User interface*

W

WFS · *Web Feature Service*

WMS · *Web Map Service*

WPS · *Web Processing Service*

# Einleitung: Der Hintergrund und die Motivation

## Vorstellung des Projekts EarlyDike Geoportal und die Schwierigkeiten in der Praxis

Das EarlyDike Geoportal ist eine Single-Page-Webanwendung (SPA), die als zentraler Zugang zu verteilter Geoinformation dient (s. Abb. 1-1). Dienste wie Web Map Service (WMS), Web Feature Service (WFS) aber auch Web Processing Service (WPS) werden für das Geoportal eingesetzt. Einerseits soll eine große Menge von Geoinformationen aus unterschiedlichen Quellen zusammengefasst und sinnvoll dargestellt werden, andererseits soll eine vernünftige Benutzeroberfläche dafür errichtet werden.

Das derzeitige zur Verfügung gestellte EarlyDike Geoportal wurde seit langem von GIA entwickelt und hauptsächlich mit „plain“ JavaScript und einigen JavaScript‑Bibliotheken geschrieben. Die Struktur von der Web-Anwendung ist leider vom Anfang an nicht ausreichend geplant und im Laufe der Zeit erhöhen sich die Wartungsaufwand sowie die Erweiterungsaufwand immer weiter, obwohl die Funktionalitäten der Web-Anwendung eigentlich vollständig da sind. Eine Optimierung der Struktur ist für die zukünftige Entwicklung nicht nur bei diesem Projekt, aber für andere Projekte, die ähnliche Grundstruktur haben sollten, erwünscht.

Außerdem ist die gegenwärtige Benutzeroberfläche nicht optimal für eine moderne Web-Anwendung. Zum Beispiel bekommen die Benutzer an manchen Stellen nicht genug Hinweise über die versteckten Funktionalitäten bzw. die Geoinformationen sind manchmal nicht geeignet dargestellt. Eine moderne Web-Anwendung soll dem Benutzer möglichst viel Hinweise geben, um eine benutzerfreundliche Umgebung zu schaffen, in der der Benutzer ohne besondere Erlernung die Web-Anwendung auch einwandfrei bedienen kann.

Die oben genannten Probleme sollten im Rahmen dieser Arbeit durch Anwendung neuester Web-Technologien (z. B. JavaScript-Frameworks) gelöst werden. In nächstem Kapitel werden einige allgemeinen Vorteile dargestellt, die durch Implementierung von JavaScript-Frameworks gebracht werden können.

## Allgemeine Vorteile der Implementierung von JavaScript-Frameworks

Jeder, der schon mal mit JavaScript eine Web-Anwendungen geschrieben hat, hat bestimmt auch Erfahrungen mit JavaScript-Bibliothek gehabt. Die berühmtesten davon sind z. B. jQuery, Vanilla JS, etc. Was bietet die JavaScript-Bibliothek uns eigentlich? Um diese Frage zu beantworten, einfach daran denken: In welcher Situation werden die Bibliotheken gesucht und eventuell für ein Projekt eingesetzt? Die Antwort ist klar: Das Hunderttausend von JavaScript-Bibliotheken können haufenweise Zeit für uns sparen.

In der Welt von JavaScript ist es ganz normal, dass die unterschiedlichen Entwickler genau die gleichen Bedürfnisse haben. Natürlich kann bzw. möchte nicht jeder Entwickler eine Lösung für sich finden. Manchmal haben die erfahrenen Entwickler die Lösung (JavaScript-Codes) in einer JS-Datei eingepackt und veröffentlicht. Und die Entwickler, die die Bedürfnisse haben, binden einfach diese JS-Datei in eigenes Projekt ein, rufen die erwünschten Funktionen auf, usw. Die Nutzung von JavaScript-Bibliotheken ist ein ganz normaler Schritt bei der Web-Entwicklung.

Obwohl die JavaScript-Bibliothek Zeit für uns sparen kann, beschränkt sie während der Anwendung nicht, wie die Codes organisiert werden sollen. D. h.: Es könnte sein, dass nachdem der Entwickler Dutzend JavaScript-Bibliotheken in sein Projekt eingebunden und zahlreiche Funktionen von solchen Bibliotheken aufgerufen hat, sehen die Codes super unübersichtlich aus. Dies hat einen fürchterlichen Nachteil: nämlich die Wartbarkeit ist damit so untergegangen, dass nach einer Weile sogar der Entwickler selber auch viel Zeit braucht, um sein eigenes Projekt zuerst zu verstehen und dann zu warten.

Ein JavaScript-Framework ist abstrakt betrachtet ein Rahmen, der uns die Möglichkeit bietet, Web-Anwendungen nach bestimmten Regeln, Schritten und auch vordefinierter Struktur zu entwickeln (Vgl. Kapitel 2.1). Die meisten JavaScript-Frameworks bezwingen bzw. empfehlen einen modularen Aufbau bei der Entwicklung und werden selbst nach einem bestimmten Architekturmuster entwickelt. D. h. das Framework hilft den Entwickler mehr oder weniger eine übersichtliche Struktur zu schaffen und saubere und effiziente Codes zu schreiben.

Ein weiterer Vorteil liegt bei der Anwendung von virtual DOM. Codes wie documente.createElement() oder dom.addEventListener (macht das Form richtig wie Codes) sind uns bekannt als DOM-Manipulationsmethode um UI (User interface) zu aktualisieren. Das ist aber nicht so leistungsstark besonders wenn man komplexe DOM-Struktur hat und viele DOM-Manipulationen ständig durchführen möchte. Das JavaScript-Framework hat eigenes virtual DOM und muss nicht für jedes einzelne Element das echte DOM manipulieren (was dann große Rechenaufwand verursachen könnte). Die erwünschten Änderungen werden zuerst nur im virtual DOM durchgeführt (nur kleine Rechenaufwand). Danach werden nur die minimalen erforderlichen Änderungen bei echtem DOM. Virtual DOM ist im Vergleich zur normalen direkten DOM-Manipulationsmethode sehr leistungsstark und bietet auch viele andere Vorteile bei der Entwicklung (Komanduri 2016).

* The main thing being solved is having a consistent and right way to do UI

# Die Architekturmuster für den Aufbau einer Web-Anwendung

## Die Architekturmuster und ihre Beziehungen zu JavaScript-Frameworks

Die Architekturmuster sind abstrakte, grundlegende Lösungsansätze für den Aufbau einer Web-Anwendungen. Das Architekturmuster (in English achitecture, pattern oder idiom) ist der Grundstein für die Entwicklung und Anwendung jeder JavaScript-Frameworks und gleichzeitig auch ein sehr wichtiges Basiswissen für jeden Frontend-Entwickler.

Der Grund und die Motivation, ein Architekturmuster zu entwickeln, sind mit einer praxisnahen Problemstellung eng verbunden: Die GUI (Graphical User Interface) von einer Anwendung muss immer in Laufe der Zeit aktualisiert werden, gleichzeitig bleiben die Daten im Hintergrund aber meistens unverändert. Wenn man spezielle GUI für eine bestimmte Anwendung entwickelt, wird einerseits die Anwendung weniger flexibel und schwer zu migrieren, andererseits die GUI selber auch schwer für andere Anwendungen wiederzuverwenden. Deswegen hat Smalltalk in 70er das erste Architekturmuster MVC (Model-View-Controller) definiert, zwar mit dem folgenden Grundgedanken: Es ist doch sinnvoll, die tatsächlichen Inhalte (die Daten bzw. Datenmodelle) von GUI zu entkoppeln (Deacon 2005). Dieser Grundgedanke hat sich bis heute nicht verändert und ist das gleiche Hauptziel von allen Architekturmustern auf dem Markt.

Ein Architekturmuster ist kein Framework und steht für sich allein. Ein Architekturmuster ist ein disziplinierter Ansatz zur Lösung eines Code-Problems (Reyes 2017). Ein Framework, im Vergleich dazu, bietet ein Rahmen an und bezwingt den Entwickler mehr oder weniger, damit bei der Entwicklung eine konstante Vorgehensweise eingehalten wird. Dieses Verhältnis nennt man „Inversion of Control“ und das ist auch der Hauptunterschied zwischen einem Framework und einer Bibliothek: Bei der Anwendung von JavaScript-Bibliotheken werden Funktionen von den Entwicklern aufgerufen und die Entwickler haben die volle Kontrolle. Bei der Anwendung von JavaScript-Framework haben die Entwickler keine volle Kontrolle mehr. Stattdessen müssen die Entwickler eigene Codes in festgelegte Strukturen des Frameworks einbetten, um bestimmte Funktionalitäten zu realisieren. Das Framework, diesmal umgekehrt, ruft die Codes von Entwicklern auf, sobald sie benötigt werden (1&1 Ionos SE, 2018) Dadurch wird bei der Entwicklung eine konstante Vorgehensweise eingehalten und am Ende kommt die Struktur der Codes näher zu einem bestimmten Architekturmuster. Ob die Struktur der gesamten Codes am Ende eher ein Architekturmuster wie MVC, MVP oder MVVM besitzt, hat jedes Framework eigene Interpretation.

Eins muss noch beachtet werden: Obwohl es bestimmten Namen für verschiedene Architekturmuster gibt (wie MVC, MVP und MVVM. S. Kapitel 2.2 und 2.3), d. h. aber nicht, dass jedes Framework, das das gleiche Architekturmuster folgt, das Architekturmuster aus jedem Aspekt exakt gleich interpretieren muss. Es handelt sich eher um ein unscharfes Architekturmuster, das lediglich eine grundlegende Lösungsansätze uns bietet.

Im folgenden Kapiteln werden drei Architekturmuster vorgestellt, die heutzutage am häufigsten benutzt werden.

## Model-View-Controller (MVC) und Model-View-Presenter (MVP)

### Grundstruktur

In MVC sowie in MVP bezeichnet man die Komponente der Datenhaltung als Model und die Komponente der Präsentation von Daten als View (Ackermann 2015). Die Grundstrukturen von MVC und MVP sehen Sie in Abbildung 2-1 und Abbildung 2-2.

Die zwei Architekturen sind recht ähnlich. Der einzige Unterschied liegt daran, dass bei MVP das View keinen Zugriff auf das Model hat (Also, Model und View komplett voneinander entkoppelt). Wobei in MVC die View-Komponente abhängig von der Model-Komponente ist und auf Änderungen am Daten (Model) reagieren bzw. anpassen kann (Ackermann 2015).

|  |  |
| --- | --- |
|  |  |
| Abbildung 2‑1: Grundstruktur MVC (Ackermann 2015, S. 13) | Abbildung 2‑2: Grundstruktur MVP (Ackermann 2015, S. 13) |

### Workflow und Beispiele

Ein typischer Workflow in MVC (s. Abb. 2-3 und Anhang 1):

* Benutzereingabe geht zum Controller
* Der Controller nimmt die Eingabe und gibt dem Model einen Befehl, damit das Model anhand der Benutzereingabe aktualisiert werden kann. Nachdem, dass die Daten im Model aktualisiert wurden, sagt der Controller dem View: bitte die neueste Daten vom Model darstellen.
* Das View holt die benötigten Daten aus Model (Zugriff auf das Model) und wird dann auch aktualisiert und bleibt damit aktuell und konsistent

|  |
| --- |
| \\pcpool.rz.rwth-aachen.de\files\home\mf247609\Desktop\t2kzD.png |
| Abbildung 2‑3: Workflow MVC (Bashir 2014) |

Ein typischer Workflow in MVP (s. Abb. 2-4): (hier code-Beispiel als Anhang 1 möglich, MVC\_MVP in MA\_code\_example)

* Benutzereingabe geht zum Presenter
* Der Presenter nimmt die Eingabe und gibt dem Model einen Befehl, damit das Model anhand der Benutzereingabe aktualisiert werden kann. Dann holt der Presenter die aktualisierten Daten aus Model und gibt sie weiter zum View
* Das View bekommt die aktuellen Daten vom Presenter und aktualisiert sich, damit bleibt das View aktuell und konsistent

|  |
| --- |
| \\pcpool.rz.rwth-aachen.de\files\home\mf247609\Desktop\adMfR.png |
| Abbildung 2‑4: Workflow MVP (Bashir 2014) |

### Fazit

Das Hauptziel von beiden ist die Entkoppelung von Model und View. Das MVP basiert auf dem MVC-Entwurfsmuster und ist eine weiterentwickelte Variante von MVC. Im Vergleich zum MVC ist MVP noch einen Schritt weiter gegangen, nämlich bei MVP ist die Verbindung zwischen Model und View nun komplett getrennt. Alle Kommunikationen müssen zunächst durch den dritte – den Presenter. Dies bietet einige Vorteile: z. B. so schafft man eine verbesserte Umgebung für die isolierte Prüfung jeder einzelnen Komponente (Besonders bei großem Projekt ist dies ziemlich wichtig) und eine bessere Wartbarkeit (maintainability).

## Model-View-ViewModel (MVVM)

MVVM lässt sich schwer mit "plain" JavaScript aufzubauen, weil es data-binding fordert. Die Entwickler, die noch keine Erfahrungen mit einem auf MVVM basiert JavaScript-Framework haben kann die Grundstruktur (s. Abb. 2-5) vermutlich sehr schwer vorstellen. Aber das Hauptziel immer noch das gleiche: Entkoppelung von Model und View, aber dieses Architekturmuster MVVM geht noch einen Schritt näher.

|  |
| --- |
|  |
| Abbildung 2‑5: Grundstruktur MVVM (Ackermann 2015, S. 16) |

Die zentrale Idee bei MVVM ist die bidirektionale Datenbindung zwischen ViewModel und View. Es wird für jedes View ein ViewModel bereitgestellt. Im ViewModel wird entsprechend Datenfelder für das View (UI-Elements) reserviert, sie dienen dazu, nach der Änderung der Datenfelder die entsprechenden UI-Elements zu aktualisieren und auch umgekehrt, nach der Änderung der UI-Elements (z. B. Benutzereingabe bei input-Element) die verbundenen Datenfelder in ViewModel zu aktualisieren. Das View hat auch die Möglichkeiten, durch command-binding (unidirektionales Command Binding) die im ViewModel gespeicherten Methoden (Funktionen) aufzurufen. (Ackermann 2015, S. 17) Zum Beispiel: Während ein Benutzer Text in das input-Element eingibt kann eine in ViewModel gespeicherter Validierungsfunktion ständig aufgerufen werden, um die Texteingabe zu validieren.

Durch das data-binding und command-binding besteht das View lediglich nur aus UI-Elementen enthält. Das bietet einen Vorteil: nämlich die Entwicklung von UI kann jetzt nahezu komplett getrennt entwickelt werden. D. h. einerseits, die Entwickler, die für UI zuständig sind, müssen nicht unbedingt die komplizierte JavaScript-Frameworks kennen, sondern nur HTML und CSS, andererseits, die Entwicklungen für UI und die hinter den UI stehenden Logik können gleichzeitig gemacht werden und somit wird die Entwicklungsperiode auch kürzer (Ackermann 2015, S. 17).

Das ViewModel in MVVM enthält im Gegensatz zu View fast alle Geschäftslogik (business logic) für die Darstellung der Daten und kommuniziert mit View durch die sogenannte bidirektionale Datenbindung (in English bekannt als "two-way data-binding"). Diese automatische Synchronisierung der Daten zwischen View und ViewModel ist ein ganz hervorragender Vorteil und deshalb benutzen auch so viele JavaScript-Frameworks MVVM Architekturmuster. Sie ist gleichzeitig auch der Hauptgrund und die Motivation, dass die Entwickler ein JavaScript-Framework benutzt (Gimeno 2018).

In Abbildung 2-6 werden die Quellcodes einer kleinen Beispiel-Anwendung dargestellt. Der Quellcode links ist mit „plain“ JavaScript geschrieben und der Quellcode rechts ist mit Hilfe eines JavaScript-Frameworks Vue programmiert. Dargestellt ist eine typische Situation, die bei jedem Projekt immer wieder auftauchen kann. Die Abbildung sollte nur einen Eindruck schaffen, dass die Codes von JavaScript-Framework deutlich kürzer, einfacher und gut organisiert aussehen und bei Codes von „plain“ JavaScript eher nicht so übersichtlich sind. Wenn man ins Detail geht, ist die Vorteile der Anwendung eines JavaScript-Framework nicht nur, dass die Codes insgesamt kürzer und einfacher sind, sondern auch sehr gut wiederverwendbar und von der Wartbarkeit erheblich besser als die „plain“ JavaScript Codes sind.

|  |
| --- |
|  |
| Abbildung 2‑6: plain JS vs. Vue (eigene Darstellung) |

Ein Hinweis für die Abbildung 2-6: Die oben dargestellte Abbildung soll nur einen Überblick schaffen. Das vollständige Beispiel mit „plain“ JavaScript sowie JavaScript-Framework Vue wird in Anhang 2 angeboten. Außer den HTML Dateien gibt es immer jeweils zwei JavaScript Dateien einmal mit Kommentar und auch ohne. Die Codes von dem Beispiel mit Vue sind teilweise spezifisch für das Vue Framework, wenn es Interesse besteht, wurde die folgende Seite empfohlen, mit der man einen Überblick auf dieses JavaScript-Framework schaffen kann (mehr muss man für dieses Arbeiten nicht unbedingt wissen):

* Vue introduction: https://vuejs.org/v2/guide/index.html

# Vergleich und Auswahl von JavaScript-Frameworks

## Allgemeine Vergleichung

### Popularität und Geschichte

Die Popularität ist einer den wichtigsten Gründen bei der Auswahl eines JavaScript-Framework. Ein weltweit von den Entwicklern viel benutzt JavaScript-Framework kann uns mindestens folgende Vorteile bieten:

* Gute Unterstützung von Community
* Weniger Fehler bzw. schon durch viele Entwickler geprüfte Belastbarkeit bzw. Stabilität des Frameworks
* Schnellere Aktualisierung und Fehlerbehebung des Frameworks

Außerdem, die Popularität eines JavaScript-Frameworks sagt uns direkt die Entscheidungen anderer Entwickler. Das zeigt uns einen guten Weg zum Einstieg in die Welt des JavaScript-Frameworks.

|  |
| --- |
|  |
| Abbildung 3‑1: Numbers of stars added on GitHub for Front-end Frameworks in 2017 (Quelle: https://risingstars.js.org/2017/en/#section-framework) |

In Abbildung 3-1 sieht man die Popularität der Frameworks im Jahr 2017, ausgewertet anhand der hinzugefügten Anzahl an Sternen bei GitHub. Vue, React und Angular sind die besten drei von 2017. Here noch ein bisschen Geschichte zu den drei erfolgreichen Frameworks:

* Vue: Vue wurde erstmals im Jahr 2014 von Ex-Google-Mitarbeiter Evan You veröffentlicht. Im Jahr 2016 kam die Version 2.0 und im gleichen Jahr bekam Vue eines der am schnellsten wachsenden JavaScript-Frameworks. Vue nennt sich „The Progressive JavaScript Framework“ und gehört zu MVVM.
* React: React wurde im Jahr 2013 von Facebook veröffentlicht. Die aktuellste Version von React ist die Version 16 (https://reactjs.org/versions). React nennt sich „a JavaScript library for building user interfaces“. Zwar nennt React sich Bibliothek anstatt Framework, das Prinzip von [Inversion of Control](https://en.wikipedia.org/wiki/Inversion_of_control) existiert bei React in der Bearbeitung von UI relevantem Teil.
* Angular: Angular.js bzw. Angular 1.x wurde im Jahr 2010 von Google veröffentlicht. Das neue Angular 2 kam aber im Jahr 2016. Die neueste Version von Angular ist momentan die Version 4. Angular ist bekannt als „complete solution“ und auch deswegen sehr beliebt bei großen Unternehmen.

|  |
| --- |
|  |
| Abbildung 3‑2: npm downloads of the top four front end frameworks at 14.07.2018 (Quelle: http://www.npmtrends.com/react-vs-vue-vs-angular-vs-@angular/core) |

In Abbildung 8 wird ein Diagramm dargestellt, es zeigt die Anzahl des Herunterladens der am meisten heruntergeladenen JavaScript-Frameworks bei npm in den letzten sechs Monaten. Kombiniert man die Auswertung bei GitHub und npm sowohl im Jahr 2017 als auch 2018, ist nicht schwer durchzuschauen, dass React, Vue und Angular die Welt der JavaScript-Frameworks dominieren.

### Lernkurve

Lernkurve ist auch ein entscheidender Faktor bei der Auswahl der JavaScript-Frameworks. Alle moderne JavaScript-Frameworks haben eigene Vor- und Nachteile. Wenn Zeit in der Entwicklungsphase keine Bedeutung hat, ist es grundsätzlich egal, welches zu wählen. Da am Ende alle modernen Frameworks ohne Probleme die Arbeit erledigen können. Aber in der Realität ist die Zeit immer ein bedeutender Faktor. D. h. diejenigen Frameworks, die super schwierig zu beherrschen aber wenn man es beherrscht, sehr leistungsstark sind, passen nicht unbedingt zu unserer Situation, die einen gemessenen Zeitdruck hat.

Wie kann man wissen, ob ein Framework relativ einfach bzw. schwierig zu lernen ist? Eine generelle Idee ist zu schauen, welche zusätzliche Kenntnisse (außer HTML, CSS, JavaScript) man noch braucht (Tab. 3-1).

Tabelle 3‑1: zusätzliche Kenntnisse für jeweilige JS-Frameworks (eigene Darstellung)

|  |  |
| --- | --- |
| **JS-Frameworks** | **Zusätzliche Kenntnisse benötigt** |
| React | **JSX** (Syntaxerweiterung zum JavaScript) zwar nur empfohlen (nicht pflichtet zu lernen), aber ohne JSX werden die Codes viel komplizierter und nicht anschaulich  (siehe auch: https://reactjs.org/docs/introducing-jsx.html) |
| Angular | **TypeScript** statt JavaScript wird angefordert. TypeScript kann man als eine verbesserte Version von JavaScript sehen. Für Entwickler, die schon Erfahrungen mit JavaScript haben, ist TypeScript nicht allzu schwer zu lernen. Aber Trotzdem, einige Änderungen (static typing, interface, arrow function, etc.) sind ein bisschen „tricky“ und dauert eine Weile bis man sie vollständig beherrscht.  (siehe auch: https://www.typescriptlang.org/docs/handbook/angular.html) |
| Vue | **Keine zusätzlichen Kenntnisse benötigt**  (siehe auch: https://vuejs.org/v2/guide/index.html) |

Außerdem könnte die folgende Abbildung (Abb. 9 von Maximilian Schwarzmüller), die die Lernkurve von React, Angular und Vue darstellt, auch einigermaßen hilfreich sein.

|  |
| --- |
|  |
| Abbildung 3‑3: Lernkurve Angular vs. React vs. Vue (Quelle: Schwarzmüller 2017) |

Ein Fazit ist nach allen Recherchen nicht schwer daraus zu ziehen. Vue ist ohne Zweifel das anfängerfreundlichste. React und Angular beide fordern neue Kenntnisse an, d. h. mehr Zeitaufwand für den Einstieg.

Es ist trotzdem zu empfehlen, die drei Frameworks selbst mal anzuschauen und eventuell die offiziellen Einführungen durchzugehen. Da für Entwickler, die unterschiedlichen Kenntnissehintergründe haben, die Schwierigkeit ganz anders sein könnte.

### Besonderheiten

// TODO

// Link: <https://hosting.1und1.de/digitalguide/websites/web-entwicklung/beliebte-javascript-frameworks-und-bibliotheken/>

## Praxisnahe Vergleichung: Implementierungsflexibilität

Ohne Rücksicht darauf, wie gut ein JavaScript-Framework von außen bewertet wurde, kommt es zuallerletzt zur praktischen Anwendung. Ein JavaScript-Framework unterscheidet sich sehr von einer JavaScript-Bibliothek bezüglich der „Inversion of Control“ (s. Kapitel 2.1). Eine Besonderheit bei EarlyDike ist, dass dieses Projekt schon sehr weit entwickelt wurde. Deshalb muss es auf jeden Fall berücksichtigt werden, ob die Möglichkeit besteht, ein JavaScript-Framework in ein existierendes Projekt zu implementieren.

Das ausgewählte JavaScript-Framework sollte im Prinzip in der Lage sein, sich an ein existierendes Projekt anpassen zu können. Obwohl ein kompletter Wiederaufbau auch eine Alternative ist und eventuell noch besser geeignet. Die Flexibilität der Implementierung ist aber immer noch ganz wichtig in der Praxis.

In der folgenden Tabelle werden die Implementierungsmöglichkeiten der jeweiligen JavaScript-Frameworks zusammengefasst.

Tabelle 3‑2: Implementierungsmöglichkeiten von JS-Frameworks

|  |  |
| --- | --- |
| **JS-Frameworks** | **Implementierungsmöglichkeiten** |
| React | * Direkte Einbindung durch Script-Element in HTML **möglich** * Für nur ein Teil eines Projektes **möglich**   (siehe auch: https://reactjs.org/docs/add-react-to-a-website.html) |
| Angular | * Direkte Einbindung durch Script-Element in HTML **unmöglich** (außer Angular.js, aber Angular ist hier gemeint ab Angular 2 oder noch höhere Version) * **Nur für ganzes Projekt implementierbar** (d. h. für existierendes Projekt entweder komplett neu aufbauen oder nicht implementierbar) |
| Vue | * Direkte Einbindung Script-Element in HTML **möglich** * Für nur ein Teil eines Projektes **möglich**   (siehe auch: https://vuejs.org/v2/guide/installation.html) |

Die beiden JavaScript-Frameworks Vue und React bieten mehr Flexibilität bei der Implementierung. Sie können laut offizieller Dokumentation (s. Tab. 3-2) sowohl in existierendes Projekt implementiert als auch für nur ein Teil des Projekts eingesetzt werden. Im Gegensatz dazu lässt das JavaScript-Framework Angular nicht in ein existierendes Projekt implementieren.

## Fazit

Kombiniert man die Faktoren aus Kapitel 3.1 und 3.2, ist es nicht schwer zu entscheiden, dass Vue das am besten geeigneten Framework für dieses Projekt ist. Der Entwickler von Vue hat mit Absicht den Einstieg für Anfänger viel leichter gemacht. Die Möglichkeit der Implementierung an existierendes Projekt bietet auch mehr Flexibilität bei der praktischen Anwendung.

Zwar hat React inzwischen mehr Popularität und deswegen deutlich mehr Unterstützungen von Community (auch React selber wird von Facebook regelmäßig verbessert), hat Vue seit 2016 auch eine Menge von Anwachsen und mittlerweile ein stabiles „Core Team“.

Eigentlich können die modernen JavaScript-Frameworks fast immer mehr als die Entwickler in der Praxis eigentlich brauchen. Die Auswahl ist nur der erste Schritt und im Vergleich zu den nachfolgenden vielleicht nicht so wichtig. Allein durch die Auswahl haben wir noch nichts für das Projekt gebracht. Nur im Laufe der Zeit und durch viele Versuche kann man die Tatsachen, die hinter den Frameworks verstecken, durchschauen und deshalb ist die Anwendung in der Praxis noch wichtiger. Im Kapitel 4 wird die Implementierung des ausgewählten JavaScript-Frameworks Vue ins Projekt EarlyDike Geoportal detailliert vorgestellt.

# Vorbereitung für die Implementierung des JavaScript-Frameworks Vue in das Projekt EarlyDike

## Angestrebte Ziele

Allgemein möchte man am Ende ein gut strukturiertes Projekt haben, das nicht nur für ihn selbst, sondern für andere Entwickler auch einfach zu verstehen ist und damit weniger Wartungsaufwand und Erweiterungsaufwand aufweist. Der Implementierungsprozess darf auch nicht viel zu lang dauern. Eine vernünftige Zeitspanne muss eingehalten werden. Außerdem ist es natürlich auch wichtig, die neusten Werkzeuge zu verwenden damit das Projekt zukunftsorientiert bleibt.

Projektspezifisch gibt es noch folgende Erwartungen:

1. Alle Parameter für die Erzeugung der Layers in einer bzw. mehrere Dateien umfassen und als statische Dateien auslagern. Damit sie nicht mehr im JavaScript-Codes gemischt bleiben.
2. Die Kernbibliothek OpenLayers von Version 3 auf die neuste Version 5 bringen. Einerseits um die zukünftige potentiale Kompatibilitätsproblem zu vermeiden, andererseits um die neuesten Features von OpenLayers 5, nämlich die Möglichkeit der Erstellung einer benutzerdefinierten Version der Bibliothek, anzuwenden (OpenLayers 2018a).
3. Die Benutzeroberfläche (UI) verbessern, damit einerseits die Funktionalitäten des Geoportals deutlicher und einfacher für die Benutzer zu verstehen sind und andererseits den Benutzern auch Möglichkeit geben, bestimmte Teile der Web-Anwendung während der Verwendung auszublenden und nur wichtige Informationen im Auge zu haben.
4. Wiederverwendbare Komponenten (reusable components) so viel wie möglich anfertigen. Dadurch andere bzw. zukünftige Projekte, die ähnliche Funktionalitäten haben, solche Komponenten wiederverwenden können und sodass haufenweise Arbeitsstunden gespart werden können.

Mit den oben zusammengefassten Zielen kann die Implementierung des JavaScript-Frameworks in EarlyDike Geoportal anfangen. Es bleibt nur, die geeignete Art der Implementierung zu finden und nach vielen Versuchen die richtigen Entscheidungen zu treffen.

## Hintergrund und Arten der Implementierung

### Hintergrund

Zuerst ist es klar zu machen, dass es großen Unterschied zwischen neuem Projekt und existierendem Projekt bei der Implementierung gibt.

JavaScript-Framework ist abstrakt betrachtet ein Rahmen und dieser Rahmen bietet viele Vorteile. Einer der wichtigsten davon ist nämlich der saubere, modulare Aufbau der Web-Anwendung. Völlig anders als eine JavaScript-Bibliothek, mit der die Entwickler während der Entwicklung keine Beschränkungen bzw. Hilfe haben und eventuell den sogenannten Spaghetticode schreiben könnten, bezwingt ein Framework den Entwickler, den Rahmen zu halten. Es ist schon sehr bekannt, dass bei der Anwendung einer Bibliothek wie jQuery viele Probleme geben könnte, allein wegen dem chaotischen Aufbau.

Es ist dann auch sinnvoll, ein JavaScript-Framework von Anfang an einzusetzen, um einen übersichtlichen, sauberen Aufbau zu erzielen. Es gibt JavaScript-Framework wie Angular (2 und höher), das sich nur sehr schwierig bei existierendem Projekt implementieren lässt. Aber es gibt auch JavaScript-Framework wie React und Vue, die sehr einfach bei existierendem Projekt implementiert werden können und dann nur für ein oder mehrere Teile des Projektes verantwortlich sind.

Es ist zwar immer zu empfehlen, ein JavaScript-Framework von Anfang an anzuwenden, aber in der Realität ist es einfach nicht so. Es gibt viele Projekte, die schon mit "plain" JavaScript geschrieben (mehr oder weniger auch mit externen JavaScript-Bibliotheken) und in Laufe der Zeit sehr kompliziert geworden sind. Solche Projekte haben mehr oder weniger die folgenden Eigenschaften:

* Unübersichtliche Struktur und deshalb sehr schwer, instand zu halten bzw. zu erweitern.
* Unnötige viele Codes, um einfache und oft benutzte Funktionalitäten zu programmieren (Das kann aber manchmal mit Hilfe von externen JavaScript-Bibliotheken gelöst werden). (plain JS vs. JS-Framework Beispiel, an Sync/data-binding)
* Viel zu viel dynamisch erzeugenden DOM-Elemente bei der JavaScript Seite, was die Instandhaltung noch schwieriger macht. Da man das View gar nicht vor Augen hat. (hier könnte Beispiele kommen, HTML-DOM und reale HTML Seite)

Die oben genannten Eigenschaften sind nicht selten bei einem real-world Projekt. Das EarlyDike Geoportal ist genau so ein Projekt. Viele Mitarbeiter haben daran gearbeitet und alles ist fast nur mit „plain“ JavaScript und ein paar externen JavaScript-Bibliotheken geschrieben. Die Struktur lässt sich nach langer Zeit kaum zu erkennen. Es gibt zahlreiche globale Variablen, die man nur sehr schwierig zuordnen kann. Zwar hat man über die Struktur der Web-Anwendung schon Gedanken gemacht und anstatt nur einer Haupt-JavaScript-Datei mehrere JavaScript-Dateien je nach Funktionalitäten erstellt, aber die Funktionen in jeder einzelnen JavaScript Datei sind meistens ungeordnet und beziehen sich manchmal sogar noch auf andere Dateien.

Bei großem Projekt ist es tatsächlich sehr aufwendig, eine klare Struktur bei der Entwicklung zu erhalten. Einerseits gibt es nicht nur einen Entwickler und andererseits ist nicht jeder Entwickler so erfahren, dass er sein eigenes "Framework" im Kopf hat.

Das bedeutet aber nicht, dass für solche Projekte keine Lösungen gibt. Im Kapitel 4.2.2 wird drei unterschiedliche Wege dargestellt, wie man JavaScript-Framework in solche Projekte implementieren kann.

### Arten der Implementierung

#### CDN Links

JavaScript-Framework kann auch ganz einfach wie eine JavaScript-Bibliothek durch die Einbindung eines Script-Elements in HTML implementiert werden. Der Quellcode 4-1 zeigt wie das JavaScript-Framework Vue mit Hilfe von CDN Links implementiert werden kann:

|  |
| --- |
| 1. <!DOCTYPE html**>** 2. **<html** lang="en"**>** 3. **<head>** 4. <!-- Vue.js - development version, includes helpful console warnings --> 5. **<script** src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"**></script>** 6. <!-- openlayers --> 7. **<script** src="./OpenLayers/OpenLayersDownload/v3.16.0/v3.16.0/build/ol.js" type="text/javascript"**></script>** 8. **<link** rel="stylesheet" href="./OpenLayers/OpenLayersDownload/v3.16.0/v3.16.0/css/ol.css" type="text/css"**>** 9. <!-- projection definition --> 10. **<script** src="./OpenLayers/OpenLayersDownload/proj4js-proj4js-2.2.3-3-gf6f3c00/proj4js-proj4js-f6f3c00/dist/proj4.js" type="text/javascript"**></script>** 11. **<script** src="GIAGS\_proj\_defs.js" type="text/javascript"**></script>** 12. <!--  jQuery einbinden --> 13. **<script** src="./jquery-3.3.1.min.js" type="text/javascript"**></script>** 14. **<title>**EarlyDike Geoportal**</title>** 15. **</head>** 16. **<html** lang="en"**>** |
| Quellcode 4‑1: Einbindung des JavaScript-Framework Vue durch CDN Link in HTML (eigene Darstellung) |

Ein offensichtlicher Vorteil davon ist: Die Implementierung kann sehr schnell durchgeführt werden. Diese Art von Implementierung ist besonders benutzerfreundlich. Da jeder Web-Entwickler die Einbindung des Script-Elements in HTML kennt und in nur einigen Sekunden kann man schon anfangen, mit Framework zu arbeiten.

Weitere Vorteile sind z. B.:

* Die existierenden Codes müssen nicht umgeschrieben werden und funktionieren genauso wie vorher.
* Neue Funktionalitäten können separat hinzugefügt werden und sie werden einwandfrei von JavaScript-Framework unterstützt.
* Einarbeitungsaufwand beschränkt sich nur auf das Framework, man muss nicht um andere Sachen kümmern.

Laut der offiziellen Dokumentation von Vue ist diese Art der Implementierung allerdings eigentlich nur geeignet, kleine bis mittlere Web-Anwendungen bzw. Web-Anwendungen, die nicht ausschließlich von JavaScript getrieben sind, aufzubauen (Vue 2018a). Auch für Anfänger, die JavaScript-Framework zuerst mal benutzen oder einfach nur einen Überblick haben wollen, ist diese Art von Implementierung auch ganz günstig.

Aber wenn man damit eine große bzw. komplett von JavaScript getriebene Web-Anwendung aufbauen bzw. existierende Web-Anwendungen verbessern möchte, ist es sehr unwahrscheinlich, dass man am Ende noch eine übersichtliche Struktur halten kann. Ohne die Unterstützung von moderner Web-Technologie wie Webpack, Browserify und ES6 ist es schwer, voran zu gehen (s. bitte auch Kapitel 4.5.1.2).

Übrigens, es besteht immer noch ein kleines Risiko: Wenn die Server von CDN offline sind, funktioniert die Web-Anwendung auch nicht mehr.

#### NPM

NPM ist ein command-line Werkzeug und wurde entwickelt für die Verwaltung der Dependencies der Web-Anwendungen. Dependencies (external code requirement, von Burgess definiert, s. Burgess 2018) sind z. B. Bibliotheken oder Frameworks, die meistens von anderen Entwicklern geschrieben sind und für eine Web-Anwendung benutzt werden. NPM verwaltet alle Dependencies für eine bestimmte Web-Anwendung.

Durch NPM kann man sehr schnell benötigte Dependencies finden und für seine Web-Anwendung einrichten. JavaScript-Frameworks gehören auch dazu (s. Quellcode 4-2).

|  |
| --- |
| 1. $ npm install vue 2. $ npm install -g @vue/cli |
| Quellcode 4‑2: Eirichtung von Dependencies durch NPM: Vue und Vue CLI 3 (eigene Darstellung) |

Die Implementierung des JavaScript-Frameworks durch NPM ist nichts anderes als ein automatisierter Vorgang. Im Kapitel 4.2.2.1 wird das gleiche aber manuell durchgeführt. Die Implementierung durch NPM bietet aber noch weitere Vorteile für die Entwicklung:

* Nicht nur das Framework, alle anderen Bibliotheken können durch NPM verwaltet werden. Z. B. für neue Entwicklungsumgebung können alle Dependencies auf einmal installiert werden (s. Quellcode 4-3).

|  |
| --- |
| 1. # Install the dependencies in the local node\_modules folder 2. $ npm install |
| Quellcode 4‑3: Einrichtung der Entwicklungsumgebung durch NPM (eigene Darstellung) |

* Alle benötigte Dependencies (mit Version) werden in package.json Datei ordentlich aufgelistet (s. Abb. 4-1). Dadurch ist die Entwicklungsumgebung einfacher zu verwalten und konfigurieren.

Außerdem arbeitet NPM mit modernen Web-Technologien wie Browserify, Webpack und begleitende Werkzeuge von Vue (s. bitte auch Kapitel 4.5.1.2) sehr gut zusammen, um in der Entwicklung einen guten Workflow zu schaffen (Vue 2018b).

Die Implementierung durch NPM hat aber ein Problem mit der Konfiguration von allen Werkzeugen, die für die Entwicklung gebraucht werden. Es kostet erheblich viel Zeit um zu verstehen und klar zu machen, wie man die Einstellungen richtig konfigurieren kann, damit alle moderne Web-Technologien wie Vue, Webpack, Browserify etc. überhaupt zusammenarbeiten. Die Zeit, die man für das „Set up“ investiert hat, ist gewissermaßen eine Verschwendung und wird auch nicht bei der Entwicklung weiterhelfen. Um dieses Problem zu lösen und gleichzeitig alle moderne Web-Technologien zu benutzen, wird das Werkzeug Vue CLI im nächsten Kapitel vorgestellt.

|  |
| --- |
|  |
| Abbildung 4‑1: Beispiel: Liste von Dependencies in package.json Datei (eigene Darstellung) |

#### CLI

Allgemein für die Entwicklung der Web-Anwendungen ist ein vernünftiger Workflow die Grundlegende Anforderung. Das bringt uns eine übersichtliche Struktur und damit wird der Aufwand der späteren Wartungsarbeit und Erweiterungsmöglichkeit erheblich verringert.

Die derzeitigen am beliebten JavaScript-Frameworks bieten alle eigene CLIs an (z. B.: Create React App, Angular CLI und Vue CLI). Die CLIs von allen JavaScript-Frameworks machen im Prinzip aber das Gleiche. Im Folgenden wird das CLI von Vue dargestellt. Da wir uns für das Projekt EarlyDike für Vue entschieden haben.

Das JavaScript-Framework Vue bietet den Entwicklern ein begleitendes Werkzeug Vue CLI, das für Vue entwickelt wurde. Das Vue CLI kann einfach durch NPM (s. Quellcode 4-2) ganz normal wie andere Dependencies installiert werden und bietet die Möglichkeiten, eine vordefinierte Struktur für ein Projekt zu erstellen (s. Abb. 4-2).

Die vordefinierte Struktur erleichtert die Einarbeitungsaufwand am Anfang hervorragend und hat einigermaßen die Regeln für die Entwicklung des Projekts gesetzt. Außerdem bringt Vue CLI die heutzutage am häufigsten benutzt modernen Web-Technologien z. B.: Webpack und Babel mit. Der Entwickler hat die Möglichkeit, bei der Initialisierung des Projekts die erwünschte Web-Technologien auszuwählen und Vue CLI konfiguriert automatisch alle notwendigen Einstellungen für das erstellte Projekt.

Durch die Initialisierung eines Projekts mit Hilfe von Vue CLI (s. Kapitel 6.1) hat das Problem der Konfiguration aller modernen Web-Technologien sich erledigt. Man muss sich nicht mehr um die Einstellungen von unterschiedlichen Web-Technologien kümmern und kann einfach auf das Projekt konzentrieren.

Bisher scheint die Implementierung durch Vue CLI die beste Lösung zu sein. Sicherlich hat diese Art von Implementierung des JavaScript-Frameworks auch Nachteile. Es ist nicht so günstig für existierende Projekte, die schon seit langem entwickelt wurden. Die notwendige Anpassung in die neue Struktur sowie die Anwendung von besonderen Dateiformaten (z. B. in Vue stellt man SFC vor), die je nach Frameworks unterschiedlich sind, und auch andere Schwierigkeiten machen dieser Weg nur schwierig für existierende Projekte. Die müssen meistens großartig umgeschrieben sogar komplett neu geschrieben werden.

|  |
| --- |
|  |
| Abbildung 4‑2: Vordefinierte Struktur von neu erstelltem Projekt mit extra Anforderungen an TypeScript- und Vuex-Unterstützung (eigene Darstellung) |

## Entscheidung der Art der Implementierung

Die Tabelle 4-1 bewertet die im Kapitel 4.2.2 vorgestellten Arten der Implementierung nach dem Erfüllungsgrad der im Kapitel 4.1 genannten angestrebten Ziele. Das hier angewendete Notensystem richtet sich nach der deutschen Schulnote.

Tabelle 4‑1: Erfüllungsgrad der angestrebten Ziele von unterschiedlichen Arten der Implementierung (eigene Darstellung)

|  | CDN | NPM (nur mit JavaScript-Framework) | NPM (mit neuer Web-Technologien | CLI |
| --- | --- | --- | --- | --- |
| Allgemeine Ziele |  |  |  |  |
| gesch. Wartungsaufwand nach Wiederaufbau | 5 | 5 | 1 | 1 |
| gesch. Erweiterungsmöglichkeit nach Wiederaufbau | 3 | 3 | 1 | 1 |
| Zeitkosten | 1 | 1 | 5 | 4 |
| Angewendete Web-Technologien | 3 | 3 | 2 | 1 |
| Projektspezifische Ziele |  |  |  |  |
| Auslagerung der Parameter von Layers | 1 | 1 | 1 | 1 |
| OpenLayers 3 auf OpenLayers 5 | 5 | 5 | 1 | 1 |
| Verbesserung von UI | 3 | 3 | 2 | 1 |
| Wiederverwendbare Komponenten | 4 | 4 | 2 | 1 |
| INSGESAMT | **3,1** | **3,1** | **1,9** | **1,4** |

Die aus den Bewertungen abgeleiteten Noten weisen den Sieger eindeutig hin. Die Implementierung des JavaScript-Frameworks durch CLI kann alle Ziele sehr gut erfüllen. Der einzige Nachteil ist die Kosten der Zeit, was aber im Vergleich zu den Schwachstellen der anderen zwei eigentlich akzeptabel ist.

Die Implementierung durch CDN sowie NPM (ohne Unterstützung von anderen Web-Technologien, nur das Framework) haben zwei Hauptprobleme. Einerseits ist eine übersichtliche Struktur immer noch schwer zu schaffen und das erhört die spätere Wartungsaufwand sowie Erweiterungsaufwand, andererseits ist heutzutage ohne Unterstützung von Web-Technologien wie Parcel, Webpack sehr unpraktisch, ES6 basiert OpenLayers 5 anzuwenden (OpenLayers 2018b). Trotzdem, die Zeitkosten sind wesentlich weniger und die anderen Ziele sind zwar nicht perfekt aber doch ausreichend erfüllt. In der Praxis ist es meisten der Fall: Das Projekt wurde schon veröffentlicht und ständige Verbesserungen müssen im Laufe der Zeit durchgeführt werden. Es gibt nur sehr beschränkte Zeit für die Implementierung. Für solchen Fällen sind die Zeitkosten der entscheidende Faktor.

Die Implementierung durch NPM mit allen nötigen Web-Technologien haben im Prinzip bei der Entwicklung des Projekts die gleiche Kraft wie die Implementierung durch CLI. Aber CLI hat einen großen Vorteil im Vergleich dazu, nämlich die automatische Einstellung der komplexen Konfiguration von allen Web-Technologien. Allein deswegen gibt es keinen Grund mehr, mit NPM selbst alles zu konfigurieren. Mit Hilfe von CLI wird alles schon startbreit sein.

In den nachfolgenden Kapiteln werden zwei Versuche durchgeführt. Die beiden Variante: Implementierung durch CDN und Implementierung durch CLI.

# Versuch 1: Implementierung durch CDN (s. Anhang 4)

## Grundprinzip: Migration

In den meisten Szenarien ist ein kompletter Wiederaufbau einer Web-Anwendung allein wegen der enormen Zeitkosten nicht möglich. Die Implementierung durch CDN ist nicht nur bei der Einbindung des Script-Elements in HTML schnell, sondern auch bei der Entwicklung deutlich schneller, da die alten Codes nicht großartige für das neue Framework angepasst werden müssen.

Die zentrale Idee ist eine Migration von alten Codes. D. h. nur die minimalen, unumgänglichen Änderungen zu den existierenden Codes machen und alternative Änderungen, die möglicherweise viel Zeit kosten könnten, verzichten. Dadurch wird viel Zeit gespart und gleichzeitig die Implementierung des JavaScript-Framework durchgeführt.

Die Migration bedeutet an dieser Stelle aber nicht, dass die Vorteile des JavaScript-Framework verzichtet werden. Im Gegensatz dazu, wird es versucht, mit dem minimalen Arbeiten eine relative gute Struktur durch Framework zu schaffen. In folgenden Kapiteln wird es vorgestellt, wie das JavaScript-Framework Vue in der Praxis angewendet wird.

## Grundstruktur des Projekts EarlyDike

Je nach den Funktionalitäten soll die Grundstruktur (s. Abb. 5-1) erstellt werden. Mit der Hilfe von Vue Komponente (s. Kapitel 5.3.2) können wir relativ einfach eine übersichtliche Struktur erstellen.

|  |
| --- |
|  |
| Abbildung 5‑1: Grundstruktur des Projekts EarlyDike - CDN Version (eigene Darstellung) |

Das Framework Vue ist teilweise von dem Architekturmuster MVVM inspiriert (Vue 2018c) und die gesamte Struktur von der durch Vue entwickelt Web-Anwendung kann grundsätzlich auch als MVVM interpretiert werden. Zum Beispiel: Die Komponenten, die einer Funktionalität bzw. mehreren eng verbundenen Funktionalitäten in EarlyDike Geoportal entspricht, können als ViewModel gesehen werden. In den JSON-Dateien werden die Parameter für WMS sowie für WFS Layers gespeichert, was dann zu Model gehört. Das echte DOM sowie alle zugehörige CSS Dateien sind das View, obwohl durch die Anwendung von Virtual DOM das echte DOM kaum noch direkt von den Entwicklern manipuliert wird.

## Technische Details der Migration

### Allgemeine Hinweise

Die Implementierung von JavaScript-Framework Vue durch CDN ist zwar der schnellste Weg um mit Vue anfangen zu arbeiten. Aber wegen der fehlenden Unterstützungen von anderen modernen Web‑Technologien kann an dieser Stelle die volle Kraft von Vue noch nicht benutzt werden. In diesem Kapitel liegt der Fokus, wie im Kapitel 5.1 schon genannt wurde, auf die Migration. Je weniger Zeit man für die Migration braucht und je weniger Änderungen vorgenommen werden, desto besser. Die folgenden Kapitel konzentrieren sich darauf, die minimalen bzw. obligatorischen Änderungen zu entdecken und die dafür gebrauchten Techniken darzustellen. Solche technischen Details schaffen gleichzeitig genau die Basis für das JavaScript-Framework Vue. Deswegen ist die Implementierung durch CDN besonders Anfängerfreundlich und am besten geeignet um Vue etwas zu verstehen.

### Ordentlicher Aufbau mit Vue Components

#### Von Web-Anwendung zu vernetzten Komponenten - Grundstruktur

Eine Web-Anwendung kann immer in mehrere Teile geteilt werden (meistens nach unterschiedlichen Funktionalitäten). Durch die Anwendung von JavaScript-Framework Vue werden die Entwickler ganz am Anfang angefordert, für jeden Teil der Web-Anwendung einzelne Komponenten (Vue Components) aufzubauen und ordentlich zuzuordnen. Für eine Komplexe Funktionalität ist es natürlich auch sinnvoll, eine Komponente wiederum in mehrere Child-Komponenten zu teilen. Dies kann so oft wiederholt werden. In Abbildung 5-2 wird dieser Vorgang abstrakt dargestellt.

|  |
| --- |
|  |
| Abbildung 5‑2: Von Web-Anwendung zu vernetzten Komponenten  (Quelle: https://vuejs.org/v2/guide/components.html#Organizing-Components) |

#### Die Zusammensetzung eines Vue Component

Das JavaScript-Framework Vue beschäftigt sich hauptsächlich mit dem ViewModel im MVVM Architekturmuster. Wie ein Vue Component aufgebaut wird ist deshalb ein Basiswissen, wenn eine Web-Anwendung mit Vue entwickelt werden soll.

In Vue gibt es zwei Typen von Komponenten: Globale und Lokale. Da für diese Implementierung hier kein modulares Aufbausystem mit der Unterstützung von „module bundler“ wie z. B. Webpack benutzt wird, werden alle Komponenten einfach global registriert. Die Registrierung wird in dem sogenannten „global scope“ durchgeführt (s. Quellcode 5-1).

Ein Vue Component kann als ein ViewModel gesehen werden (s. Kapitel 5.2). Ein sogenanntes Optionsobjekt (options object) wird in die Komponente übergeben, nachdem ein Vue Component erstellt wurde. Ein Optionsobjekt setzt sich aus mehreren Feldern wie z. B.: data, computed, watch, methods und lifecycle hooks (s. Quellcode 5‑1). Das Feld data entspricht den bereitgestellten Datenfelder in ViewModel (s. Kapitel 2.3). Das Feld methods enthält alle JavaScript Funktionen. Das Feld template ist zuständig, um das Virtual DOM zu erstellen und zusammen mit anderen Feldern später das echte DOM zu manipulieren. Wie die Felder in einer Komponente zusammenarbeiten werden in den folgenden Kapiteln jedoch nur konzeptionell aber nicht detailliert erklärt. Eine vollständige Dokumentation ist bei der offiziellen Homepage von Vue zu finden. Es ist auf jeden Fall zu empfehlen, zuerst ein paar Tage die Einführung von Vue in Ruhe durchzulesen.

|  |
| --- |
| 1. **var** exampleGlobalComponent = Vue.component('example-global-component', { 2. props:['exampleprops'], 3. data: **function**() { 4. **return** { 5. // ... reserved place to store datas 6. } 7. }, 8. methods: { 9. // ... JS functions 10. }, 11. created: **function**() { 12. // ... lifecycle hook, the function here will be excuted 13. // after the instance is created 14. }, 15. computed: { 16. // ... for computed properties 17. }, 18. template: ` 19. // ... vue template syntax / Virtual DOM 20. ` 21. }); |
| Quellcode 5‑1: Registrierung globaler Komponente in Vue (eigene Darstellung) |

### Template Syntax von Vue

#### Virtual DOM

Um ein dynamisches UI zu erzeugen muss das DOM ständig manipuliert werden. Die in Quellcode 5-2 dargestellte Methode ist einer davon, die oft benutzt werden, um direkt das DOM zu manipulieren. In Vue wird jedoch Virtual DOM benutzt. D. h. die Entwickler manipulieren nicht mehr das echte DOM direkt. Stattdessen wird das von Vue vorgestellte Template manipuliert. Das JavaScript-Framework wird später zuerst das Template kompilieren und dann das echte DOM manipulieren.

|  |
| --- |
| 1. // change the text content of a <div> with id of "demo" 2. document.getElementById("demo").innerHTML = "Hello World!"; 3. // create a <p> tag 4. **var** pEle = document.createElement('p'); 5. // removes a specified child element from the HTML element that calls this method 6. ele.removeChild(childEle) |
| Quellcode 5‑2: Direkte DOM-Manipulationsmethoden |

Es gibt hauptsächlich zwei Vorteile:

* 1. Allgemein ist die Anwendung von Virtual DOM viel leistungsstärker als direkte DOM-Manipulation (s. Kapitel 1.3).
  2. Die direkte DOM-Manipulationsmethode könnte sehr verwirrend sein, obwohl es keine großartigen Änderungen bei dem UI besteht. Vue bietet den Entwicklern eine auf HTML basierte Template Syntax an, mit der die Manipulation von Virtual DOM sehr übersichtlicher und einfacher wird. Im Quellcode 5-3 wird das Feld Template von einem Vue Instance dargestellt.

|  |
| --- |
| 1. template: ` 2. **<li** class="singleLayer"**>** 3. <!-- Rectangular switch --> 4. **<label** class="switch"**>** 5. **<input** 6. type="checkbox" 7. v-model="layerSwitch" 8. @change="WMSLayerSwitch" 9. **/>** 10. **<span** class="slider"**></span>** 11. **</label>** 12. **<span** style="cursor: pointer" @click="showOpacity = !showOpacity"**>** 13. {{ singlelayerinfo.layers }} 14. **</span** 15. **</li>** 16. ` |
| Quellcode 5‑3: Template von Vue Instance / Component (eigene Darstellung) |

#### Interpolation (one-way data-binding) und Form Input Bindings (two-way data-binding)

Die sogenannte Interpolation (one-way data-binding) hilft dabei, das View anhand der in Vue instance (ViewModel) gespeicherten Daten zu aktualisieren. Ohne die Anwendung von JavaScript-Framework wird häufig die direkte DOM-Manipulation durchgeführt. Aber in Vue wird dies durch one-way data-binding und Virtual DOM realisiert. In Anhang 3 ist ein Code-Beispiel, das die zwei Methoden darstellt.

Das Form Input Binding (two-way data-binding) ist für die drei HTML-Elemente: input, textarea und select. Damit die in Vue instance gespeicherten Daten auch anhand der Benutzereingabe aktualisiert und für weitere Anwendung zur Verfügung dargestellt werden. In Anhang 4 ist ein Code-Beispiel, das das two-way data-binding darstellt.

#### Directives

Directives sind besondere Attribute im Template von Vue instance. Directives haben mehrere Funktionen wie z. B.: bedingte Darstellung von View, reaktive Aktualisierung von HTML-Attributen etc. Im Quellcode 5-3 ist das Zeichen „@“ ein „shorthand“ für das „v-on“ Directive und „v‑model“ ist auch ein Directive für bidirektionale Datenbindung.

### Obligatorische Änderungen bei der Migration

Der Aufbau durch Komponenten schafft eine Herausforderung für die Migration. Die Anwendung von Template (also Virtual DOM) in der Komponente ändert komplett die Art und Weise, wie dynamische UI‑Elemente erzeugt werden. Außerdem, durch Interpolation, Form Input Bindings und Directives ist das Abrufen von Attributwerten der echten DOM-Elemente nicht mehr nötig, wobei alle benötigten Daten schon im Feld data gespeichert sind und jederzeit abgerufen bzw. geändert werden können.

Die Regeln für den Aufbau von einer Komponente sind der Rahmen, den das Framework Vue den Entwicklern anbietet. Jedes einzelne Feld in der Komponente hat eigene Aufgabe. Ein vernünftiger Aufbau der Komponente mit minimalen Änderungen der alten Codes ist der Schwerpunkt der Migration.

Die meisten Teile der Migration folgen nach den folgenden Schritten (s. Anhang 3 und auch Anhang 2):

* + 1. Alle globalen Variablen müssen entweder direkt in Root Vue Instance im Feld data gespeichert oder als JSON-Datei ausgelagert werden.
    2. Die alten direkten DOM‑Manipulationsmethoden, durch die neue HTML-Elemente eingelegt werden, müssen jetzt für das Feld template konvertiert werden.
    3. Das Abrufen bzw. das Aktualisieren von Attributwerten der DOM-Elemente durch direkte DOM‑Manipulationsmethode werden nicht mehr gebraucht, stattdessen werden die relevanten Attributwerten durch data-binding im Feld data gespeichert.

## Fazit

Der größte Vorteil der Implementierung durch CDN ist die relativ kurze Bearbeitungszeit, die nur 42 Stunden beträgt (Die Lernzeit für das Framework Vue jedoch nicht enthalten).

Äußerlich kommt jetzt CSS-Grid zusätzlich für die Benutzeroberfläche zum Einsatz. Aber eine deutliche optische Verbesserung gibt es leider nicht (s. Abb. 5‑3), nur in HTML ist die Struktur viel übersichtlicher durch die Anwendung von CSS-Grid.

Innerlich werden alle Funktionalitäten durch Komponente aufgebaut und global registriert. Zwar hat man jetzt einen viel besseren Überblick auf Grundstruktur und dynamische Erzeugung der UI-Elemente jedoch sind die meisten JavaScript Funktionen nur angepasst übernommen. D. h. in jeder einzelnen Komponente könnte es immer noch viel zu ungeordnet sein. Da die übernommenen Funktionen nicht objektorientiert sind. Also für die existierten Codes ist die Wartungsaufwand immer noch nicht leicht, aber die Basis für die zukünftige Erweiterung wurde schon mit Vue geschafft.

Übrigens, die Aktualisierung der OpenLayers von derzeitiger Version 3 auf Version 5 ist hier zwecklos. Da das wichtigste neue Feature von OpenLayers 5, nämlich die Erstellung der benutzerdefinierten Bibliothek eine auf ES6 Modules basierte Implementierung anfordert, d. h. eine Unterstützung von „module bundler“ (z. B.: Webpack) ist erforderlich.

Der erste Versuch ist zwar nicht ohne Erfolg, aber die angestrebten Ziele sind nicht gut erfüllt. Eine moderne Web-Anwendung muss sowohl äußerlich schön als auch innerlich gut strukturiert sein. Im nächsten Kapitel (also bei dem zweiten Versuch) wird zur optischen Verbesserung der Benutzeroberfläche das Vuetify Plugin und zum noch besseren Management der Inhalte und der Struktur das Vuex Plugin zum Einsatz kommen. Außerdem wird OpenLayers auf Version 5 aufgebracht, dadurch nur die benötigen Teile von OpenLayers Bibliothek genommen werden und damit eine neue benutzerdefinierte Bibliothek erzeugt wird.

|  |
| --- |
|  |
| Abbildung 5‑3: Benutzeroberfläche der EarlyDike Geoportal (CDN Version) |

# Versuch 2: Implementierung durch CLI (s. Anhang 5)

## Grundprinzip: Wiederaufbau

Im ersten Versuch wird eine schnelle Lösung vorgestellt, wobei die meisten alten Codes angepasst in den neuen Rahmen übernommen werden können. Dieser Versuch soll aber zeigen, wie man das alte Projekt zerlegt und mit Hilfe der modernen Web-Technologien ganz vom Anfang wiederaufbauen kann. Der Zeitaufwand dafür ist deutlich höher, aber gleichzeitig können auch mehr Ziele, die für dieses Projekt gestellt werden, erfüllt werden (s. Tab. 4‑1).

Im Gegensatz zu dem ersten Versuch ist das Grundprinzip in diesem Kapitel nicht mehr die Migration der alten Codes, sondern ein kompletter Wiederaufbau. Der Wiederaufbau soll am Ende alle angestrebten Ziele im Kapitel 4.1 erfüllen. Um dies zu verwirklichen, werden die derzeitig meistens benutzt modernen Web-Technologien eingesetzt. Dieser Versuch dient gleichzeitig auch zu der Erforschung der neusten Web-Entwicklungstechnologien.

In diesem Kapitel werden jegliche Details bei der Anwendung neuer Web-Technologien vorgestellt. Es soll ein Eindruck geschafft werden, wie eine Web-Anwendung heutzutage mit Hilfe von JavaScript-Framework entwickelt wird.

## Initialisierung des Projekts EarlyDike Geoportal mit der Hilfe von Vue CLI 3

Die Initialisierung des Projekts scheint ganz einfach zu sein, jedoch schafft sie im Hintergrund eine sehr wichtige Basis für die Entwicklung: Nämlich neben der vordefinierten Struktur (s. K 4.2.2) erledigt das Vue CLI bei der Initialisierung des Projekts alle Konfigurierungen von z. B. Webpack, Babel und auch andere ausgewählten Plugins automatisch. Dadurch kann der Entwickler sich mehr auf die Entwicklung konzentrieren und die Zeit für die komplizierten Konfigurierungen sparen (Gore 2018). Nachfolgend wird der Initialisierungsablauf kurz vorgestellt.

Mit dem folgenden Befehl (s. Quellcode 6‑1) fängt Vue CLI an, ein neues Projekt mit dem Namen „earlydike-vue“ zu erstellen.

|  |
| --- |
| 1. $ vue create earlydike-vue |
| Quellcode 6‑1: Initialisierung eines neuen Projekts mit Vue CLI (eigene Darstellung) |

Dann wird gefragt, ob die Standardeinstellungen angewendet werden sollen. Da das EarlyDike Geoportal Projekt das Plugin Vuex später braucht, kann das Plugin bei der Initialisierung schon hinzufügt werden. Außerdem werden Babel und Linter für die ES6 Syntax Transformation und Fehlererkennung der Codes als Standard ausgewählt (s. Abb. 6‑1).

Es werden danach noch nach weiteren Einstellungen gefragt. Die sind aber für dieses Projekt nicht relevant und daher wurden direkt die Standardeinstellungen genommen. Sobald alle Fragen beantwortet werden, fängt das Vue CLI an, eine Vordefinierte Struktur zu erzeugen und alle Konfigurierungen automatisch zu stellen.

|  |
| --- |
|  |
| Abbildung 6‑1: Ofizielle Plugins von Vue CLI (eigene Darstellung) |

Nach der Initialisierung müssen nur noch die OpenLayers Bibliothek sowie das Vuetify Plugin manuell durch NPM installiert werden. Die folgende kurze Vorstellung zeigt, wofür die alle Plugins bzw. Bibliothek gebraucht werden, und sie zusammen schafft die Grundlage für die Entwicklung.

* OpenLayers 5: JavaScript Bibliothek für die Darstellung verteilter Geoinformationen aus unterschiedlichen Quellen.
* Vuex: Plugin für das JavaScript-Framework Vue, dient zum zentralen State Management.
* Vuetify: Plugin für das JavaScript-Framework Vue, bietet UI-Elemente an, die nach Google Material Design angefertigt werden.

Mit Hilfe von Vue CLI kann ein neues Projekt in wenigen Minuten initialisiert und gleichzeitig eine moderne Entwicklungsumgebung dafür geschafft werden. In dem nächsten Kapitel wird die Grundstruktur der EarlyDike Geoportal Web-Anwendung dargestellt.

## Grundstruktur des Projekts EarlyDike

Im Vergleich zum Kapitel fünf hat sich die Grundstruktur erheblich geändert (s. Anhang 6). Dank der automatischen Konfigurierungen durch Vue CLI können hier das spezielle Dateiformat mit der Endung „\*.vue“ benutzt werden. Dieses Dateiformat nennt man Single‑File Components (SFC). Jedes SFC ist im Prinzip ein lokal registriertes Vue Component, viel anders als das global registriertes Vue Component im Kapitel fünf ist es nicht. Nur hat Vue jetzt das Feld template separate aufgeführt, damit die Entwickler bei der Bearbeitung auch Syntax-Highlighting für template haben können. Außerdem wird CSS auch in das SFC integriert. D. h. bei der Entwicklung werden fast nur noch SFC erstellt, aber das JavaScript‑Framework Vue im Hintergrund kompiliert alle SFC zu HTML, CSS und JS, damit später der Browser die Web-Anwendung richtig darstellen kann. Die Struktur von einem SFC wird im Quellcode 6‑2 dargestellt. Im Quellcode 5‑1 steht ein global registriertes Vue Component zur Vergleichung.

|  |
| --- |
| 1. **<template>** 2. <!-- ... vue template syntax / Virtual DOM  --> 3. **<div>** 4. **<p>**this is a example SFC file**</p>** 5. **</div>** 6. **</template>** 7. <script> 8. **import** exampleComponent from './exampleComponent' 10. **export** **default** { 11. name: 'exampleComponent', 12. components: { 13. exampleComponent  // locally register another Component inside this Component 14. }, 15. data () { 16. **return** { 17. // reserved place to store data 18. } 19. }, 20. methods: { 21. // ... functions 22. }, 23. created () { 24. // ... lifecycle hook, the function here will be excuted 25. // after the instance is created 26. }, 27. computed: { 28. // ... for computed properties 29. } 30. } 31. </script> 32. <style scoped> 33. // with "scoped" attribute means the CSS rules are only for this SFC 34. // normal CSS rules here 35. </style> |
| Quellcode 6‑2: Struktur eines Single‑File Component |

Durch das spezielle Dateiformat SFC hat Vue eigentlich das View und ViewModel kombiniert. Das ist auch der Grund, dass bei der Grundstruktur ein MVVM Architekturmuster kaum zu erkennen ist. Diese Kombination bietet einen Vorteil: Jedes SFC ist für sich allein zuständig und der Wartungsaufwand wird dadurch erleichtert. Es hat eigenes template (also das Virtual‑DOM, später wird das View dadurch erzeugt), eigene CSS‑Regeln und ein vollständiges Vue Component. Innerhalb einer SFC Datei ist ein quasi geschlossener Kreis, in dem alle relevanten Codes nebeneinanderstehen (Vue 2018a).

In den folgenden Kapiteln werden einige sehr wichtigen technischen Details vorgestellt. Es gibt sowohl Details in globaler Ebene: z. B. die Anwendung von Vuex, mit dem alle Komponenten vernünftig miteinander arbeiten, aber auch Details in einzelner Komponente: z. B. das Konstruktionskonzept von unterschiedlichen Komponenten.

## Technische Details

### Details in globaler Ebene

#### Von OpenLayers 3 bis OpenLayers 5

Die JavaScript Bibliothek OpenLayers ist der Kern zur Darstellung verteilter Geoinformationen. In dem alten EarlyDike Geoportal wurde jahrelang immer die Version 3.16.0 verwendet. Die tatsächlichen Funktionalitäten von der Version 3 bis zu der Version 5 haben sich eigentlich nicht viel geändert. Ein sehr wichtiger Grund und gleichzeitig auch eine Motivation dafür, die Bibliothek nun auf Version 5 zu bringen liegt daran, dass ab Version 5 die Bibliothek auch auf ES6 Modules basierte Implementierung (OpenLayers 2018c) unterstützt. Das hat eine große Bedeutung für die Leistung bei dem Öffnen einer Web-Anwendung.

In dem alten EarlyDike Geoportal sowie in der CDN Version wird die ganze gezippte OpenLayers Bibliothek einfach auf dem Server gespeichert. Jedes Mal ein neuer Benutzer die Web-Anwendung öffnen möchtet, muss zuerst die ganze gezippte OpenLayers Bibliothek heruntergeladen werden, obwohl für die Web-Anwendung nur ein kleines Teil der Bibliothek benutzt wird. Das ist nämlich eine große Leistungsverschwendung.

Dank der automatischen Konfigurierung von Vue CLI kann die OpenLayers 5 einwandfrei und auf ES6 Modules basiert implementiert werden (s. Quellcode 6‑3). Dadurch werden nur noch die benötigten Teile von der OpenLayers Bibliothek in die endgültige Web-Anwendung mitgenommen.

|  |
| --- |
| 1. **import** Map from 'ol/Map'; 2. **import** View from 'ol/View'; 3. **import** Attribution from 'ol/control/Attribution'; 4. **import** Zoom from 'ol/control/Zoom'; 5. **import** SelectInteraction from 'ol/interaction/Select'; 6. **import** LayerGroup from 'ol/layer/Group'; 7. **import** Rotate from 'ol/control/Rotate'; 8. **import** { fromLonLat, transform } from 'ol/proj.js'; 10. // select interaction for the WFS features 11. **import** Select from 'ol/interaction/Select'; 12. **import** { click, pointerMove } from 'ol/events/condition'; |
| Quellcode 6‑3: Auf ES6 Modules basierte Implementierung von OpenLayers 5 |

Die Version 3.16.0 von OpenLayers an sich besitzt eine Größe von fast 0,5 MB (s. Anhang 6). Im Vergleich dazu, in der CLI Version, eine fertig gebaute Web-Anwendung hat am Ende alle benötigten Dependencies in einer JavaScript Datei gespeichert, in diesem Fall sind das Framework Vue, das Plugin Vuetify, das Plugin Vuex und die OpenLayers 5 dabei. All dies zusammen hat eine Größe von nur 1,0 MB (s. Anhang 6). Leider ist eine Trennung dieser endgültige JavaScript-Datei nicht möglich. D. h. der Teil des OpenLayers von dieser zusammengestellte JavaScript‑Datei kann an dieser Stelle nicht bestimmt werden. Aber das Prinzip hier ist klar: Nur die tatsächlichen angewendete Teile in das endgültige Produkt nehmen und alle anderen verwerfen.

#### Centralized State Management mit Vuex

Eine Web-Anwendung stellt immer Daten in unterschiedlicher Art und Weise dar, und ist meistens in der Lage, mit dem Benutzer zu interagieren. So ist es auch im EarlyDike Geoportal: Das Geoportal stellt am Anfang nur einen Standardlayer dar und bietet dem Benutzer sämtliche Möglichkeiten, durch die der Benutzer mit dem Geoportal interagieren kann. Der Benutzer kann z. B. selbst entscheiden, welcher Layer in welcher Opazität dargestellt werden soll, welche WFS Service eingeschaltet werden soll und welche Informationstab angezeigt werden soll etc.

Für alle solche Interaktionen wie Ein-/Ausschalten des Layers, Opazitätsänderung sowie für alle dargestellten Informationen, stehen änderbare Zustandsobjekte dahinter. Solche Zustandsobjekte werden offen in Englisch „State“ genannt.

Die folgenden zwei einfache Beispiele sollen zuerst einen Eindruck schaffen, was ein Zustandsobjekt bzw. „State“ ist:

1. Das Map-Objekt von OpenLayers enthaltet alle Daten über die im Geoportal dargestellte Karte. Jede Opazitätsänderung, die durch eine Komponente ausgelöst wird, wird am Ende eine Mutation des Map-Objekts sein. In diesem Fall ist das Map-Objekt ein Zustandsobjekt, die durch viele Child‑Komponenten mutiert werden können.
2. Wenn der Benutzer einen Punkt aus einem WFS‑Layer ausgewählt hat, wird die entsprechende Daten dafür zuerst von anderer Quelle abgerufen und filtriert. Als nächstes wird ein Datenobjekt gebaut, damit die filtrierten Daten im WFS‑Informationstab dargestellt werden können. Das Datenobjekt ist in diesem Fall auch ein Zustandsobjekt. Wenn der Benutzer später einen anderen Punkt wählt, müssen die dargestellten Daten aktualisiert werden, d. h. das Datenobjekt muss aktualisiert werden.

Das Zustandsobjekt wird immer zwischen den Komponenten übergeben und auch ständig von verschiedenen Komponenten mutiert. Die folgende Struktur (s. Abb. 6‑2) zeigt die Standardlösung von Vue für die Zustandsverwaltung (state management) ohne Hilfe von Vuex.

|  |
| --- |
|  |
| Abbildung 6‑2: Kommunikation zwischen Vue Components (eigene Darstellung) |

Das Zustandsobjekt wird als „props“ von Parent‑Komponente zu Child‑Komponente übergeben. Dort wird das Zustandsobjekt durch die Child‑Komponente mutiert. Nach der Mutation wird das mutierte Zustandsobjekt zur Parent-Komponente zurückgegeben, damit anderen Komponenten auch das mutierte Zustandsobjekt von Parent‑Komponente durch one-way data-binding bekommen können.

Die Standardlösung ist einfach zu verstehen und kann ohne großen Aufwand implementiert werden. Deshalb ist sie auch sehr geeignet für kleine bis mittlere Web-Anwendungen. Es gibt aber einige Schwierigkeiten, wenn die Web-Anwendung später groß bzw. komplex wird.

Wie in Abbildung 6‑2 dargestellt, ein Zustand kann durch drei oder auch mehr Child‑Komponenten mutiert werden. Dies macht die spätere Wartungsarbeit aber schwer, weil von außen nicht festgelegt werden kann, welche Komponente was mutiert hat. Die Wartungsarbeit wird noch schwieriger, wenn dazwischen noch asynchrone Prozesse laufen.

Ein anderes Problem kommt vor, wenn in der Web-Anwendung eine ganz tief vernetzte Child‑Komponente das Zustandsobjekt von einer Parent‑Komponente in ganz höherer Ebene ändern muss. Wie in Abbildung 6.2 gezeigt, die Root‑Komponente muss das Zustandsobjekt zuerst der Child‑Komponente zwei in der ersten Ebene übergeben, dann weiter zu Child‑Komponente vier in der zweiten Ebene und am Ende zu tief vernetzte Child‑Komponente fünf in der dritten Ebene. Der Rückweg muss leider auch durch Ebene zwei und eins. Es kann sein, dass die Child‑Komponente zwei und vier gar nichts mit dem Zustandsobjekt, das die Child‑Komponente fünf mutieren will, zu tun haben. Trotzdem muss das Zustandsobjekt zu den beiden übergeben werden. Das ist einerseits sehr aufwendig und verschwendet Zeit bei der Programmierung, andererseits muss eine große Menge von Zustandsobjekten in den Komponenten, die in ganz höhen Ebenen liegen, gespeichert werden. Die Struktur solcher Komponenten wird dadurch chaotisch und das macht die zukünftige Wartungsarbeit nur schwieriger.

Die grundsätzliche Idee von dem Plugin Vuex ist eine Auslagerung aller Zustandsobjekte (Vuex o. J. a). Eine zentrale Lagerung aller Zustandsobjekte ermöglicht, dass jedes zentral gelagertes Zustandsobjekt von beliebiger Komponente ohne großen Aufwand mutiert werden kann. Zusätzlich dazu ist Vuex in der Lage, das mutierte Zustandsobjekt in jeder Komponente, die dieses Zustandsobjekt abgerufen hat, automatisch zu aktualisieren. Das zentralisierte Zustandsmanagement (Centralized State Management) besteht hauptsächlich aus vier Bereichen:

1. State: Wo alle Zustandsobjekte gespeichert sind.
2. Getter: Sammlung der Methoden, die für Abrufen eines Zustandsobjektes zuständig sind.
3. Mutation: Sammlung der Methoden, die für synchrone Mutation eines Zustandsobjektes zuständig sind.
4. Action: Sammlung der Methoden, die für asynchrone Mutation eines Zustandsobjektes zuständig sind.

Die Abbildung 6‑3 zeigt ein Ablaufdiagramm, in dem dargestellt wird, wie eine Komponente durch Vuex ein ausgelagertes Zustandsobjekt mutieren kann und wie eine andere Komponente, die das gleiche Zustandsobjekt braucht, abrufen kann. Ein weiteres Code Beispiel dazu steht ebenfalls im Anhang 6 zur Verfügung.

|  |
| --- |
|  |
| Abbildung 6‑3: Kommunikation zwischen Vue Components durch Vuex (eigene Darstellung) |

Ob und wann Vuex zum Einsatz kommen soll hängt dicht von der Größe der Web-Anwendung ab. Es ist am Ende nur ein Kompromiss zwischen kurzfristiger und langfristiger Produktivität. (Vuex o. J. b). Im EarlyDike Geoportal wird Vuex für solche Zustandsobjekte, die sehr oft von unterschiedlichen Komponenten mutiert werden, eingesetzt (z. B. das Map-Objekt von OpenLayers und Datenobjekt für Informationstab). Das macht die Struktur der einzelnen Komponenten sauber und erleichtertet die zukünftige Wartungsarbeit. Für Zustandsobjekte, die nur zwischen wenigen Komponenten übertragen werden, wird die Traditionelle Art und Weise (s. Abb. 6‑2) verwendet. Da die Verwendung von Vuex immer wieder viel Vorausdenken, vorausplanen und Vorbereitungsarbeit anfordert.

#### UI-Verbesserung mit Vuetify

Mit „plain“ JavaScript beschäftigen die Entwickler sich heutzutage sehr viel mit CSS‑Regeln, die nur sehr schwer zu beherrschen sind. Die unzähligen CSS‑Regeln miteinander zu kombinieren und damit eine schöne Benutzeroberfläche zu erstellen ist eine große Herausforderung für die meisten Entwickler, die sich meistens hauptsächlich auf die Programmierung konzentrieren und nicht so viel Zeit für die Benutzeroberfläche investieren wollen.

Die Benutzeroberfläche ist aber ein besonders wichtiger Teil einer Web-Anwendung. Die Benutzeroberfläche ist der Ort, wo alle Interaktionen zwischen Benutzern und Web-Anwendung stattfinden. Damit die Interaktionen reibungslos gehen können, soll die Benutzeroberfläche so ausgestattet werden, dass die Benutzer mit möglichst wenigen Versuchen schon klar werden, welche Funktionalitäten die Web-Anwendung bietet und wie sie bedient werden können.

Oftmals wird die Wichtigkeit des Designs einer Web-Anwendung nicht gut beachtet, obwohl eine gute Benutzeroberfläche einen großen Anteil der Zeit der gesamten Programmierungsarbeit kosten könnte. Um alle obengenannten Probleme zu lösen, hat das JavaScript‑Framework Vue das Plugin Vuetify zur Verfügung gestellt.

Vuetify bietet sämtliche vordefinierte Vue Components, die gleich wie von den Entwicklern selber gemacht Komponenten sind und direkt ins template Feld eines Vue Components eingesetzt werden können (s. Quellcode 6‑4). Das UI‑Design von solchen vordefinierten Komponenten folgt dem Google Material Design Spec v1 (Vuetify o. J.) und daher brauchen die Entwickler nicht mehr selbst zu testen, ob die Benutzeroberfläche passend aussieht.

Obwohl die Codes in Quellcode 6‑4 auf den ersten Blick ein bisschen schrecklich aussehen, ist die Anwendung der von Vuetify vorgefertigten Komponenten (UI-Components) eigentlich nicht allzu schwer zu lernen. Anhand der offiziellen Dokumentation sowie der Beispiele von jedem UI-Component, können die Regeln zu erstellen eines UI-Component sehr schnell gelernt werden.

Das Hauptziel des Einsatzes von Vuetify ist nicht nur dass die Web-Anwendung am Ende schön aussieht, sondern eine vernünftige und benutzerfreundliche Benutzeroberfläche zu schaffen und gleichzeitig mit Hilfe der vordefinierten UI-Components Zeit sparen.

|  |
| --- |
| 1. **<template>** 2. **<div>** 3. **<v-list-tile>** 5. <!--**<v-list-tile-avatar>** 6. **<v-icon>**layers**</v-icon>** 7. **</v-list-tile-avatar>**--**>** 8. **<v-list-tile-action>** 9. **<v-switch** 10. v-model="checkbox" 11. @click.native="layerSwitch" 12. color="primary" 13. **></v-switch>** 14. **</v-list-tile-action>** 16. <!-- for the helper class, please see: https://vuetifyjs.com/en/style/typography --> 17. **<v-list-tile-content** 18. class="caption" 19. **>** 20. {{ singlelayerinfo.name }} 21. **</v-list-tile-content>** 23. **<v-list-tile-action>** 24. **<v-tooltip** bottom**>** 25. **<v-btn** 26. slot="activator" 27. icon 28. ripple 29. @click="sliderShow = !sliderShow" 30. **>** 31. **<v-icon** color="secondary"**>**opacity**</v-icon>** 32. **</v-btn>** 34. **<span>**change opacity**</span>** 35. **</v-tooltip>** 36. **</v-list-tile-action>** 38. **</v-list-tile>** 40. **<v-list-tile** 41. v-show="sliderShow" 42. **>** 43. **<v-slider** 44. v-model="sliderValue" 45. thumb-label 46. step="10" 47. ticks="always" 48. prepend-icon="settings\_brightness" 49. @change="opacityChange" 50. **></v-slider>** 51. **</v-list-tile>** 52. **</div>** 53. **</template>** |
| Quellcode 6‑4: Anwendungsbeispiel: Vuetify UI-Component Lists im EarlyDike Geoportal (eigene Darstellung) |

### Details in Komponenten

#### Implementierung von unterschiedlicher Layers

Die zentrale Aufgabe von EarlyDike Geoportal soll die Darstellung unterschiedlicher Geoinformationen von verschiedenen Quellen sein. Die Geoinformationen, die in der Karte von Geoportal dargestellt werden, sind im großen Teil Layers und Features aus einem Layer Deshalb hat eine Vernünftige Implementierung aller Layers eine große Bedeutung.

Die Layers werden mit Hilfe von OpenLayers Bibliothek durch WMS- bzw. WFS-Dienste abgerufen und im Geoportal dargestellt. Zum Abrufen werden jedoch einige Parameter gebraucht, z. B. url, layers, format, SRS etc. Bevor der Implementierung des JavaScript‑Framework wurden fast alle Parameter zur Erzeugung des Layers verteilt in JavaScript‑Codes fest programmiert. Das macht die JavaScript‑Codes extrem lang und extrem schwer zu ändern. Daher wird im Rahmen dieser Masterarbeit ein systematische Implementierungsweise entwickelt, die einfach zu verstehen, zu verbessern und für andere Projekte einzusetzen sind (s. Abb. 6‑4).

|  |
| --- |
|  |
| Abbildung 6‑4: Ablaufdiagramm: systematische Implementierung der Layers (eigene Darstellung) |

Zuerst werden alle benötigten Parameter zur Erzeugung eines Layers nicht mehr verteilt in JS‑Datei platziert, sondern in einer Konfigurationsdatei ordentlich je nach Gruppe zusammengefasst. Dann wird eine spezielle JS-Datei für die Bearbeitung aller Parameter zu OpenLayers Layergruppe- / Layer-Objekte eingesetzt.

Diese Vorgehensweise bringt einige Vorteile:

1. Alle benötigten Parameter werden jetzt systematisch in einer Konfigurationsdatei zusammengefasst (s. Quellcode 6‑5). D. h. Änderungen von existierenden Parametern sowie Erweiterungen für neue Layers sind einfach vorzunehmen.
2. Die systematische Bearbeitung durch die JS-Datei GIAGS\_layer.js kann im Prinzip für jedes Projekt, das OpenLayers angewendet hat, wiederverwendet werden.

|  |
| --- |
| 1. { 2. "groupName": "singleLayer", 3. "layerCollection": [ 4. { 5. "type": "WMSBase", 6. "url": "http://service.gdi-sh.de/WMS\_SH\_BDDcol\_KF?", 7. "layers": "DTK5col,DTK25col,DTK50col,DTK100col,UEK250,UEK600,UEK1500", 8. "name": "layer\_sh", 9. "format": "image/jpeg", 10. "SRS": "EPSG:4326", 11. "version": "1.1.1", 12. "visible": **true** 13. } 14. ] 15. } |
| Quellcode 6‑5: Zusammengefasste Parameter von einem Layer in GIAGS\_config.json (eigene Darstellung) |

Nachdem, dass alle Layers erfolgreich in das OpenLayers Map-Objekt hinzugefügt wurden, wird das Map-Objekt auch in Vuex als Standardzustand gespeichert. Damit ist einer der Kernaufgaben - die systematische Implementierung von Layers fertig.

#### Implementierung von Auswahllisten der Layers

Die Auswahllisten der Layers bietet den Benutzern einige Basisfunktionalitäten des Geoportals an. Z. B.: Das An- und Ausschalten der Layers sowie die Opazitätsänderung von einzelnem Layer.

Im Geoportal werden für die Auswahllisten insgesamt drei Kategorien geplant: Basic Layer, WMS Layer und WFS Layer. Daher werden drei Komponenten dafür erstellt. Jede Kategorie besteht aus einzelnem Layer und eventuell auch Layer-Gruppe. Deshalb werden wieder zwei Child‑Komponenten dafür gebraucht. Da die Layer-Gruppe wiederum aus einzelnem Layer besteht, kann die Komponente für einzelnen Layer wiederum als Child‑Komponente für die Komponente der Layer-Gruppe verwendet werden (s. Abb. 6‑5).

Die in Abbildung 6‑5 gezeigte Struktur benutzt die volle Kraft von Wiederverwendbarer Komponente. Die Wiederverwendbarer Komponente ist eine der zentralen Ideen bei der Entwicklung einer Web-Anwendung mit dem JavaScript‑Framework Vue oder auch mit anderen Komponentenbasierten Frameworks.

|  |
| --- |
|  |
| Abbildung 6‑5: Struktur der Auswahllisten von Layers (eigene Darstellung) |

#### Implementierung von Informationstabs

Die Informationen der Features aus unterschiedlichen Layers werden in unterschiedliche Informationstabs dargestellt. Es gibt insgesamt vier Tabs: WFS-INFO, WMS-INFO, DIAGRAMM und ACTIVE LAYER. Die dienen dazu, alle verfügbaren Geoinformationen von dem Benutzer ausgewählten Layers klassifiziert darzustellen.

Nur die von Benutzer ausgewählten Geoinformationen sollen dargestellt werden. D. h. Jede Interaktion zwischen dem Benutzer und Geoportal könnte theoretisch eine Auswirkung auf zu darstellende Geoinformationen haben. In Tabelle 6‑1 werden die Interaktionen und zusammenhängende Auswirkungen aufgelistet.

Tabelle 6‑1: Interaktionen und Auswirkungen auf die Informationstabs

|  |  |
| --- | --- |
| Interaktionen | Auswirkungen auf die Informationstabs |
| Benutzer schaltet einen Layer ein / aus | Im Tab ACTIVE LAYER müssen die Informationen des neuen Layers hinzugefügt / gelöscht werden |
| Benutzer klickt auf die Karte (kein WFS Features) | Alle Informationen zur aktuellen Layers an dem geklickten Punkt müssen in WMS-INFO dargestellt werden. |
| Benutzer klickt auf ein WFS Features auf der Karte | Alle Informationen zu diesem Feature müssen im Tab WFS-INFO und eventuell auch im Tab DIAGRAMM dargestellt werden. |

Für die obengenannten drei verschiedenen Interaktionen stehen im Hintergrund drei unterschiedliche Zustandsobjekte (States). Mit der Unterstützung von Vuex können die Zustandsobjekte ausgelagert und optimiert verwaltet werden (s. Abb. 6‑3 und Abb. 6‑6).

|  |
| --- |
|  |
| Abbildung 6‑6: Aktualisierung der Daten von Informationstab mit Vuex (eigene Darstellung) |

Mit der Unterstützung von Vuex müssen wir nicht mehr um die Synchronisierung der Mutation von den Zustandsobjekten kümmern. Die im Rahmen dieser Masterarbeit angefertigten Informationstabs sollen als ein Beispiel für spätere Erweiterung dienen. Es ist zu erwarten, dass in der Zukunft mehr unterschiedliche Inhalte sowie Funktionalitäten in Informationstabs kommen. Aber mit den vorhandenen Informationstabs sollte eine Erweiterung nicht allzu schwer sein.

## Projekt übergreifend wiederverwendbare Dateien

Eins der Ziele dieser Masterarbeit ist die Entwicklung einer Grundgerüste bzw. einer Standardvorgehensweise für andere Projekte, die auch verteilte Geoinformationen darstellen. Um dies zu verwirklichen, reicht die Implementierung von JavaScript‑Framework Vue sowie anderen Plugins noch nicht aus. Daher werden folgenden Dateien (s. Tab. 6‑2) während der Entwicklung von neuem EarlyDike Geoportal erstellt. Die vier Dateien wurden absichtlich nicht Projekt spezifisch aufgebaut und können daher als Fundamente für andere Projekte wiederverwendet werden. Eine detaillierte Beschreibung dazu und wie solche Dateien für die zukünftige Projekte eingesetzt werden können finden Sie ebenfalls in der Tabelle 6‑2.

Tabelle 6‑2: Projekt übergreifend wiederverwendbare Dateien

|  |  |
| --- | --- |
| Dateiname | Beschreibung |
| **GIAGS\_Config.json**  *(Konfigurierungsdatei für EarlyDike Geoportal)* | Diese JSON-Datei beinhaltet die benutzerdefinierten Konfigurationen der Web-Anwendung:   * Projektion der Karte * Zentralpunkt der Karte * Zoomstufe der Karte * Zusammengefasste Parameter der Layers |
| **GIAGS\_layer.js**  *(Die Layerfabrik)* | Diese JavaScript-Datei ist zu ständig für die Bearbeitung der aus GIAGS\_Config.json gelesenen Parameter der Layers. Je nach den Typen der unterschiedlichen Layers werden die zusammengefassten Parameter entsprechend bearbeitet. Am Ende gibt die GIAGS\_layer.js immer ein OpenLayers Layer-Objekt zurück, in dem alle einzelne Layers nach den Anforderungen von OpenLayers aufgebaut werden.  Diese Datei kann im Prinzip unverändert für andere Projekte eingesetzt werden, sofern die in GIAGS\_config.js festgelegte Struktur gleich bleibt. |
| **GIAGS\_proj\_defs.js**  *(Projektionsdefinitionen)* | Diese JavaScript-Datei enthält jede Menge Projektionsdefinitionen, die OpenLayers nicht als Standardprojektionen anbietet, aber für Geoinformationen in Deutschland oftmals gebraucht werden. Für z. B. Koordinatentransformation werden sie gebraucht.  Für ähnliche Projekte, die eine Koordinatentransformation brauchen, ist sie einzusetzen. |
| **GIAGS\_OlstyleDefs.js**  *(Die vordefinierten Stile für OpenLayers Vektorlayer)* | In dieser JavaScript-Datei werden alle Definitionen von Styles des Vektorlayers von OpenLayers gespeichert. Die Definitionen beschreiben, wie ein Feature mit einer bestimmten Geometrie (Punkt, Polygon, Kreis, etc.) auf der Karte aussehen soll.  Diese Datei kann ebenfalls direkt für andere Projekte als Standardstyle eingesetzt werden. |

## Fazit

Die Implementierung des JavaScript‑Framework Vue durch CLI bietet einen schnellen Weg zur eine moderne Entwicklungsumgebung, wobei alle Konfigurationen automatisch durch das CLI‑Werkzeug eingestellt werden. Mit den Plugins Vuex und Vuetify können die Grundstruktur und Benutzeroberfläche nochmal verbessert werden. Dank der von Vue‑CLI gebrachten modernen Entwicklungsumgebung kann OpenLayers 5 auch auf ES6 Modules basiert implementiert werden.

Allerdings ist die zeitliche Beanspruchung im Vergleich zu der Implementierung durch CDN enorm gestiegen. Das Basiswissen von Framework Vue reicht eben nicht mehr aus. Die Anwendung des Vue‑CLI verlangt Voraussetzungen von Node.js und die zugehörigen Werkzeuge. Außerdem steht noch die Einstiegsaufwand von Vuex und Vuetify im Weg.

In einem zukünftigen Aspekt sind die Bemühungen für alle obengenannten zusätzlichen Wissen aber ein sehr gutes Fundament.

# Zusammenfassung

Eine Web-Anwendung zu entwickeln ist heutzutage kein großes Thema mehr. Jeder, der ein Basiswissen über JavaScript, HTML und CSS besitzt, kann eigene Web-Anwendung aufbauen. Aber die Erwartungen und die Anforderungen von den Benutzern haben sich in den letzten Jahren auch sehr stark gewachsen. Statisches HTML existiert fast nicht mehr und dynamisches HTML wird immer komplexer: Hinter einer schönen Animation verstecken sich unzählige Zeilen von CSS-Regeln. Die zahllosen direkten DOM‑Manipulationen haben bei großer Web-Anwendung nicht nur die Grenze der Leistung vom Rechner, sondern auch die Grenze der Wartbarkeit von Codes erreicht. Wegen solcher Probleme kann eine moderne Web-Anwendung kaum noch nur mit JavaScript, HTML und CSS entwickelt werden.

Um mit solchen Problemen umzugehen, kommen die JavaScript‑Frameworks erst im Jahr 2010 ins Spiel. Das AngularJS ist das erste JavaScript‑Framework mit komplettem Architekturmuster (MVC) und hat sofort an Popularität gewonnen (Barker 2018). Danach wurden in Jahr 2013 das React von Facebook und in Jahr 2014 das Vue von Evan You veröffentlicht. Die drei Frameworks zusammen dominieren zurzeit den Market von Web-Entwicklung. Heutzutage ist ein JavaScript‑Framework bei der Entwicklung eine mittlerer bis größerer Web-Anwendung eine Standardlösung.

Außerdem werden eine große Menge von neuen Web-Technologien auf dem Market zur Verfügung gestellt. Einige davon sind weit verbreitet, z. B.: NPM, Webpack, TypeScript etc. Sie alle bieten bestimmte Lösungen zu bestimmten Problemen. Geeignete Werkzeuge zu finden und sie auch richtig anzuwenden ist ganz wichtig.

In Rahmen dieser Masterarbeit wird zukunftsorientiert auf JavaScript‑Frameworks und einigen mit JavaScript-Frameworks zusammenhängenden Web-Technologien geforscht. Nach der Analyse über die am meisten benutzten Frameworks auf dem Market wird das JavaScript‑Framework Vue ausgewählt. Damit werden zwei Versuche mit jeweils unterschiedlicher Art und Weise der Implementierung durchgeführt.

Das neu aufgebaute EarlyDike Geoportal soll als ein zukunftsorientiertes Muster bei der Entwicklung der Web-Anwendung mit JavaScript‑Framework zur Darstellung verteilter Geoinformation dienen.

# Literaturverzeichnis

1&1 Ionos SE, 2018

„Die beliebtesten JavaScript-Frameworks und -Bibliotheken“, <https://www.ionos.de/digitalguide/websites/web-entwicklung/beliebte-javascript-frameworks-und-bibliotheken/>, Zugriff am: 15.12.2018.

Ackermann, Philip, 2015

„Architekturmuster und Konzepte moderner JavaScript-Webframeworks – Bewährte Lösungsansätze“ in: webundmobile 10/2015, S. 12-23.

Barker, Adam, 2018

„The Super-Brief History of JavaScript Frameworks For Those Somewhat Interested“, <https://dev.to/_adam_barker/the-super-brief-history-of-javascript-frameworks-for-those-somewhat-interested-3m82>, Zugriff am: 14.01.2018.

Bashir, Ashraf, 2014

„What are MVP and MVC and what is the difference?“, <https://stackoverflow.com/questions/2056/what-are-mvp-and-mvc-and-what-is-the-difference>, Zugriff am: 15.12.2018.

Burgess, Matt, 2018

„Why is there so much to learn in web development?“, <https://medium.com/@mattburgess/why-is-there-so-much-to-learn-in-web-development-41adbc54731c>, Zugriff am: 19.12.2018.

Deacon, John, 2005

„Model-View-Controller (MVC) Architecture“, <http://www.johndeacon.net/JDL/briefings/mvc.html>, Zugriff am: 15.12.2018.

Gimeno, Alberto, 2018

„The deepest reason why modern JavaScript frameworks exist“, <https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445>, Zugriff am: 15.12.2018.

Gore, Anthony, 2018

„Vue CLI 3: A Game Changer For Frontend Development“, <https://vuejsdevelopers.com/2018/03/26/vue-cli-3/>, Zugriff am: 02.01.2019.

Komanduri, Sai Kishore, 2016

„The one thing that no one properly explains about React — Why Virtual DOM“, <https://hashnode.com/post/the-one-thing-that-no-one-properly-explains-about-react-why-virtual-dom-cisczhfj41bmssp53mvfwmgrq>, Zugriff am: 15.12.2018.

OpenLayers, 2018a

„Quick Start“, Offizielle Homepage der JavaScript-Bibliothek OpenLayers, <https://openlayers.org/en/latest/doc/quickstart.html>, Zugriff am: 20.12.2018

OpenLayers, 2018b

„Hosted builds for development“, Offizielle Homepage der JavaScript-Bibliothek OpenLayers, <https://openlayers.org/download/>, Zugriff am: 20.12.2018.

OpenLayers, 2018c

„v5.0.0“, Offizielle release page der JavaScript-Bibliothek OpenLayers, <https://github.com/openlayers/openlayers/releases/>, Zugriff am: 04.01.2019.

Reyes, Camilo, 2017

„The MVC Design Pattern in Vanilla JavaScript“, <https://www.sitepoint.com/mvc-design-pattern-javascript/>, Zugriff am: 15.12.2018.

Vue, 2018a

„Single File Components - Introduction“,

<https://vuejs.org/v2/guide/single-file-components.html#Introduction>, Zugriff am: 19.12.2018.

Vue, 2018b

„Installation - NPM“, <https://vuejs.org/v2/guide/installation.html#NPM>, Zugriff am: 19.12.2018.

Vue, 2018c

„The Vue Instance – Creating a Vue Instance“, <https://vuejs.org/v2/guide/instance.html#Creating-a-Vue-Instance>, Zugriff am: 20.12.2018.

Vuex, o. J. a

„What is a "State Management Pattern"?“, <https://vuex.vuejs.org/#what-is-a-state-management-pattern>, Zugriff am: 10.01.2019.

Vuex, o. J. b

„When Should I Use It?“, <https://vuex.vuejs.org/#when-should-i-use-it>, Zugriff am: 10.01.2019.

Vuetify, o. J.

„Design principles“, <https://vuetifyjs.com/en/getting-started/why-vuetify#design-principles>, Zugriff am: 11.01.2019.

Schwarzmüller, Maximilian, 2017

„A Brief Introduction to the Frameworks“, <https://www.academind.com/learn/javascript/angular-vs-react-vs-vue-my-thoughts/>,

Zugriff am: 19.12.2018.

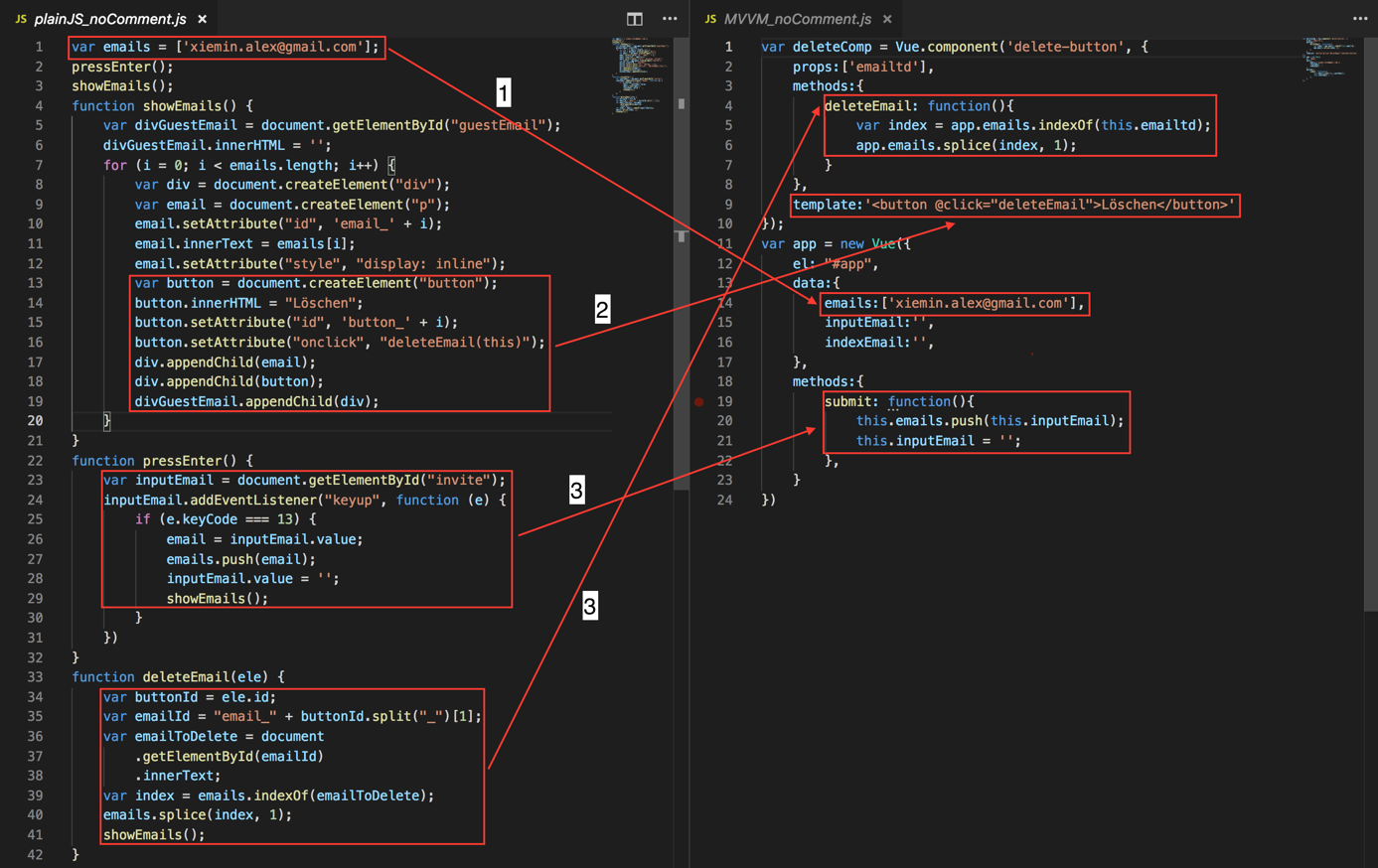
# Anhang 1: Beispiel - MVC und MVP Architekturmuster

Alle Dateien zum Anhang 1 finden Sie unter folgendem Pfad: Anhänge\Anhang\_1

# Anhang 2: Beispiel - plain JavaScript vs. JavaScript‑Framework Vue

Alle Dateien zum Anhang 2 finden Sie unter folgendem Pfad: Anhänge\Anhang\_2

# Anhang 3: Schritten der Migration



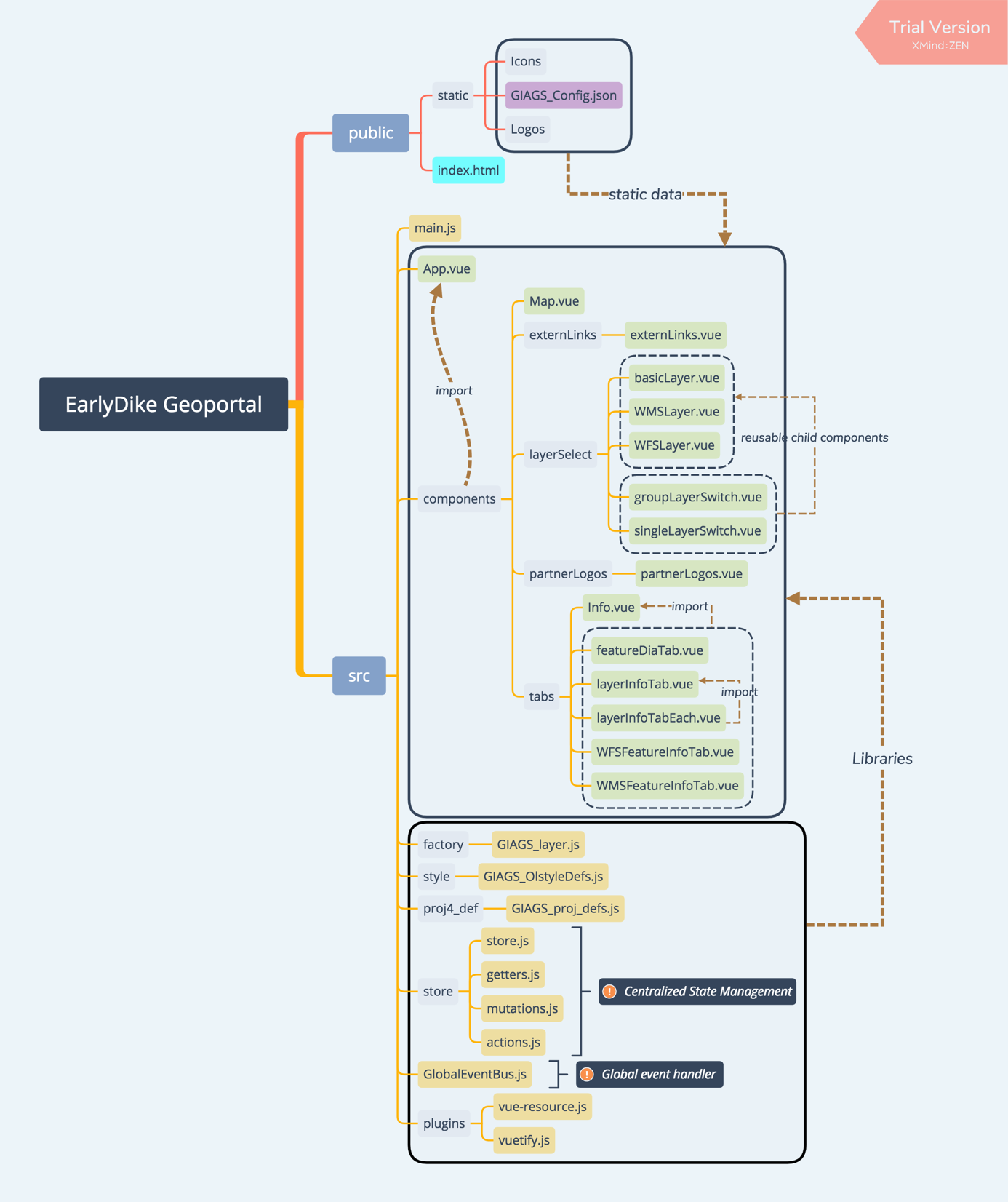
# Anhang 4: Codes von EarlyDike Geoportal (CDN Version)

Alle Dateien zum Anhang 4 finden Sie unter folgendem Pfad: Anhänge\Anhang\_4

# Anhang 5: Codes von EarlyDike Geoportal (CLI Version)

Alle Dateien zum Anhang 5 finden Sie unter folgendem Pfad: Anhänge\Anhang\_5

# Anhang 6: Grundstruktur von EarlyDike Geoportal (CLI Version)



# Anhang 7: Code Beispiel über Veux

Alle Dateien zum Anhang 7 finden Sie unter folgendem Pfad: Anhänge\Anhang\_7