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

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

Inhaltsverzeichnis

[1 Einleitung: Der Hintergrund und die Motivation 7](#_Toc533103583)

[1.1 Vorstellung des Projekts EarlyDike 7](#_Toc533103584)

[1.2 Schwierigkeiten und Probleme in der Praxis 7](#_Toc533103585)

[1.3 Vorteile der Implementierung von JavaScript-Frameworks 7](#_Toc533103586)

[2 Die Architekturmuster für den Aufbau einer Web-Anwendung 9](#_Toc533103587)

[2.1 Die Architekturmuster und ihre Beziehungen zu JavaScript-Frameworks 9](#_Toc533103588)

[2.2 Model-View-Controller (MVC) und Model-View-Presenter (MVP) 10](#_Toc533103589)

[2.2.1 Grundstruktur 10](#_Toc533103590)

[2.2.2 Workflow und Beispiele 10](#_Toc533103591)

[2.2.3 Fazit 11](#_Toc533103592)

[2.3 Model-View-ViewModel (MVVM) 12](#_Toc533103593)

[3 Vergleich und Auswahl von JavaScript-Frameworks 15](#_Toc533103594)

[3.1 Allgemeine Vergleichung 15](#_Toc533103595)

[3.1.1 Popularität und Geschichte 15](#_Toc533103596)

[3.1.2 Lernkurve 16](#_Toc533103597)

[3.1.3 Besonderheiten 17](#_Toc533103598)

[3.2 Praxisnahe Vergleichung: Implementierungsflexibilität 18](#_Toc533103599)

[3.3 Fazit 18](#_Toc533103600)

[4 Vorbereitung für die Implementierung des JavaScript-Frameworks Vue in das Projekt EarlyDike 20](#_Toc533103601)

[4.1 Angestrebte Ziele 20](#_Toc533103602)

[4.2 Hintergrund und Arten der Implementierung 20](#_Toc533103603)

[4.2.1 Hintergrund 20](#_Toc533103604)

[4.2.2 Arten der Implementierung 22](#_Toc533103605)

[4.3 Entscheidung der Art der Implementierung 25](#_Toc533103606)

[5 Versuch 1: Implementierung durch CDN 27](#_Toc533103607)

[5.1 Grundprinzip: Migration 27](#_Toc533103608)

[5.2 Grundstruktur des Projekts EarlyDike 27](#_Toc533103609)

[5.3 Technische Details über die Migration 28](#_Toc533103610)

[5.3.1 Allgemeine Hinweise 28](#_Toc533103611)

[5.3.2 Template Syntax von Vue 28](#_Toc533103612)

[5.3.3 Komponentenweiser Aufbau durch Vue Components 29](#_Toc533103613)

[6 Versuch 2: Implementierung durch CLI 31](#_Toc533103614)

[6.1 Initialisierung des Projekts EarlyDike Geoportal mit der Hilfe von Vue CLI 3 31](#_Toc533103615)

[6.2 Grundstruktur des Projekts EarlyDike 32](#_Toc533103616)

[6.3 Technische Details 32](#_Toc533103617)

[6.3.1 Details in globaler Ebene 32](#_Toc533103618)

[6.3.2 Details in Komponenten 33](#_Toc533103619)

[6.4 Projekt übergreifend wiederverwendbare Dateien 33](#_Toc533103620)

[6.4.1 Die Konfigurierung des EarlyDike Geoportals - GIAGS\_Config.json 33](#_Toc533103621)

[6.4.2 Layerfabrik - GIAGS\_layer.js 33](#_Toc533103622)

[6.4.3 Die Projektionsdefinitionen - GIAGS\_proj\_defs.js 33](#_Toc533103623)

[6.4.4 Die vordefinierten Stile für OpenLayers Vektorlayer - GIAGS\_OlstyleDefs.js 33](#_Toc533103624)

[7 Fazit: 33](#_Toc533103625)

Abbildungsverzeichnis

[Abbildung 2‑1: Grundstruktur MVC (Ackermann 2015, S. 13) 10](#_Toc533103626)

[Abbildung 2‑2: Grundstruktur MVP (Ackermann 2015, S. 13) 10](#_Toc533103627)

[Abbildung 2‑3: Workflow MVC (Bashir 2014) 11](#_Toc533103628)

[Abbildung 2‑4: Workflow MVP (Bashir 2014) 11](#_Toc533103629)

[Abbildung 2‑5: Grundstruktur MVVM (Ackermann 2015, S. 16) 12](#_Toc533103630)

[Abbildung 2‑6: plain JS vs. Vue (eigene Darstellung) 14](#_Toc533103631)

[Abbildung 3‑1: Numbers of stars added on GitHub for Front-end Frameworks in 2017 (Quelle: https://risingstars.js.org/2017/en/#section-framework) 15](#_Toc533103632)

[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) 16](#_Toc533103633)

[Abbildung 3‑3: Lernkurve Angular vs. React vs. Vue (Quelle: Schwarzmüller 2017) 17](#_Toc533103634)

[Abbildung 4‑1: Beispiel: Liste von Dependencies in package.json Datei (eigene Darstellung) 24](#_Toc533103635)

[Abbildung 4‑2: Vordefinierte Struktur von neu erstelltem Projekt mit extra Anforderungen an TypeScript- und Vuex-Unterstützung (eigene Darstellung) 25](#_Toc533103636)

[Abbildung 5‑1: Grundstruktur des Projekts EarlyDike - CDN Version (eigene Darstellung) 27](#_Toc533103637)

Tabellenverzeichnis

[Tabelle 3‑1: zusätzliche Kenntnisse für jeweilige JS-Frameworks (eigene Darstellung) 16](#_Toc533103638)

[Tabelle 3‑2: Implementierungsmöglichkeiten von JS-Frameworks 18](#_Toc533103639)

[Tabelle 4‑1: Erfüllungsgrad der angestrebten Ziele von unterschiedlichen Arten der Implementierung (eigene Darstellung) 25](#_Toc533103640)

Quellcodeverzeichnis

[Quellcode 4‑1: Einbindung des JavaScript-Framework Vue durch CDN Link in HTML (eigene Darstellung) 22](#_Toc533103689)

[Quellcode 4‑2: Eirichtung von Dependencies durch NPM: Vue und Vue CLI 3 (eigene Darstellung) 23](#_Toc533103690)

[Quellcode 4‑3: Einrichtung der Entwicklungsumgebung durch NPM (eigene Darstellung) 23](#_Toc533103691)

[Quellcode 5‑1: Direkte DOM-Manipulationsmethoden 28](#_Toc533103692)

[Quellcode 5‑2: Templateteil von Vue Instance / Component (eigene Darstellung) 29](#_Toc533103693)

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*

# Einleitung: Der Hintergrund und die Motivation

## Vorstellung des Projekts EarlyDike

## Schwierigkeiten und Probleme in der Praxis

* The callback hell problem
* // Link: <https://stackoverflow.com/questions/25098066/what-is-callback-hell-and-how-and-why-rx-solves-it>
* The main problem modern JavaScript frameworks solve is keeping the UI in sync with the state
* //Link:<https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445>

## 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): (hier code-Beispiel als Anhang 1 möglich, MVC\_MVP in MA\_code\_example)

* 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 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

## Grundprinzip: Migration

In den meisten Szenarien ist ein komplett neuer Aufbau 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 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 über die Migration

// TODO: what needs to be done during the migration? What is different from the old days?

// TODO: und muss was bei der Migration gemacht werden?

### 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.

### Template Syntax von Vue

#### Virtual DOM

Um ein dynamisches UI zu erzeugen muss das DOM ständig manipuliert werden. Die in Quellcode 5-1 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 5‑1: 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 5-2 wird der Templateteil 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 5‑2: Templateteil 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 z. B. bedingte Darstellung von View, reaktive Aktualisierung von HTML-Attributen etc. Im Quellcode 5-2 ist das Zeichen „@“ ein „shorthand“ für das „v-on“ Directive und „v‑model“ ist auch ein Directive für bidirektionale Datenbindung.

### Ordentlicher Aufbau mit Vue Components

Durch die Anwendung von JavaScript-Framework Vue werden die Entwickler angefordert, Komponenten aufzubauen und ordentlich zuzuordnen. 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) |

Ein Vue Component kann als ein ViewModel gesehen werden (s. auch 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 Teilen wie z. B.: data, computed, watch, methods und lifecycle hooks (s. Quellcode 5‑3). Der Teil data entspricht den bereitgestellten Datenfelder in ViewModel (s. Kapitel 2.3). Der Teil methods enthält alle Javascript Funktionen. Der Teil template ist zuständig, um das Virtual DOM zuerstellen und zusammen mit anderen Teilen es zu manipulieren.

|  |
| --- |
| 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 5‑3: Registrierung globaler Komponente in Vue (eigene Darstellung) |

Die meisten Teile der Migration folgen nach den folgenden Schritten (s. Abb. 5‑3 und Anhang 2):

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

|  |
| --- |
|  |
| Abbildung 5‑3: Schritten der Migration (eigene Darstellung) |

/\*

In Vue gibt es zwei Typen von Komponenten: Global und Lokale. Da für diesen Aufbau hier kein modulares Aufbausystem mit der Unterstützung von z. B. Webpack benutzt wird, werden alle Komponenten einfach global registriert. Die Registrierung wird in dem sogenannten „global scope“ wie in Quellcode 5-3 durchgeführt.

\*/

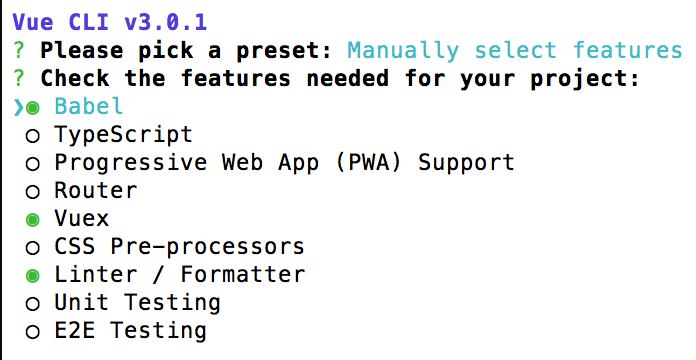
## Fazit

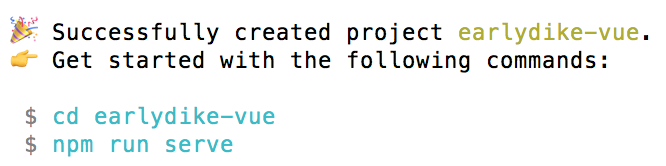
// TODO: see the notes 20.12.2018

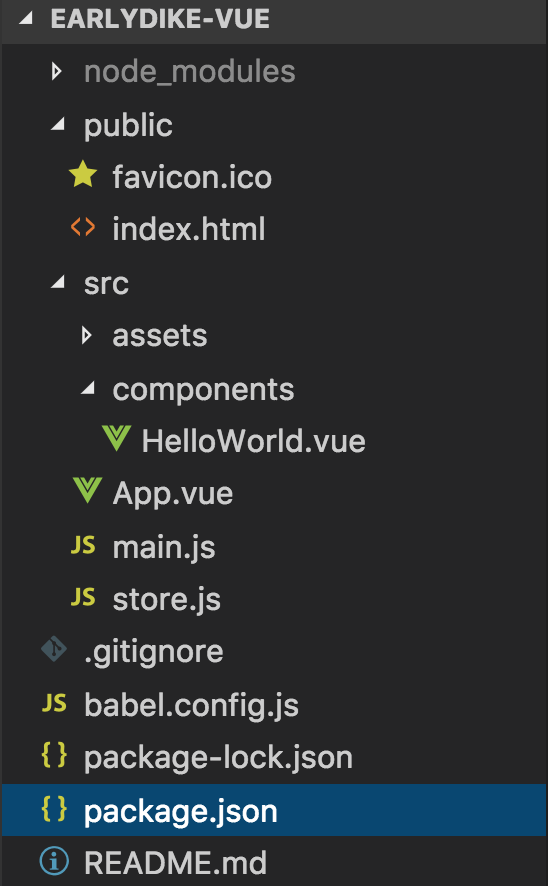
# Versuch 2: Implementierung durch CLI

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

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

Dann wird gefragt, ob die Standardeinstellung angewendet soll. Da das EarlyDike Geoportal Projekt das Plugin Vuex später braucht, wollen wir das selbst konfigurieren und das Plugin bei der Initialisierung schon hinzufügen. Außerdem werden Babel und Linter für die ES6 Syntax Transformation und Fehlererkennung der Codes als Standard ausgewählt (s. Abb. XXX).

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. Das kann ein paar Minuten dauern und diese Nachricht zeigt, dass die Initialisierung des Projekts fertig ist.

Das ist die vordefinierte Struktur von Vue CLI erzeugtem Projekt mit benutzerdefinierten Einstellungen (s. Abb. XXX). Es fehlt noch die OpenLayers Bibliothek sowie das Vuetify Plugin. Die können aber später noch manuell zu dem Projekt hinzugefügt werden.

## Grundstruktur des Projekts EarlyDike

## Technische Details

### Details in globaler Ebene

#### Von OpenLayers 3 bis OpenLayers 5

// TODO: Die höchste Version von OpenLayers, mit der man die alte JavaScript Codes nicht anpassen muss, ist die Version 4.6.5. Ab die Version 5 unterstützt OpenLayers nur auf ES6 Modules basierte Implementierung.

#### Die Vorteile der Anwendung von Vue CLI 3

#### Centralized State Management mit Vuex

#### UI-Verbesserung mit Vuetify

### Details in Komponenten

#### Implementierung von unterschiedlicher Layers (Map.vue)

#### Implementierung von Auswahllisten der Layers (layerSelect Folder)

#### Implementierung von Informationstabs (tabs Folder)

## Projekt übergreifend wiederverwendbare Dateien

### Die Konfigurierung des EarlyDike Geoportals - GIAGS\_Config.json

### Layerfabrik - GIAGS\_layer.js

### Die Projektionsdefinitionen - GIAGS\_proj\_defs.js

### Die vordefinierten Stile für OpenLayers Vektorlayer - GIAGS\_OlstyleDefs.js

# Fazit:

Quellenverzeichnis

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.

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.

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.

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.

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.