Kapitel 4 Der Wiederaufbau des EarlyDike Geoportals mit der Kraft von Vue

4.1 Angestrebte Ziele

Allgemein möchte man am Ende ein gut strukturiertes Projekt haben, das nicht nur für ihn selber sondern für andere Entwickler auch einfach zu verstehen ist und damit weniger Wartungsaufwand und Erweiterungsaufwand aufweist. Der Wiederaufbau 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 hat man die folgenden 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 um die zukünftige potentiale Kompatibilitätsproblem zu vermeiden.
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.

4.2 Hintergrund und Arten der Implementierung

4.2.1 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.js, 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. (Beispiel GIAGS\_inputForm.js)
* 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 HauptJavascript 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.

4.2.2 Arten der Implementierung

4.2.2.1 CDN Links (Content Delivery Networks)

JavaScript-Framework kann auch ganz einfach wie eine JavaScript-Bibliothek durch die Einbindung eines Script-Elements in HTML implementiert werden. Die Abbildung XXX zeigt wie das JavaScript-Framework Vue.js mit Hilfe von CDN Links implementiert werden kann:

(Roter Rahmen für Script-Tag noch, sowie Abbildungsverzeichnis)

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.

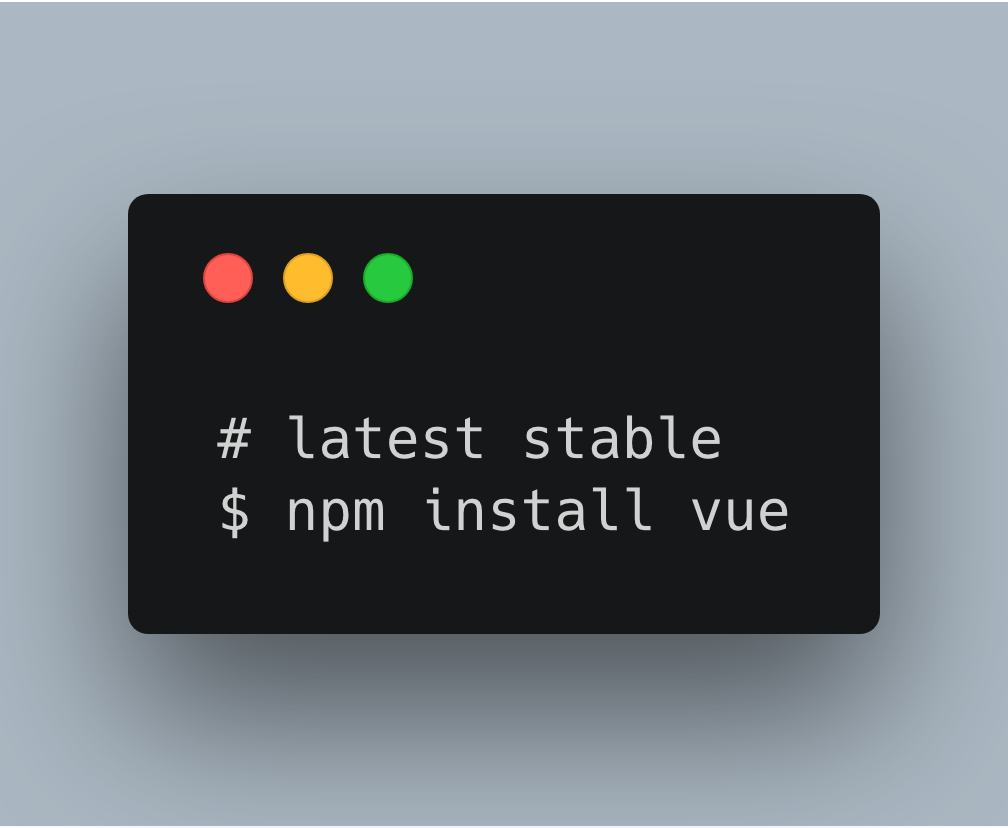
Laut der offiziellen Dokumentation von Vue.js ist diese Art der Implementierung allerdings eigentlich nur geeignet, kleine bis mittlere Web-Anwendungen bzw. Web-Anwendungen, die nicht ausschließlich von JavaScript getrieben sind, aufzubauen (Zitat: https://vuejs.org/v2/guide/single-file-components.html#Introduction). 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.

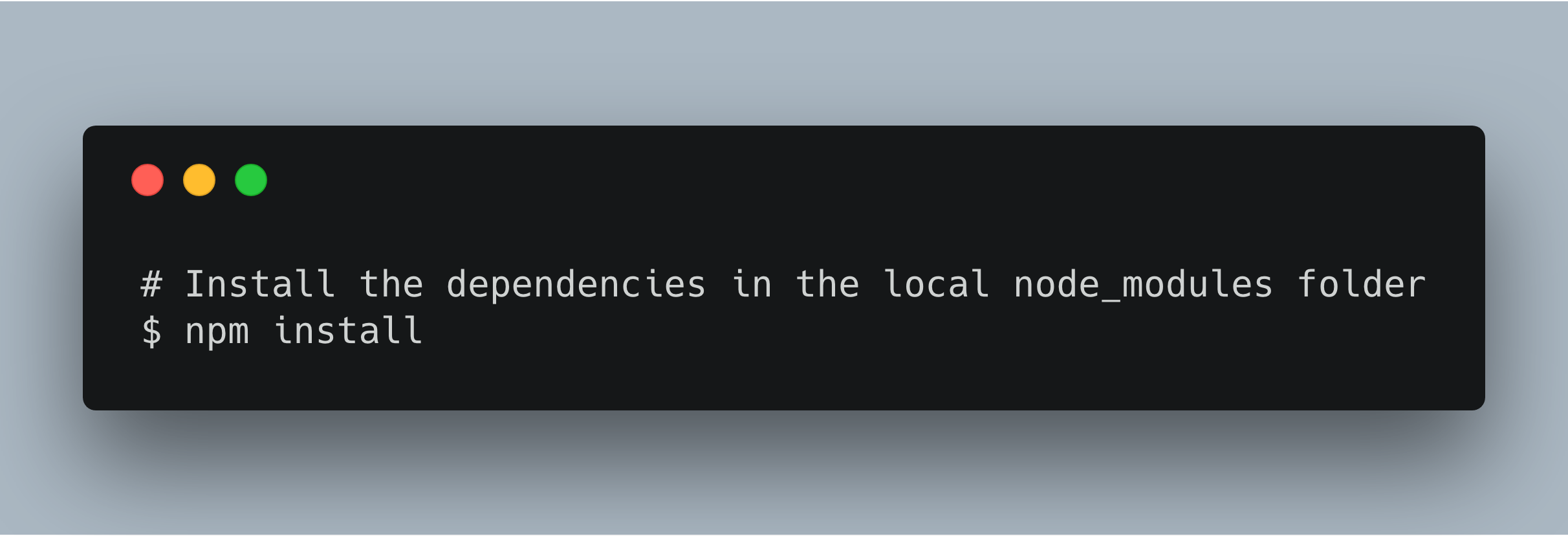
4.2.2.2 NPM (Node Package Manager)

NPM ist ein command-line Werkzeug und wurde entwickelt für die Verwaltung der Dependencies der Web-Anwendungen. Dependencies (external code requirement, https://medium.com/@mattburgess/why-is-there-so-much-to-learn-in-web-development-41adbc54731c) sind z. B. Bibliotheken oder Frameworks, die meistens von anderen Entwicklern geschrieben sind und für eine Web-Anwendung benutzt werden. Ein Framework gehört auch dazu. 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. Abb. X).

(Eirichtung von Dependencies durch NPM () Vue.js und Vue CLI 3)

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 andere Bibliotheken können durch NPM verwaltet werden. Z. B. für neue Entwicklungsumgebung können alle Dependencies auf einmal installiert werden (s. Abb. X, <https://docs.npmjs.com/cli/install>).
* Alle benötigte Dependencies (mit Version) werden in package.json Datei ordentlich aufgelistet.

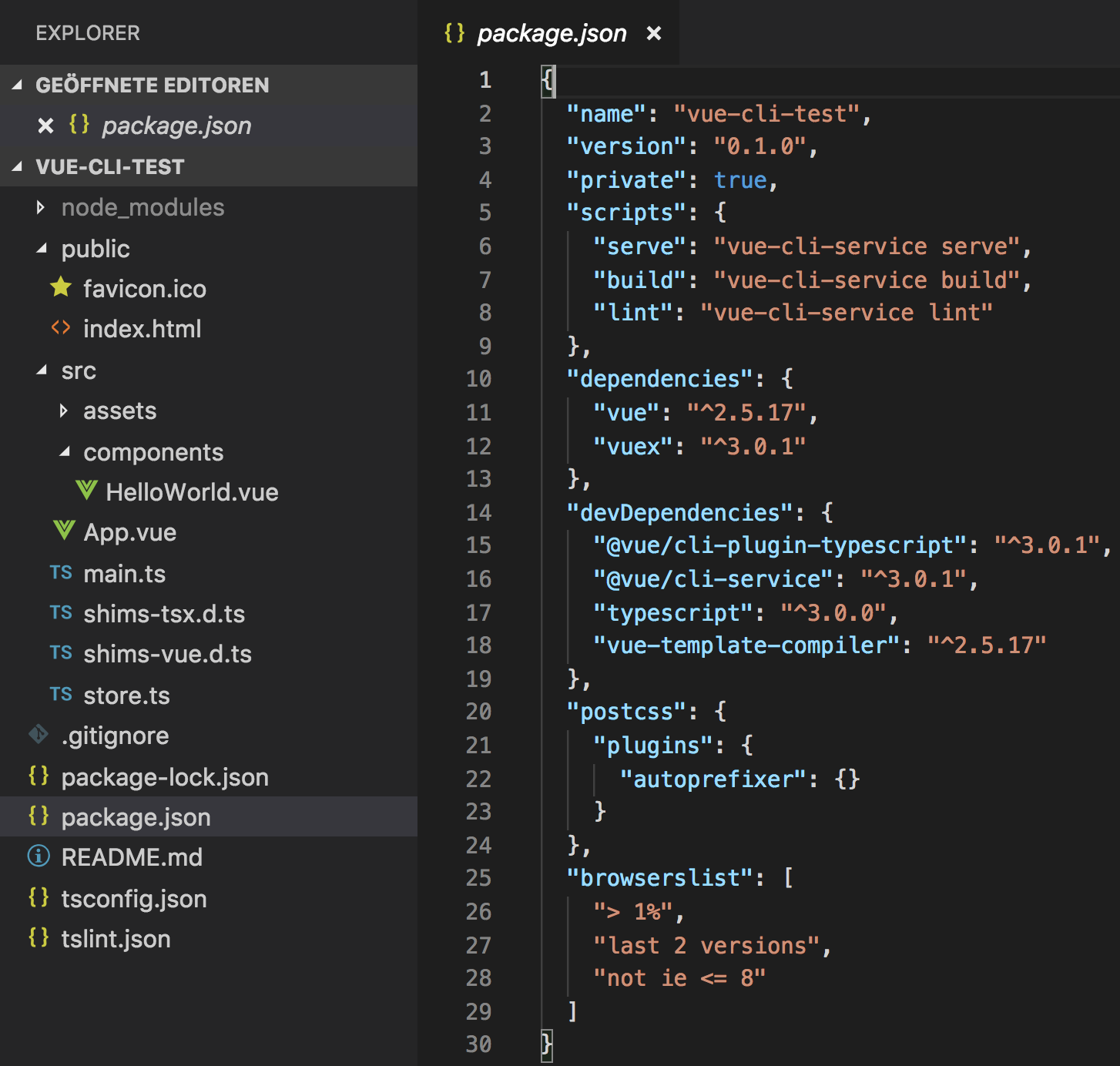
Außerdem arbeitet NPM mit modernen Web-Technologien wie Browserify, Webpack und begleitende Werkzeuge von Vue.js (s. bitte auch Kapitel 4.5.1.2) sehr gut zusammen, um in der Entwicklung einen guten Workflow zu schaffen.(Zitat: <https://vuejs.org/v2/guide/installation.html>).

Die Implementierung durch NPM hat aber ein Problem mit der Einstellung 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.js, 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.

4.2.2.3 CLI (Command line interface)

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

Die derzeitigen am beliebten JavaScript-Frameworks bieten alle eigene CLIs an (Create React App: https://github.com/facebook/create-react-app, Angular CLI: https://github.com/angular/angular-cli). Die CLIs von allen JavaScript-Frameworks machen im Prinzip aber das Gleiche. Im Folgenden wird das CLI von Vue.js dargestellt. Da wir uns für das Projekt EarlyDike für Vue.js entschieden haben.

Das JavaScript-Framework Vue.js bietet den Entwicklern ein begleitendes Werkzeug Vue CLI, das für Vue.js entwickelt wurde. Das Vue CLI kann einfach durch NPM (s. Abb. im Kapitel 4.2.2.2) ganz normal wie andere Dependencies installiert werden und bietet die Möglichkeiten, eine vordefinierte Struktur für ein Projekt zu erstellen (s. Abb. XXX).

(vue\_cli\_pre-defined\_structure: Vordefinierte Struktur von neu erstelltem Projekt mit extra Anforderungen an TypeScript Unterstützung und Vuex)

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 4.3) haben das Problem im Kapitel 4.2.2.2 wunderbar gelöst. 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. Leder 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.js stellt man Single file Component vor), die je nach Frameworks unterschiedlich sind, und auch andere Schwierigkeiten machen dieser Weg sehr schwierig für existierende Projekte. Die müssen meistens großartig umgeschrieben sogar komplett neu geschrieben werden.

4.3 Entscheidung der Art der Implementierung

Die folgende Tabelle 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.

|  | 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** |

(Tab. Erfüllungsgrad der angestrebten Ziele von unterschiedlichen Arten der Implementierung)

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

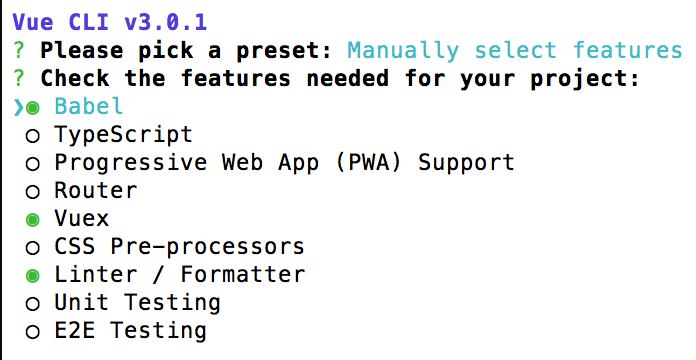
Die Implementierung durch CDN sowie NPM (ohne Unterstützung von anderen Web-Technologien, nur Framework) haben zwei Hauptprobleme. Einerseits ist eine gute Struktur schwer zu schaffen und das erhört die spätere Wartungsaufwand sowie Erweiterungsaufwand extrem, andererseits ist heutzutage ohne Unterstützung von Web-Technologien wie Babel, Webpack sehr unpraktisch (Zitat? Ol 5 update log?), ES6 basiert OpenLayer 5 anzuwenden.

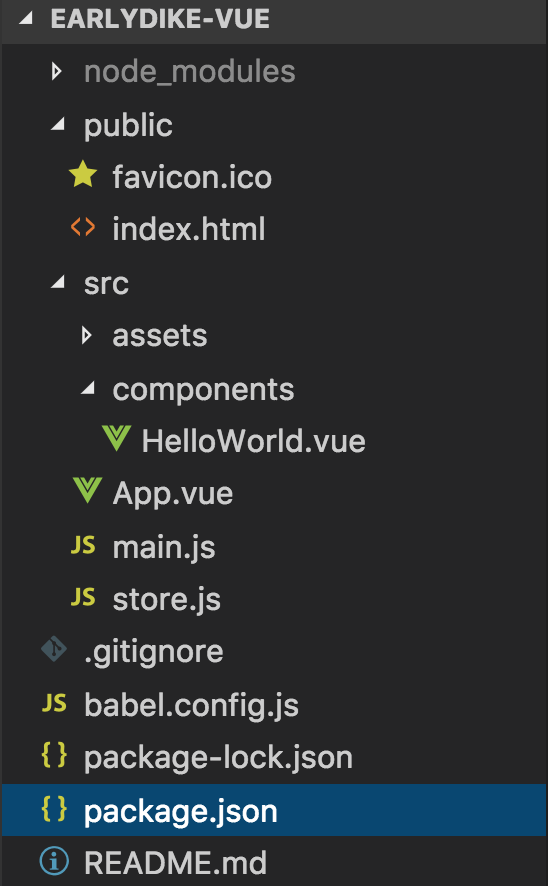
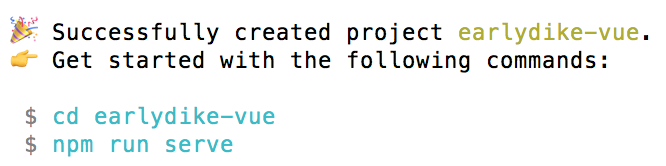
Die Implementierung durch NPM mit allen nötigen Web-Technologien haben im Prinzip bei der Entwicklung des Projekts die gleiche Kraft wie die Implementierung durch CLI. Aber CLI hat einen großen Vorteil im Vergleich dazu, nämlich die automatische Einstellung der komplexen Konfiguration von allen Web-Technologien.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

Ab nächsten Kapitel fängt der Wiederaufbau mit Vue (Implementiert durch Vue CLI) an. Die Struktur sowie die technischen Details werden auch im nachkommenden Kapiteln gründlich aufgeklärt.

4.4 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 Projekt 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.

4.5 Grundstruktur und Datenfluss von Projekt EarlyDike Geoportal

4.6 Technische Details

4.6.1 Details in globaler Ebene

4.6.1.1 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.

4.6.1.2 Die Vorteile der Anwendung von Vue CLI 3

4.6.1.3 Centralized State Management mit Vuex

4.6.1.4 UI-Verbesserung mit Vuetify

4.6.2 Details in Komponenten

4.6.2.1 Implementierung von unterschiedlicher Layers (Map.vue)

4.6.2.2 Implementierung von Auswahllisten der Layers (layerSelect Folder)

4.6.2.3 Implementierung von Informationstabs (tabs Folder)

4.7 Projekt übergreifend wiederverwendbare Dateien

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

4.7.2 Layerfabrik - GIAGS\_layer.js

4.7.3 Die Projektionsdefinitionen - GIAGS\_proj\_defs.js

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