

Hauptstudie Umsetzung VA

Niklas Liechti

Inhaltsverzeichnis

1. Management summary	1
2. Architektur	2
2.1. Frontend	2
2.2. Backend	2
3. Technologien	3
3.1. Frontend	3
3.2. Backend	7
3.3. Database	7
3.4. Deployment	7
3.5. Buildsystem	7
3.6. CI / CD	8
4. Schwierigkeiten bei der Umsetzung	9



1. Management summary



2. Architektur

Die Architektur ist eine recht einfach gehaltenes Frontend welches über eine Rest API mit dem Backend kommuniziert.

2.1. Frontend

Da ich das Frontend recht simple halten wollte, habe ich mich dafür entschieden die Komponenten fast komplett voneinander zu trennen. Die einzigen Komponenten, die mehrfach verwendet werden, sind die Navigationskomponenten. Auch verwende ich nicht den Vue router, daher wird ganz klassisch per "full page reload" navigiert.

2.2. Backend

Das Backend ist mehr oder weniger klassisch aufgebaut. Im Main.kt sind alle rest endpoints in einzelnen Funktionen definiert. Dort sind diese dann mit Controllern verlinkt.

Die Controller übernehmen dann die basic funktionalität und einen Teil der Logik.

Komplexe und ab kapselbare Logik ist dann in Services ausgelagert, die von den Controllern aufgerufen werden.

Der Persistenz Layer ist über sogenannte Repositories verfügbar. Dies können Implementationen sein, welche auf eine DB gehen, oder auf den Kubernetes cluster.



3. Technologien

3.1. Frontend

Aktuell ist das Frontend immer noch massiv von Javascript dominiert. Langsam kommen Konkurrenten wie [Web Assembly](#) hervor. Diese sind aber noch viel zu Jung und zu wenig verbreitet um momentan für eine Applikation ein zu setzten.

Daher habe ich mich dazu entschieden noch mit einem Javascript Framework weiter zu machen.

Frontend Javascript Frameworks sind seit Jahren im starken Wandel.

3.1.1. Framework

Im Frontend bereich habe ich von den "modernen" Frameworks nur Erfahrung in Angular.js, welches aber schon seit einigen Jahre eigentlich überholt ist.

Daher habe ich mich bei der Entscheidung an einen Blogpost von [dev.to](#) und Erfahrung von Arbeitskollegen angelehnt.



Angular 2+

Angular ist das nachfolgende Framework von angular.js, welches weiterhin von Google maintained wird.

Mit Angular habe ich minimale Erfahrung durch einen 1 monatigen Einsatz in einem kleineren Projekt.

Wie unten ersichtlich ist das Lernen und aufsetzen eines Angular Projektes relativ komplex. Grundsätzlich ist es extrem gut einsetzbar für grössere Singlepage apps. Die vorgegebene Struktur ist ein grosser Vorteil für Projekte mit vielen Personen, was in diesem Projekt aber nicht relevant ist.



Figure 1. Angular



Vue.js

Vue.js ist das neuste aller Frameworks. Es ist sehr einfach zu lernen und grundsätzlich sehr leichtgewichtig. Es ist ähnlich reaktiv wie react.js und nutzt Web components excessiv.



Figure 2. Vue.js



React.js

React.js ist eine möglichst schlank gehaltene Frontend Library, welche von Facebook maintained wird. Das Konzept ist wie der Name schon sagt reaktives Verhalten. Das heisst man beschreibt grundsätzlich welche Daten dargestellt werden sollen und das Framework kümmert sich dann um das "reagieren" auf changes des states.



Figure 3. React.js



3.1.2. Entscheidung

Das am einfachsten auf zu setzende Framework ist in meinem Fall Vue, da es direkt von meinem Backend Framework unterstützt wird. Da sich alle Frameworks für meinen Anwendungsfall nicht viel nehmen, habe ich mich für Vue.js entschieden.

3.2. Backend

Das Backend ist grundsätzlich seit jeher stabiler, da diese Plattform meist voll unter Kontrolle des betreibenden liegt. Diese wahl wurde in den Letzten Jahren aber auch immer flexibler, da mit Docker fast jede beliebige Betriebssystem Umgebung gebaut werden kann.

Die meiste Erfahrung habe ich selber mit Java EE Applikationen, welche aber hier ein zu grosses Kaliber wären.

Zusätzlich habe ich in letzter Zeit viel Positives über [Kotlin](#) gehört. Kotlin ist eine auf der Jvm basierte Sprache, die nahe an Java ist, aber sehr viel Syntactic Sugare anbietet und das Entwickeln von genau solchen kleinen Applikationen extrem einfach macht.

3.2.1. Framework

Als Framework habe ich mir nach kurzem recherchieren und nach Feedback von Arbeitskollegen [Javalin](#) ausgesucht.

Javalin ist ein leichtgewichtiges Microservice Framework, welches mit nativ Kotlin unterstützung kommt.

Auch bietet Javalin integrierter Support für Vue.js komponenten, was natürlich ein sehr praktischen Vorteil ist.

3.3. Database

Um Datenbank migrationen für so ein kleines Projekt zu Vermeiden setzte ich auf ein NoSQLDB.

Nach kurzer suche bin ich auf [Nitrite](#) gestossen

3.4. Deployment

Da das Deployment der Applikation auf einem Standard TBZ Cluster laufen muss, war das Deployment als Docker Image fix.

3.5. Buildsystem

Als Buildsystem für das Projekt habe ich [Gradle](#) mit der Kotlin DSL eingesetzt. Dieses Buildsystem ist sein einigen Jahren Standard im Java Universum. Mit gradle baue ich ein FatJar mit dem ShadowJar plugin, welches dann in ein Docker Image verpackt wird.



3.5.1. Dependencies

Gradle macht auch das Dependencie management. Die dependencies sind [hier](#) zu sehen.

3.6. CI / CD

Da bei einer so komplexen Umgebung ein Continous integration nur mit viel Aufwand möglich ist, gibt es dies hier nicht.

Für den Continous delivery Teil sorgt [Travis](#), welcher bei jedem Push auf das Repo ein latest Docker image baut und dieses zu Dockerhub pusht. Sobald ein Commit ge tagt ist, wird zusätzlich ein fixe Version released.



4. Schwierigkeiten bei der Umsetzung