LERNPORTFOLIO

Joel Maximilian Mai

Matrikelnummer 11118561

25. APRIL 2019

TH KOELN

WEB DEVELOPMENT

Inhalt

Workshop 04/04/2019

Zu Beginn des Workshops bekamen wir Problemszenarien vorgestellt – meine Gruppe und ich wählten das Szenario mit dem Kommunikationsproblem einer Familie, da es für uns näher an der Realität war und uns so leichter fiel Problemlösungen zu entwickeln. Für das Modul brauchen wir Kenntnisse in Javascript. Ich hatte bisher Javascript nur eingeschränkt im Abitur und die Basics in WBA1 – was bedeutet, dass ich mir einiges noch erarbeiten müsste. Jedoch konnte ich bei der Projektplanung mich viel einbringen, da ich als einziger von uns dreien Software Technik schon mit einer guten Note bestanden hatte und vorschlug nach den aus MCI bekannten Verfahren an das Problem ran zu gehen und das Problemszenario zu analysieren. Das wurde dann zu meinem Part. Während dessen befassten sich Moritz und Emre damit, welches Tool uns wohl zum fertigen Ergebnis verhelfen würde. Im Workshop erfuhren wir von den Vorteilen der Asynchronen Kommunikation. Um diese in unserem Projekt zu verwenden, nutzen wir das Framework RabbitMQ und Node JS.

Nachdem wir uns das Verhalten des Systems vorstellen konnten, stellten wir uns allerdings die Frage wie wir weiter vorgehen sollen. Natürlich könnten wir weiter Modelle bauen, stattdessen wollten wir uns aber lieber durch die verschiedenen Tutorials im Netz arbeiten, um ein besseres Verständnis für RabbitMQ zu bekommen.

Ich habe mich auch entschlossen meine Kenntnisse in Javascript zu verbessern. Des Weiteren fehlte uns das Verständnis für ein Domainmodell – Ich möchte mir also die Screencasts dazu angucken. Zusätzlich dazu, sollen wir uns die ersten Kapitel des Buchs „Reactive Design Patterns“ erarbeiten.

Unsere erste Idee zu dem System ist ein Service der Absprachen über diverse Kanäle ermöglicht und zusätzlich dazu in Events organisiert ist und diese dann dynamisch mit Push-Benachrichtigungen über veränderte Wetterbedingungen am Eventstandort informiert, so dass jeder Teilnehmer – egal welcher Kanal – informiert wird.

Erkenntnisse aus dem Buch

„Reactive Design Patterns“ – Kapitel 1 bis 2

- Responsive – Reaktion auf den User
- Resilient – Reaktion auf Versagen des Systems und das Überkommen und aktiv bleiben des Systems
- Elastic – Flexibel und Funktionstüchtig bei der Menge des Workloads
- Message-Driven – Reaktion auf verschiedene Inputs

Sharding-Patterns:

Das Kopieren einzelner Systemkomponenten, ermöglicht es dem System, selbst nach Überlastung oder Ausfälle einzelner Teilkomponenten verfügbar zu bleiben -> Replicate.

Hierbei unterscheidet man zwischen Aktiv-Passiven Replikaten, ein Replikat ist aktiv an Transaktionen beteiligt, während die Anderen nur Updates von dem Aktiven bekommen um immer auf dem neusten Stand zu sein, den Consensus-based Replikaten, bei denen alle Replikaten Updates annehmen und so immer konsistent sind, jedoch zeitweilig verzögert reagieren oder nicht verfügbar sind, den Optimistischen Replikaten, welche großteilig aktiv an das Netzwerk angeschlossen sind und Transaktionen durchführen, jedoch bei Konflikten Rollbacks durchführen müssen, daher sind diese optimistisch, und den Konfliktfreien Replikaten, bei welchen eine Zusammenführungsstrategie entwickelt und implementiert wird, sodass Konfliktfehler gar nicht auftreten können. Sie erfordern jedoch besondere Planung.

Responsives System entwickeln:

In der klassischen Entwicklung von Webservices setzt man Circuit Breaker Patterns ein, um ein System am laufen zu halten. Zum Beispiel zwischen Anwendungsebene des Users und dem Webserver aber auch zwischen Webserver und Backend-Services. Diese beobachten dann das Anfrageverhalten der Nutzer und schalten sich zwischen bei zu vielen Anfragen. Auch hilfreich ist, zwischen essenziellen und nichtessenziellen Features zu unterscheiden und zu priorisieren. Das hilft dann einen „Big Ball of Mud“ zu verhindern, bei dem die Kommunikationswege verstopfen und es zunehmend träge wird oder gar crashed. Eine spezielle Designform dabei ist: Message-Driven.

Integrating nonreactive Components:

Eine API-Einbindung kann bei Anfragen an das System zu Verzögerungen führen - wichtig ist für diesen Fall bei der Nutzung von APIs vorgesorgt zu haben. Zum Beispiel durch beobachten dieser Verzögerungen und einer für den Nutzer hilfreichen Nachricht an den Nutzer. Zum Beispiel: Temporäre Fehlermeldungen, Warnungen in der Antwort oder ihn zum späteren erneut-Versuch zu motivieren.

Understanding the traditional approach:

Um ein System zu optimieren ist es wichtig das Verhältnis von eingehenden Anfragen und aktiven Verbindungen richtig zu wählen. Sowohl der Ansatz mehr aktive Verbindungen zu halten als Neue anzunehmen als auch mehr anzunehmen als man halten kann, macht wenig Sinn. Die Lösung liegt zwischen den Extremen.

Analyzing latency with shared resource:

Ankommende Anfragen sollten immer in irgendeiner Form bearbeitet werden und nicht einfach verfallen. Hier für eignet sich eine Queue-Struktur. Diese kann dann Anfragen einreihen und sie auf eine Verarbeitung vorbereiten.
WICHTIG: Immer auf Anfragen-Overload vorbereiten und System beschützen.

Limiting maximum latency with a queue:

Ein einfacher erster Schritt ist es, bei dem Fall, dass die Datenbank nicht verfügbar ist, eine Anfrage mit NULL zu beantworten und eine absolute Absage der Anfrage zu schicken. Für die anderen Fälle kann man mit dem Einbau einer Explicit-Queue einige Error-Antworten vermeiden, in dem man diese dann bis zum Voll laufen in die Queue schiebt. Erst dann wird abgewiesen.

Exploiting parallelism:

Die meisten Programme erwarten, dass eine Funktion eine Antwort liefert. Wenn eine Funktion keine Antwort liefert und zum Beispiel einen Fehler wirft, kann es sein, dass das aufrufende Programm auch einen Fehler wirft. Dabei würde es schon helfen Unteraufgaben einzeln zu betrachten und parallel zu schalten. So läuft das Programm nur so lang wie die längste Unteraufgabe. Ist ein Ergebnis ein geworfener Fehler so hilft ein Try/ Catch – diesen Thread dann zu suspendieren würde auch Platz für neue schaffen.

Reacting to failure:

Resillience – Die Fähigkeit einer Substanz oder Objekt sich wieder zurück in eine vergange Form zu bringen und die Kraft sich schnell von Schwierigkeiten oder Anstrengungen zu erholen.
Distribute and compartmentalize – wichtige Daten in kopierter Form zu sichern und wichtige Kopien von Systemzentren auf verschiedene Stromnetze oder gar Länder oder Kontinente zu verteilen.

Compartmentalization and bulkheading:

Falls die Compartments nicht hundert prozentig voneinander getrennt sind, kann ein Systemfehler in einem von ihnen das gesamte System zum crashen bringen.

Losing strong consistency:

Nimm zwei von drei:

- Konsistenz – Alle Kopien beinhalten identischen Inhalt
- Hohe Verfügbarkeit der Daten für Updates
- Toleranz zu Netzwerkpartitionen

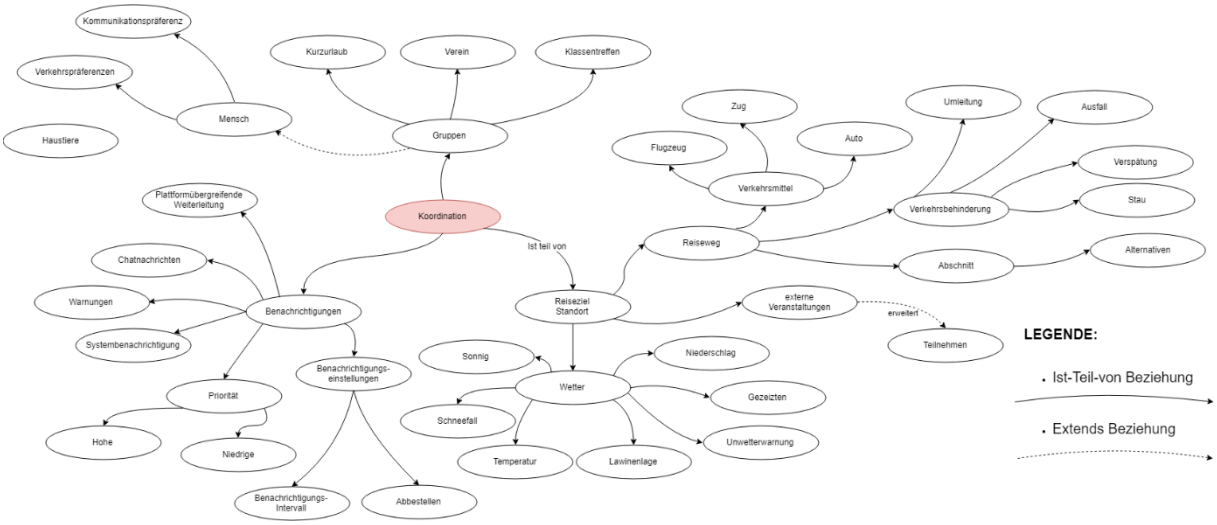
The need for Reactive Design Patterns:

- Essential Complexity – Ausgelöst durch ein Problem
- Incidential Complexity – Ausgelöst durch die Lösung

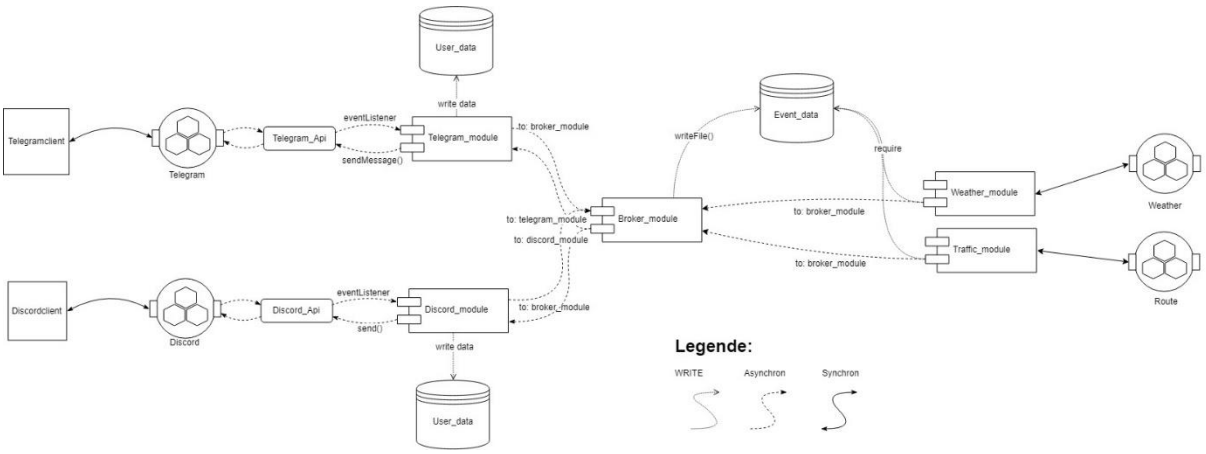
Workshop 11/04/2019

Zusammen mit Moritz hatten wir ja bereits erste Scripte zum Versenden und Empfangen von Nachrichten via Rabbit MQ geschrieben und so schonmal Basics verstanden. Leider waren die Prüfer immer noch nicht zufrieden mit unserem Domainmodell, trotz des Anschauens der Screencasts und Gespräch mit dem Professor war es immer noch fehlerhaft. Ich würde noch weitere Iterationen mit den Verbesserungsvorschlägen entwickeln. Aus dem Buch nahm ich einiges Vokabular mit und wollte es in weiteren Dokumentationen anwenden.

Domainmodell:



Architekturmodell:



Für den kommenden Workshop sollen wir weitere Kapitel des Buches lesen und für uns erarbeiten, das Domainmodell neu iterieren und unseren Code etwas ausarbeiten. Ich muss mir noch mehr zu JavaScript anschauen und lernen, da mir immer noch der „Flow“ fehlt, um flüssig am Code arbeiten zu können. Ich habe jedoch schon Wissen aus den anderen Teilmodulen anwenden können und das GitHub Repository angemessen eingerichtet. In wie fern wir dieses zum Zentrum machen oder uns über WhatsApp organisieren steht zu diesem Zeitpunkt noch nicht fest.

Erkenntnisse aus dem Buch 2

„Reactive Design Patterns“ – Kapitel 4 bis 6

Message Passing:

Die Unveränderlichkeit von Nachrichten hat oberste Priorität – sie können an andere Computer, Prozesse oder sogar innerhalb desselben Prozesses an Messagequeues versendet werden.

Vertical scalability:

Bei dieser Art der Kommunikation werden Sender und Empfänger entkoppelt. Der Sender weiß nicht wie der Empfänger mit der Nachricht umgeht oder sie verarbeitet.

Event-based vs. message-based:

Ein eventbasiertes System enthält Komponenten, die auf ein eintretendes Event mit Listnern warten, um darauf zu reagieren. Ein messagebasiertes System enthält Komponenten, die auf die Ankunft von Nachrichten wartet, um ihren Inhalt zu verarbeiten.

Synchronous vs. asynchronous:

Bei der synchronen Kommunikation müssen beide Partner bereit sein, miteinander zu kommunizieren. Einer stellt dann eine Anfrage und warten bis sie beantwortet wurde.

Bei der asynchronen Kommunikation kann der Sender einfach senden, ohne auf die Verfügbarkeit seines Partners zu warten. Das hat zur Folge, dass der Sender keine Wartezeit hat und sich den nächsten Aufgaben widmen kann.

Flow control:

Die Flow Control (Flusskontrolle) gibt dem Empfänger von Daten die Möglichkeit, den Sender über eine Überlastsituation (Overflow) zu informieren und darauf zu reagieren, beispielsweise die Übertragungsrate zu verringern oder das Senden einzustellen bis das Problem behoben ist, sodass keine Daten verloren gehen.

Delivery guarantees:

Da man immer davon ausgehen muss, dass Nachrichten verloren gehen könnten gibt es prinzipiell drei Möglichkeiten, um einen erfolgreichen Nachrichtenaustausch zu garantieren:

- Ein Request wird nur einmal gesendet – keine Wiederherstellung möglich
- Falls ein Request nicht beantwortet wurde, muss der Sender den Request behalten und sendet ihn erneut
- Ein Request darf genau einmal erhalten und bearbeitet werden, daher führt der Empfänger ein Log darüber welche Requests bereits bearbeitet wurden

Events as messages:

Eine eingehende Nachricht kann als Event angesehen und behandelt werden. Wenn man diese dann weiterleitet kann man von einem Event-Driven System sprechen. Dies wird als ‚natürlichste‘ Form der Kommunikation angesehen, da viele PC-Komponenten eh schon mit dieser Form arbeiten.

Synchronous message passing:

Solange keine asynchrone Kommunikation erforderlich ist, ist die asynchrone Übergabe von Nachrichten an entkoppelte Systeme unnötig und führt zu einem größeren Aufwand, als durch eine synchrone Weiterleitung erfolgen würde.

What is location transparency?:

Der Source-Code zum Senden einer Nachricht sieht gleich aus, ohne Rücksicht darauf ob der Empfänger es verarbeiten kann. Die Knoten können beschränkungslos sich Nachrichten schicken da Diese über die gleiche Art verschickt werden. Lediglich der Weg und Inhalt ist eigen.

Explicit message passing to the rescue:

Location Transparencys Ziel ist es den Lokalen und Globalen Nachrichtenaustausch zu vereinheitlichen und zu abstrahieren. Mit Location Transparency ist jede Nachricht potenziell eine übers Netz geschickte, aber da man auf eine Antwort nicht wartet ist hier alles erledigt. Nutzt man Location Transparency, so kann man die Software überall laufen lassen, ohne Verbindungsprobleme zu den anderen Teilservices zu haben.

Horizontal scalability:

Mehr Server einzurichten, die dasselbe tun kann, die Performance erhöhen.

Location transparency makes testing simpler:

Da man mehrere Server hat, die dasselbe tun kann, man sich einen rauspicken und Tests durchführen, ohne das System vom Netz zu nehmen.

Dynamic composition:

Im Optimalfall nutzt man die Parallelisierung von Abfragen durch Location Transparency und durch Implementierung von Fallbacks durch zusätzliche Horizontal Scalability erreicht man die beste Performance im Sinne der verteilten Systeme.

Dependencies vs. descendant modules:

„Ownership“ ist ein wichtiger Punkt. Es sollte ein Modul vorhanden sein von den anderen Modulen auf Funktionen des ersten Moduls zugreifen wollen, diese sind dann abhängig oder haben Abhängigkeiten.

Building your own big corporation:

Ohne ordentliche Trennung der Segmente hinsichtlich ihrer Verantwortung und Aufgabe entstehen Konflikte zwischen ihnen. Aufgaben werden von höherem Rang übernommen, wenn der untere Ausfällt.

Advantages of specification and testing:

Jede Aufgabe eines Moduls muss eindeutig sein, damit man sie isoliert testen kann. So entsteht, bei erfolgreichem Abschluss der Tests ein funktionierendes System. Dadurch wird ein besseres Design erzielt.

Horizontal and vertical scalability:

Es ist sinnvoll mehrere aktive Teilnehmer zu einem Topic zu haben, da man so wesentlich wirtschaftlicher vorrangeht als für alles eine eigene Instanz zu haben.

Workshop 25/04/2019

Bisher konnte ich meine Kenntnisse über JavaScript deutlich verbessern, aber ich würde immer noch nicht davon sprechen ein „Pro“ zu sein. Unsere API-Einbindung war dank Moritz Erfahrung in Telegram-Bots erfolgreich und das Einbinden der Wetter-API verlief auch reibungslos. Ich nutzte die Chance das wir mit GitHub arbeiten, um hilfreiche Issues einzubauen. Heute im Workshop möchten wir unser bisheriges Script um nützliche Features und Message-Driven-Kommunikation erweitern. Für den nächsten Workshop sollen wir einen Artikel über Schema.org lesen und kritisch begutachten. Und unser Domainmodell scheint immer noch nicht korrekt also auch hier nochmal ausarbeiten.

Erkenntnisse aus dem Schema.org Artikel

„Evolution of Structured Data on the Web“ – R.V. Guha

Um das Web nach Informationen zu durchsuchen muss man die Markup Language HTML in ein für Crawler interpretierbares Format übersetzen und in einer geeigneten Form strukturieren.

Warum war XML nicht geeignet?

Weil XML für Maschinen und nicht für Menschen optimiert war…

Schema.org wurde von den großen Suchmaschinen der Zeit geformt. Sie bildeten ein Klassenbasiertes System was die Strukturierung und Interpretierung von Daten vereinfacht. Wie sie die Daten interpretieren, liegt jedoch immer noch bei jeder Suchmaschine selbst.

Schema.org korrigiert seine Knoten im Graph nur, wenn genug Beschwerden ankommen, was zur Folge hat, dass die präsentierten Daten nicht garantiert korrekt sind. Des Weiteren wird das Vokabular von Schema.org nur selten nennenswert aufgeräumt.

Was man aus dem Verhalten von Schema.org lernen kann ist, dass man es Entwicklern oder Autoren leichter machen sollte an der Vokabular Bildung teilzuhaben, keine langen Spezifikationen, sondern lieber Schablonen mit Dokumentationen, und Komplexität sollte mit der Zeit hinzugefügt werden.

Alternative zu Schema.org

Open Graph vs Schema.org

Bei meiner Internet Recherche stoß ich immer wieder darauf, dass man die Beiden Markups wohl nur schwer vergleichen könnte.. Während Open Graph für Social Media Plattformen, ähnlich wie RSS, eine kompakte Kurzvorschau liefert, so ist Schema.org für Suchmaschinen. Das heißt, wenn man sich entscheiden möchte eins von Beiden oder sogar Beides zu nutzen, sollte man sich fragen was man damit bezwecken möchte. Also eine bessere Suchmaschinenplatzierung oder besser Social Media Nutzung, wo bei manche Quellen behaupten, dass die gemeinsame Nutzung von Nachteil sei.

Allgemein, kann man sagen, dass das Thema momentan ziemlich umstritten ist, was man Nutzen sollte. Bei meinen Recherchen fand ich keine einheitliche Meinung vor, mit Ausnahme davon, dass Schema.org das Internet „auffrisst“.

Workshop 02/05/2019

Zu diesem Zeitpunkt habe ich die eventbasierte Kommunikation verstanden und weitere NodeJS Basics erlernen können, das Coden läuft schon flüssiger. Um unser Projekt zu managen organisieren wir unser Projekt in GitHub mit Karten und Issues. Unser Wiki hat auch einen neuen Anstrich bekommen. Ich würde gerne mehr Zeit haben, dieses etwas schicker und sauberer zu halten. Momentan wirkt alles zeitlich etwas knapp und es ist ziemlich demotivierend zu sehen, dass wir anscheinend kein Modell auf die Beine bekommen was so abgenommen wird und scheinbar alle Gruppen an denselben Schwierigkeiten scheitern wie wir. Aber ich bin guter Dinge, dass es wie immer am Ende doch klappt. Einfach weiter dransetzen und verstehen lernen. Ich denke nach dem heutigen Feedback sollten alle Fragen und Missverständnisse geklärt sein, auch wenn ich den Workload immer noch als ziemlich umfangend empfinde. Für alles eine Doku/Lerntagebuch anzulegen und das Repository clean zu halten ist zeitaufwendig… Und die brauch ich eigentlich gerade für das Selbststudium und Projektarbeit. Die anderen Module sind leider zurzeit auch nicht weniger anspruchsvoll.

Ich habe mich entschieden das Lerntagebuch noch einmal zu überarbeiten und es mehr zu kürzen/ in ganzen Sätzen/Erkenntnissen zu organisieren – auch das frisst Zeit.

Als neues Lernziel habe ich mir das Domain- und Architekturmodell vorgenommen und es zusammen mit Emre zu verbessern. Dann steht noch das implementieren des neuen Prototyps auf dem Plan, so wie das Aufräumen und optimieren des Repository…

Workshop 09/05/2019

Heute haben wir kleinere Bugs gefixed und an den Modellen nach dem heutigen Feedback gearbeitet. Das Repository haben wir auch noch einmal durchgearbeitet. Ich konnte ein besseres Verständnis von Javascript erwerben und von mir behaupten, wieder auf meinem alten Niveau zu sein.

Open Data Recherche

SELBSTSTUDIUM

Open Data (Offene Daten) sind Daten, welche von jedem benutzbar, teilbar und weiterverwendbar sind. Nur der Urheber kann das Einschränken.

- BENUTZBARKEIT/VERFÜGBARKEIT: DIE DATEN MÜSSEN IMMER VOLLSTÄNDIG SEIN. ANFALLENDE KOSTEN MÜSSEN SICH AUF EIN MINIMUM BESCHRÄNKEN, DAHER IST EIN DOWNLOAD AUS DEM INTERNET AM IDEALSTEN.
- WEITERVERWENDBARKEIT: DIE DATEN MÜSSEN UNTER BEDINGUNGEN ZUR VERFÜGUNG GESTELLT WERDEN, DIE EINE WIEDERVERWENDUNG UND WEITERGABE ERMÖGLICHEN, INKLUSIVE EINER VERWENDUNG DER DATEN ZUSAMMEN MIT DATENSÄTZEN AUS ANDEREN QUELLEN.
- UNIVERSELLE BETEILIGUNG: JEDER MUSS IN DER LAGE SEIN, DIE DATEN ZU NUTZEN, ZU VERARBEITEN UND WEITERZUVERTEILEN.

Open Data ist mir noch ein Rätsel und was ich davon halten soll. Es gibt sicherlich viele interessante Ansätze, aber im Großen und Ganzen ist mir noch unklar warum es so viele Ansätze gibt und man nicht einen Gemeinsamen, unter Aufsicht von Gremien, durchsetzt.

- SOLLTEN OFFENE DATEN VOM INSTANZ/STAAT KONTROLLIERT WERDEN?
  - JEDER MUSS SELBST ENTSCHEIDEN OB MAN DER QUELLE GLAUBT…
    - FOKUS AUF BEREITSTELLEN UND WENIGER AUF DIE KORREKTHEIT
  - VORTEIL IST: JEDER KANN SCHREIBEN UND LESEN/WEITERVERARBEITEN.
- SOLLTE ES EINEN STANDARD GEBEN ODER MEHRERE VARIANTEN?
  - SCHWIERIG WEIL VERSCHIEDENE NUTZUNGSGEBIETE – MASCHINENPARSING ODER MENSCHEN LESEN
  - KOMPROMISS DAS BESTE?
    - VOKABULAR IST JA SCHON VEREINHEITLICHT ODER ANGELEGT – TATSÄCHLICHE SYNTAKTISCHE REPRÄSENTATION DARF UNTERSCHIEDLICH BLEIBEN
- WAS IST DAS DATA-PORTAL DER EU
  - VIELE METADATEN AUS DEM EU-SEKTOR ALS ZIEL DIE VERÖFFENTLICHUNG UND TRANSPARENZ VON ÖFFENTLICHEN DATEN ZU VERBESSERN.

Finaler Recap – Was habe ich in diesem Workshop gelernt

Im Wesentlichen habe ich die Nutzungsmöglichkeiten für Verteilte Systeme und deren Vorteile verinnerlicht und meine Kenntnisse in Javascript, JSON und NodeJS wieder auf einen guten Stand gebracht. Open Data ist nach der Diskussion für mich immer noch ein komplexes Thema, das weiteres Digging benötigt. Die Kritik an Schema.org konnte ich voll und ganz nachvollziehen, jedoch muss ich sagen, dass was mir am meisten gebracht und Spaß gemacht hat war, die Zusammenarbeit im Team und die Organisation im GitHub.