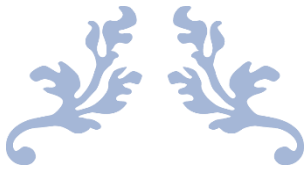

LERNPORTFOLIO

Moritz Dahlke

Matrikelnummer 11117538

INHALT

WORKSHOP AM 04.04.19

ZIELE DES KURSES „FRAMEWORKS, DIENSTE UND DATEN IM WEB“:

- Gezielter, selbstständiger Wissenserwerb mittels verschiedener Quellen
- Auswahl und Implementierung von Frameworks für serverseitige Anwendungen
- Verständnis über asynchrone, serverseitige Kommunikation und Framework Standards
- Verständnis über offene Daten im Web

VORHANDENES VORWISSEN:

Die Wahl des Szenarios wurde vor allem wegen vorherigen Arbeiten mit serverseitigem NodeJS und Telegram-Bots getroffen. Ein dafür benötigtes, brauchbares Skillset in Javascript war bereits vorhanden und konnte angewendet werden.

Es war kein Vorwissen in Sachen Kommunikationsprotokolle wie XMPP oder RabbitMQ vorhanden, was auf jeden Fall durchs Selbststudium geändert werden sollte.

Grundlagen der Softwaremodellierung waren aus dem Fach Softwaretechnik 1 bereits bekannt, allerdings gab es Schwierigkeiten beim Erstellen eines Domänenmodells, da es unterschiedliche Varianten zu geben scheint.

PLAN UND FORTSCHRITT:

- Erstellung eines angemessenen Domänenmodells als Voraussetzung der Zielsetzung

Die Erstellung eines Domänenmodells benötigte mehrere Anläufe, jedoch konnte durch etwas Recherche und Kommunikation mit dem Kursleiter ein entsprechendes Modell angefertigt werden, mit dem wir zufrieden letztlich zufrieden waren.

- Zielsetzung

Die Zielsetzung für den Workshop am 11.04.2019 bestand aus der Aufbereitung der vorgeschlagenen Kapitel des Buches „Reactive Design Patterns“, sowie NodeJS (Javascript) und dem Erhalt von Informationen über Kommunikationsprotokolle wie XMPP oder RabbitMQ, um sich später auf eines davon festlegen zu können. Es standen nur diese Protokolle zur Auswahl, da diese einen relativ hohen Stellenwert besitzen.

SELBSTSTUDIUM ZUM THEMA „REACTIVE DESIGN“ (KAPITEL 1 & 2)

Das Selbststudium erfolgte in Gruppenarbeit, um ein ausreichendes Verständnis zu ermöglichen.
Aus diesem Grund sind die folgenden Ergebnisse bei anderen Gruppenmitgliedern ähnlich.

COMPUTER STANDARDS:

- Responsive = Reaktion auf den User
- Resilient = Reaktion auf Versagen des Systems
- Elastic = Flexibel und Funktionstüchtig bei der Menge des Workloads
- Message-Driven = Reaktion auf verschiedene Inputs

SHARDING-PATTERNS:

- Replikation/Kopie eines Systems um Verfügbarkeit zu garantieren -> Replicate
- Active-Passive Replication:
    - Festlegen welches (aktive) Replikat Updates akzeptiert
        - Leitet Updates an die Anderen (passiven) weiter
        - Bei Ausfall übernimmt ein passives Replikat
- Consensus-based Multiple-master Replication:
    - Jedes Update wird von allen Replika bestätigt und übernommen
    - Damit sind zwar alle konsistent (Consistent), aber anfällig für Verzögerungen (Latency) oder nicht verfügbar (Availability) auf Grund des höheren Workloads
- Optimistic Replication (with conflict detection and resolution):
    - Mehrere aktive Replika verbreiten Updates und führen Transaktionen zurück, wenn Konflikte auftreten oder verwerfliche Updates verworfen werden, die während einer Netzwerkpartition durchgeführt wurden
- Conflict Free Replication:
    - Dieser Ansatz schreibt Zusammenführungsstrategien so vor, dass Konflikte nicht definitionsgemäß entstehen können, jedoch nur die eventuelle Konsistenz erfordert und besondere Sorgfalt bei der Erstellung des Datenmodells erforderlich ist

RESPONSIVES SYSTEM ENTWICKELN:

- Anwendung läuft lokal und speichert Daten lokal
- Synchronisiert mit Server/-n
    - Erzeugt Spannung zwischen dem Wunsch das System zu verteilen und responsiv zu bleiben
- Circuit Breaker Patterns
    - Beobachtet das Anfrageverhalten des Nutzers und schaltet sich zwischen, wenn zu viele Anfragen gemacht werden
    - Zusätzliche Features:
        - Flow Control
        - Backlogs
    - Anwendungsbereich von Circuit Breaker Patterns
        - Zwischen Anwendungsebene des Users und Webserver
        - Zwischen Webserver und Backend Services
- Wichtig ist auch die Einstufung des Features: Essential / Non-Essential
- Avoiding the ball of mud:
    - Klassische Visualisierung eines Systems ist Frontend Backend
    - Jedoch durch die Erweiterung/Verteilung des Backends auf verschiedene Services
        - Kann ein Big ball of mud entstehen wenn die Kommunikationswege
        - Nicht frei sind
- Spezielle Designform: Message-Flow / Message-Driven

INTEGRATING NONREACTIVE COMPONENTS:

- Device Drivers
- API (Normalerweise synchron – Kann Funktionalität der App beinflussen)
- Daher wichtig: Resource-Management Patterns
    - Ständiger retrofit (Ausbau) des Systems
    - Extra Threads, Prozesse oder Maschinen, wenn nötig
    - Festlegen was bei einem Overload passieren soll
    - Auskapseln von APIs und sie auslagern um Crashes zu vermeiden
    - IPC = Inter Process Communication
        - Pipes
        - Sockets
        - Shared Memory
- Verzögerungen beobachten und im Falle dieser:
    - Temporäre Fehler Meldungen
    - Warnungen in der Antwort
    - Nutzer benachrichtigen, dass
        - Später erneut versuchen
        - Oder über die Verzögerung
- Reactive Manifesto

UNDERSTANDING THE TRADITIONAL APPROACH:

- Wenn Sie die Leistung eines Systems wie dieses optimieren, ist einer der Schlüsselparameter das Verhältnis von Anforderungsthreads zu Einträgen des Verbindungspools. Es macht nicht viel Sinn, den Verbindungspool größer als den Request-Thread-Pool zu machen.
- Verbindungspool zu klein -> Bottleneck (Engpass beim Transport von Daten)
- Die beste Antwort für eine gegebene Last liegt irgendwo zwischen den Extremen. Der nächste Abschnitt befasst sich mit dem Finden eines Gleichgewichts.

ANALYZING LATENCY WITH SHARED RESOURCE:

- Bei Anfragen, die darauf warten prozessiert zu werden, benötigt man immer eine Art von Queue Data Structure (wie Stacks, aber an beiden Enden offen)
- Lösungsansätze:
    - Controller
    - TCP Buffer
    - Cache
    - Fallback
- WICHTIG: Immer auf Anfragen Overload vorbereiten und System beschützen

LIMITING MAXIMUM LATENCY WITH A QUEUE:

- Erster einfacher Schritt: If no database connection available -> Return null
- Einführung einer Explicit Queue
    - Statt direkt einen Error zu senden können hier noch Anfragen zwischen geparkt werden
    - Erst wenn diese Queue voll ist wird abgewiesen

EXPLOITING PARALLELISM:

- Die meisten Programme erwarten, dass ihre Funktionen eine Antwort liefern, tun sie dies nicht funktioniert das ganze Programm nicht
- Lösung:
    - Sub-Tasks einzeln betrachten und parallel schalten -> Die Aufgabe dauert nur so lange wie die Längste Sub-Task
    - Und das Ergebnis nicht als zwingend passend betrachten -> Try Catch
        - Fehlerwerfende Threads suspendieren, um den Platz für neue Threads freizugeben

THE LIMITS OF PARALLEL EXECUTION:

- Um von diesem Wachstum zu profitieren, müssen Sie die Berechnungen sogar auf einer einzigen Maschine verteilen.
- Ein herkömmlicher Ansatz ist, wenn ein gemeinsamer State verwendet wird, der auf gegenseitigem Ausschluss durch Sperren basiert, werden die Kosten für die Koordinierung zwischen Kernen sehr CPU-bedeutend.

REACTING TO FAILURE

- Software will fail
- Hardware will fail
- Humans will fail
- Timeout is Failure
- Resillience:
    - Die Fähigkeit einer Substanz oder Objekt, sich wieder zurück in eine vergangene Form sich zu bringen
    - Die Kraft sich schnell von Schwierigkeiten oder Anstrengungen zu erholen
- Wichtige Teilaufgaben eines Systems delegieren
- Distribute and Compartmentalize (Verteilen und Gliedern)
- Wichtige Daten in kopierter Form sichern
- Wichtige Kopien von Systemzentren auf verschiedene Stromnetze oder gar Länder oder Kontinente 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.

USING CIRCUIT BREAKER

- Wenn die Antwort einer Anfrage bereits nicht mehr benötigt wird, ist es sinnlos sie überhaupt noch zu bearbeiten
- Falls der Circuit Breaker eine Verzögerung im Systemablauf feststellt kann er Anfragen von akzeptiere-alle auf wartet-hier-und-kommt-nach-der-wartezeit-dran oder gar ich-nehme-nichts-mehr-an umstellen um dem System Zeit zu geben sich zu erholen und die eigenen Warteschleifen zu leeren
- Wie viele Re-Connection Versuche ergeben Sinn?
    - Wie lange sollte man zwischen ihnen warten?

LOSING STRONG CONSISTENCY

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

BASE

- Verteilte Systeme sind meist auf verschiedenen Prinzipien gebaut
- Eins davon heißt BASE
    - Basically Available
    - Soft State (Ein Zustand der aktiv versucht erreicht zu werden, statt von einem Standard Zustand auszugehen)
    - Eventually Consistent
- Der Letzte Punkt bedeutet, dass zwar zu manchen Zeitpunkten ein inkonsistenter Zustand herrschen kann, dieser jedoch mit der Zeit wieder konsistent wird.
- BASE setzt sich zusammen aus
    - Associative
        - Jede Aktion kann in Stapeln erfolgen (Assoziativ)
    - Commutative
        - Jede Aktion kann in jeder Reihenfolge erfolgen (Kommutativ)
    - Idempotent
        - Jede Aktion kann beliebig oft durchgeführt werden
    - Distributed

ACCEPTING UPDATES

- Wichtig hierfür sind CRDTs (Conflict-Free-Replicated-Data Types)
- Wie man zwei Dokumente wieder konsistent bekommt
    - Die beiden Änderungsketten nicht simultan bearbeiten, sondern über einander legen
    - Erst die eine Kette anwenden und dann die Andere
    - Bis beide Partitionen „geheilt“ sind
- Wenn das System offline oder einfach unerreichbar ist, ist die Lösung, dem Nutzer immer noch manche Features anzubieten.
- Man könnte auch das Ganze so sehen, dass das System auf einen ungefähren Zustand wechselt bis eine Verbindung wieder hergestellt werden konnte

THE NEED FOR REACTIVE DESIGN PATTERNS

- SOA = Service-Oriented Architecture
    - Zusammensetzung (Orchestrierung) von mehreren Services
- Managing Complexity
    - Essential Complexity = Ausgelöst durch ein Problem
    - Incidential Complexity = Ausgelöst durch die Lösung
    - Message-Oriented Development führt genauso zu einem konsistenten Service wie synchrone Entwicklungen und weil sie deutlich leichter zu implementieren sind, werden sie immer öfter benutzt

WORKSHOP AM 11.04.19

PLAN UND FORTSCHRITT:

- Festlegung des Kommunikationsprotokolls: RabbitMQ
    - Bietet umfassende Dokumentation und Tutorials
    - Anwendung in vielen verschiedenen Programmiersprachen möglich
        - Eine davon: Javascript via NodeJS

Nach der Installation eines lokalen RabbitMQ Servers war ein kleines Script mit „Sender“ und „Empfänger“ mittels dokumentierter Tutorials und vorhandenen Kenntnissen schnell lauffähig. Die Tutorials wurden im Selbststudium gelesen, verstanden, sowie umgesetzt und nicht einfach kopiert. Das hat dazu geführt, dass der „Code“ sehr leicht anzupassen war.

- Festlegung von Lernzielen bis zum 25.04.2019

Nach der Erstellung des Tutorial-Scripts war beispielsweise nicht auf Anhieb ersichtlich, wie der „Empfänger“ wieder mit dem „Sender“ kommunizieren kann. Dies war darauf zurückzuführen, dass lange Zeit unklar war, was der Prototyp des Systems am Ende genau darstellen und wie dieser umgesetzt werden sollte.

Das Ziel zum Workshop am 25.04.2019 sollte ein „Prototyp“ sein, der bereits eine simple, realistische Kommunikationskette darstellt. Das sollte wie folgt aussehen:

- Kommunikation einzelner, unabhängiger „Service“-Module mit einem „Hauptmodul“
    - Server-Modul (Hauptmodul)
    - Wetter-Modul (Servicemodul)
    - Telegram-Modul (Servicemodul)
- Keine Kommunikation unter Servicemodulen – ausschließlich mit dem Hauptmodul
- Kommunikation des Hauptmoduls mit den jeweiligen Servicemodulen

Dieser „Prototyp“ sollte also ein Hauptmodul starten, welches auf die Nachrichten von Service-Modulen wartet. Ein Wetter-Modul prüft dafür beispielsweise jede Minute die Temperatur am Veranstaltungsort und sendet eine Nachricht an das Hauptmodul, wenn eine Veränderung vorliegt. Des Weiteren sollte ein Telegram-Modul das Ganze etwas realitätsnaher gestalten. In diesem Fall ermöglicht also ein Telegram-Bot die Kommunikation mit dem Hauptmodul.

WORKSHOP AM 25.04.19 (MEILENSTEIN 1)

AUSGANG:

Der geplante „Prototyp“ wurde bis zum 25.04.2019 fertiggestellt. Während des intensiven Beschäftigens mit der Materie ergeben sich durch Überblick und Fokus plötzlich auch einfache Dinge, wie zum Beispiel die Antwort auf die Frage wie ein „Empfänger“ auch wieder eine Nachricht an den „Sender“ leiten kann. Da wundert man sich gleich, wieso solch eine Frage überhaupt aufkommt.

Das Einbinden einer Wetter-API und das Aufsetzen eines Telegram-Bots war durch vorhandenes Vorwissen absolut kein Problem.

PLAN UND FORTSCHRITT:

- Anlegen eines Lernportfolios

Das zuvor noch nie ein Lernportfolio angelegt wurde führte zu leichten Schwierigkeiten und zu einem dementsprechenden Ergebnis. Der Fokus lag durch die vorgegebene Aufgabenstellung auf der Dokumentation zum Thema „Reactive Design“.

- Erlangung eines ersten Verständnisses für das Thema "offene Daten"
    - Daten, die von jedem für jegliche Zwecke verwendet werden können

SELBSTSTUDIUM ZUM THEMA „REACTIVE DESIGN“ (KAPITEL 4-6)

KAPITEL 4:

4.1 MESSAGES

- Immutability (Unveränderlichkeit) hat oberste Priorität
- Messages werden in Form von Message Queues versendet, um bearbeitet werden zu können
    - Anders als bei Shared-Memory Systemen gibt es keinen gemeinsamen Speicher
- Messages werden versendet:
    - An andere Computer
    - An andere Prozesse
    - Innerhalb des eigenen Prozesses

4.2 VERTICAL SCALABILITY

- Message Passing (Nachrichtenaustausch) entkoppelt Sender und Empfänger
    - Dadurch muss der Sender nicht wissen, wie der Empfänger mit der Message (Nachricht) umgeht und sie verarbeitet oder ob der Empfänger die Message parallel verarbeiten kann oder nicht -> Muss also keine „Rücksicht nehmen“
    - So kann beispielsweise die Rechenkapazität des Empfängers unabhängig vom Sender beliebig ausgebaut werden und die Verarbeitung auf mehrere Einheiten verteilt werden, um effizienter zu sein.

4.3 EVENT-BASED VS. MESSAGE-BASED

- Event-Driven
    - Ein Event ist ein ausgelöstes Signal sobald ein Ereignis eintritt.
    - In einem event-basierten System gibt es sog. Listener die aufgerufen werden sobald ein Ereignis eintritt auftritt.
- Message-Driven
    - Eine Message (Nachricht) sind Daten, die an einen bestimmten Empfänger gesendet werden
    - Ein message-basiertes System wartet auf die Ankunft von Nachrichten, um darauf zu reagieren.

4.4 SYNCHRONOUS VS. ASYNCHRONOUS

- Synchrone Kommunikation
    - Beide Parteien müssen bereits sein, miteinander zu kommunizieren
        - Sender stellt Anfrage und wartet auf eine Antwort
            - Blockiert den Prozess, bis die Kommunikation abgeschlossen ist
- Asynchrone Kommunikation
    - Sender kann senden ohne, dass der Empfänger dafür bereit ist
        - Sender stellt Anfrage, wartet aber nicht auf eine Antwort
            - Blockiert den Prozess nicht, wodurch die nächste Aufgabe durchgeführt werden kann

4.5 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.

4.6 DELIVERY GUARANTEES

- Man muss immer von der Möglichkeit ausgehen, dass Nachrichten verloren gehen können
    - Auch in synchronen Systemen
- Prinzipiell gibt es die folgenden Möglichkeiten zum Garantieren von Nachrichtenaustausch
    - At-Most-Once Delivery
        - Jeder Request wird einmal gesendet, wenn er verloren geht oder der Empfänger dabei scheitert ihn zu verarbeiten gibt es keine Möglichkeit zur Wiederherstellung
    - At-Least-Once Delivery
        - Der Versuch, eine Verarbeitung eines Requests zu garantieren
        - Benötigt eine Bestätigung des Empfängers
        - Der Sender muss den Request behalten, um ihn erneut zu senden, falls keine Bestätigung des Empfängers eintrifft
        - Empfänger können einen Request mehrmals erhalten, wenn das Senden der Bestätigung fehlschlägt und der Sender ihn anschließend erneut sendet
    - Exactly-Once Delivery
        - Ein Request muss und darf nur genau einmal verarbeitet werden
        - Der Empfänger muss Informationen darüber haben, welche Requests er bereits bearbeitet hat

4.7 EVENTS AS MESSAGES

- Eine Nachricht, die versendet wird und ankommt kann man als Event sehen
- Da sie weitergeleitet werden können, spricht man auch von einem event-basierten System, da die Events die einzelnen Komponenten verbinden.
- Da viele PC-Komponenten sowieso schon mit Messages/Events arbeiten ist es so gesehen die „natürlichste“ Form der Kommunikation unabhängiger Komponenten.

4.8 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.

KAPITEL 5:

Warum sich darauf beschränken nur Threads zu trennen?

- Systeme auch trennen
- Dies eröffnet neue Blickwinkel für Performance der Systeme

5.1 WHAT IS LOCATION TRANSPARENCY?

- 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.

5.2 THE FALLACY OF TRANSPARENT REMOTING

- Schon lange ist das Ziel von Computer Netzen, die Nachrichten so anzulegen, dass sie sowohl lokal als auch remote (also aus der Ferne) gleichermaßen verschickt und verarbeitet werden können -> Einheitlichkeit
    - Transparent Remoting
- Das Problem ist, dass ein Funktionsaufruf lokal nur zu Exception oder Resultat führen kann, jedoch über ein Netzwerk deutlich mehr schief gehen kann.
    - Nicht richtig aufgerufen
    - Verloren gegangen
    - Fehlerhaft versendet
- Regelfalllösung: Timeout Exception erhöhen
- Performanceproblem ist: Der Aufruf muss erst versendet werden und dann auf die Antwort gewartet werden
- Performanceproblem: Dateigröße, die übers Netz gesendet werden kann, ist kleiner als die Lokale

5.3 EXPLICIT MESSAGE PASSING TO THE RESCUE

- Location Transparency 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
- Mit Transparent Remoting hofft man bei jedem Aufruf auf eine erfolgreiche Antwort
- Mit Location Transparency kann man die Software überall laufen lassen ohne Probleme mit der Verbindung zu haben zu den anderen Teil Services

5.4 OPTIMIZATION OF LOCAL MESSAGE PASSING

- Um Verzögerungen zu verringern kann man den Nachrichtenaustausch hier auch über Referenzen vollziehen
- Falls Lokal implementiert, ist es meist sinnvoller sich and den oben genannten Punkt zu orientieren und nicht mit Nachrichtenaustausch zu arbeiten – da man hier die sende und Empfangs Zeit sparen würde

5.5 MESSAGE LOSS

- Nachrichten, die über Netzwerke versendet werden, können über viel mehr Wege verloren gehen
- Falls nicht alle Antworten einer Anfrage wieder ankommen
    - Überlegen was getan werden soll
        - Wiederstand des Systems gesteigert

5.6 HORIZONTAL SCALABILITY

- Mehr Server einzurichten, die dasselbe tun kann, die Performance erhöhen
- Da Location Transparency
    - Unabhängig vom Standort

5.7 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

5.8 DYNAMIC COMPOSITION

- Parallelisierung von Abfragen durch Location Transparency
- Und Fallbacks durch zusätzliche Horizontal Scalability

KAPITEL 6:

6.1 HIERARCHICAL PROBLEM DECOMPOSITION

- Bei dieser Applikation werden Probleme in kleinere aufgeteilt.
- Nicht immer ideal da die resultierende Anzahl der Probleme überwältigend werden kann.

6.1.1 DEFINING THE HIERARCHY

- Aufteilung der Hierarchie
- Am wichtigsten sind die logischen Funktionen welche Implementiert werden
- Die mit der niedrigsten Priorität sind die kleinsten „Details“ welche keinen starken Einfluss auf das System haben.
- Die Beziehung zwischen einem Modul und dessen Nachfolger ist mehr als nur eine Notwenigkeit
    - Erlaubt die Bearbeitung eines spezifischen Problems ohne Gefahren
- Höherer Rang = wahrscheinlicher für spezifischen Use-Case

6.2 DEPENDENCIES VS. DESCENDANT MODULES

„Ownership“ ist ein wichtiger Punkt

- Es sollte ein Modul vorhanden sein mehreren Implementierungen *Andere Module, welche auf Funktionen des ersten Moduls zugreifen wollen, sind Abhängig/ haben Abhängigkeiten

6.3 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
    - Eigene Zuweisung der Zusammenhänge

6.4 ADVANTAGES OF SPECIFICATION AND TESTING

- Jede Aufgabe eines Moduls muss eindeutig sein
- Tests sind ein Notwendiger Bestandteil
    - Von test-driven development auf testability-driven Design wechseln
    - Dadurch wird besseres Design erzielt
    - Divide et Regna sollte Module hervorrufen, welche einfach zu testen sind

6.5 HORIZONTAL AND VERTICAL SCALABILITY

- Vorhandenen Protokolle kommunizieren via Message-Passing
- Frei wählbare Größe der Latenz, um die Suchanfrage zu optimieren
    - Wichtig, weil eine Instanz pro Nutzer ist nicht wirtschaftlich
        - Mehrere Nutzer auf einer Leitung, Größe der Latenz sollte nach Minimalprinzip gewählt werden

SELBSTSTUDIUM ZUM THEMA „SCHEMA.ORG“ & „OPEN-DATA“

SCHEMA.ORG:

Schema.org stellt ein Markup zur Kennzeichnung und Strukturierung von Inhalten auf Webseiten zur Verfügung, damit diese von Suchmaschinen leichter verarbeitet werden können. Dies geschieht unter der Zusammenarbeit der weltweit größten Suchmaschinen einschließlich Google.

Der Einsatz von Typisierungen von Schema.org zeichnet sich vor Allem durch die Erstellung von sog. „Rich Snippets“ aus, welche direkte Informationen in den Suchergebnissen ermöglichen. So können die Lyrics bei einer Suche nach Song-Lyrics direkt in den Ergebnissen/dem Ergebnis angezeigt werden, da diese Information als solche indiziert wurden.

Da ich mich zuvor schon mit Themen wie z.B. SEO-Optimierung auseinandergesetzt hatte, waren mir die Grundzüge und der Hintergrund von Schema.org bereits bekannt. Für mich stellte es aber lediglich ein Markup dar, welches es zwar auch ist, aber nicht, was sich noch dahinter verbirgt und welche Möglichkeiten zur Umsetzung es gibt.

Eine Kritikpunkt ist, dass die Umsetzung von Schema.org-Standards aktuell mit sehr viel Arbeit verbunden ist, da das Heraussuchen und die Implementierung jeweiliger Markups sehr viel Zeit kostet und Add-Ons/Plug-Ins diese Aufgabe noch nicht 100% sicher übernehmen können. Aus diesem Grund muss man abwägen, ob sich die Verwendung lohnt und wenn ja, in welchem Maße.

OPEN-DATA:

Offene Daten sind offiziell wie folgt definiert:
„Offene Daten sind Daten, die von jedermann frei benutzt, weiterverwendet und geteilt werden können – die einzige Einschränkung betrifft die Verpflichtung zur Nennung des Urhebers“

Diese Aussage sagt eigentlich bereits alles aus, aber es lassen sich im Detail folgende Regeln definieren:
Verfügbarkeit und freier Zugang
Die Daten müssen als Ganzes und zu nicht mehr als zumutbaren Vervielfältigungsunkosten verfügbar sein, idealerweise als Download im Internet. Die Daten müssen weiterhin in einem zweckmäßigen und editierbaren Format vorliegen.

Wiederverwendung und Weitergabe
Die Daten müssen unter Bedingungen zur Verfügung gestellt werden, die eine Wiederverwendung und Weitergabe inklusive einer Verwendung der Daten zusammen mit Datensätzen aus anderen Quellen ermöglichen.

Universelle Beteiligung
Jeder muss in der Lage sein, die Daten zu nutzen, zu verarbeiten und weiter zu verteilen - es darf keine Benachteiligung von einzelnen Personen, Gruppen, oder Anwendungszwecken geben. Zum Beispiel sind Einschränkungen in der kommerziellen Nutzung oder Beschränkungen auf bestimmte Nutzungszwecke (z.B. nur für Bildungseinrichtungen) nicht erlaubt.

Für mich bedeutet dies also, dass Open-Data die Möglichkeit bietet, verschiedene Datensätze zusammen zu nutzen und miteinander zu verknüpfen, ohne dass dies durch Lizenzvorgaben oder Formatprobleme erschwert oder sogar verhindert werden kann, da alle Datensätze transparent zur Verfügung stehen.

WORKSHOP AM 09.05.2019 (MEILENSTEIN 2)

Die Vorarbeit zum Workshop am 09.05.2019 beinhaltete die Umsetzung von Prototyp v2. Da hierfür die Implementierung der Discord-API zusätzlich zur Telegram-API geplant war, musste ich mir zunächst das nötige Wissen dafür aneignen. Dieses Wissen war allerdings bereits nach kurzen Internetrecherchen vorhanden und konnte erfolgreich umgesetzt werden. Die Arbeit an den Prototypen war generell kein Problem. Ich hätte im Nachhinein vielleicht einiges anders programmiert und gelöst, aber im Endeffekt war es eben nur ein Prototyp. Die Erkenntnisse darüber bleiben mir natürlich für die Zukunft.

Bei der Behandlung des Themas „Open-Data“ im Plenum wurden noch ein paar offene Fragen geklärt, wobei die meisten Antworten für mich irgendwie selbstverständlich waren. Darunter zum Beispiel die Antwort auf die Frage, ob offene Daten von einer höheren Instanz kontrolliert werden sollten oder nicht. Offensichtlich ist das aber nicht der Sinn von offenen Daten, da deren Quelle ähnlich wie bei einer Informationsrecherche immer individuell bewertet werden muss.

FAZIT ZUR PROJEKTARBEIT

Die Projektarbeit im Ganzen bot für mich eine gute Möglichkeit, meine Stärken und Schwächen im Bezug auf Teamarbeiten zu ermitteln. So konnte ich zum Beispiel feststellen, dass ich im Hinblick auf Projektplanung und Zielsetzung weniger aktiv werde, aber bei der Umsetzung eine sehr gewissenhafte und umso aktivere Rolle einnehme. Die praktische Implementierung des Prototyps war beispielsweise gar kein Problem, anders als das Erarbeiten von Zielen und Modellen selbst in Absprache mit Kollegen, da es dabei immer wieder zu Schwierigkeiten kam, welche aber nicht zwangsläufig auf Wissenslücken zurückzuführen waren.

Der Workload des Projekts war meines Erachtens nach sehr hoch (Im Hinblick auf das gesamte Modul), wobei dies eventuell daran liegen könnte, dass man nicht immer genau wusste, was genau von einem erwartet wurde und welche Anforderungen genau gestellt wurden

Ich verlasse diese Projektarbeit mit einem guten Verständnis über asynchrone, serverseitige Kommunikation und Framework Standards, sowie einem grundlegenden Verständnis von offenen Daten im Web. Mein Javascript-Skillset hat sich durch die Arbeit nicht grundlegend erweitert, da ich teilweise „schlampiger“ gearbeitet habe, als ich es im Normalfall getan hätte. Da wir dem Prototyp neben Telegram noch einen weiteren realistischen Use-Case beifügen sollten, konnte ich mich zum ersten Mal mit der Discord-API beschäftigen, was ich sowieso schon länger geplant hatte.