LERNPORTFOLIO

Emre Inci

Matrikelnummer 11118040

INHALT

WORKSHOP 04.04.19

WISSENSSTAND:

Nach kurzer Recherche wurde sichtlich das der Wissenstand in Javascript nicht ausreichend ist und die Kommunikationsprotokolle ebenfalls durch Selbststudium erlernt werden müssen.

Einige der Grundlagen der Softwaremodellierung sind aus dem Fach Softwaretechnik 1 bereits vorhanden, doch gab es Schwierigkeiten beim erstellen des Domänenmodells.

REQUIREMENTS:

Im ersten Arbeitsschritt haben wir die notwendigen Requirements erarbeitet:

Als Lösungsvorschlag wählten wir ein asynchrones Peer-to-Peer Netzwerk. Bestehend aus Nutzern und Services

RECAP:

Die Erarbeitung eines Domänenmodells ist welches akzeptabel ist stellt die Basis der Zielsetzung da.

Zudem wurde in geplant Tutorials in Javascript zu bearbeiten und um Lücken aufzufüllen und gewisse Teile neu zu erlernen. Es sollte außerdem das Buch Reactive Design Patterns gelesen werden bis Kaptiel 3.

- https://www.w3schools.com/Js/js_intro.asp
- https://www.w3schools.com/nodejs/nodejs_intro.asp
- GDW müssen aufgearbeitet werden (da Vorlesung nicht besucht wurde)

SELBSTSTUDIUM MIT DEM BUCH „REACTIVE DESIGN PATTERNS“ (KAPITEL 1&2)

Das Selbststudium wurde in Gruppenarbeit betrieben um Aufkommende Fragen zu beantworten und um ein ausreichendes Verständnis generieren.

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:

- Replicate = das Kopieren eines Systems/ einer Anwendung um eine Ausreichende Verfügbarkeit zu gewährleisten
- Active-Passive Replication:
    - Festlegen welches Replikat Updates akzeptiert
    - Updates werden weiter geleitet an die anderen Replikas
    - Falls das aktive ausfällt kann ein passives Replika eintreten
- 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/
    - 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 entstehen
    - 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 Schritt: If no database connection available -> Return null
- Einführung einer Explicit Queue
    - Anstatt direkt einen Error zu senden ist es möglich noch Anfragen zwischen zu schalten
    - Erst wenn diese Queue voll ist wird abgewiesen

EXPLOITING PARALLELISM:

- Die meisten Programme erwarten, eine Antwort der Funktionen, wird das nicht gewährleistet 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 komplett voneinander getrennt sind, kann ein Systemfehler das ganze Programm zum Absturz bringen

USING CIRCUIT BREAKER

- Wenn die Antwort einer Anfrage 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 11.04.19

PLAN UND FORTSCHRITT:

Aufgrund von Krankheit war es mir nicht möglich an diesem Tag den Workshop zu besuchen, doch die Installation und das seperate Aufsetzten eines lokalen RabbitMQ Servers erfolgte in den Tagen danach.

Festlegung des Kommunikationsprotokolls: RabbitMQ

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

Das Ziel des Workshop am 25.04.2019 sollte ein „Prototyp" werden, der bereits eine einfache und realistische Kommunikationskette darstellt.

WORKSHOP 25.04.19 (MEILENSTEIN 1)

AUSGANG:

Die Wetter Api des Prototypen wurde erweitert.

PLAN UND FORTSCHRITT:

- Anlegen eines Lernportfolios

Da bis jetzt noch kein Lernportfolie angelegt wurde, führte dies zu Schwierigkeiten hinsichtlich des Fokuses. Somit wurde der Fokus letzten Endes auf das Erlernte Wissen hinsichtlich der Aufgabenstellung „Reactive Design" gelegt.

- Das erlernen der "offene Daten" durch den Theorieteil des Workshops
  - Sind Daten welche für jeglichen Zweck verwendet werden können.

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

MESSAGE PASSING

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

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.

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.

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

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

- 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 ein Mal 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

EVENTS AS MESSAGES

- Eine Nachricht die versendet wird und ankommt kann mal als Event sehen.
- Da sie weitergeleitet werden können, spricht man auch von einem Event-driven 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.

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.

LOCATION TRANSPERANCY

- Warum sich darauf beschränken nur Threads zu trennen?
    - Systeme auch trennen
    - Dies eröffnet neue Blickwinkel für Performance der Systeme

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.

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
    - Oder 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

EXPLICIT MESSAGE PASSING TO THE RESCUE:

- Location Transparency's Ziel ist es den Lokalen und Globalen Nachrichtenaustausch zu vereinheitlichen und zu abstrahieren
- Mit location transparency ist jede Nachricht potentiell 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 teilservices

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 empfangszeit sparen würde

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

HORIZONTAL SCALABILITY:

- Mehr Server einzurichten die das selbe tun kann die Performance erhöhen
- Da Location Transparency ◊ Unabhängig vom Standort

LOCATION TRANSPARENCY MAKES TESTING SIMPLER:

- Da man mehrere Server hat die das selbe tun kann man sich einen rauspicken und tests durchführen ohne das system vom netz zu nehmen

DYNAMIC COMPOSITION:

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

DIVIDE AND CONQUER:

HIERARCHICAL PROBLEM DECOMPOSITION

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

.1 DEFINING THE HIERARCHY

Aufteilung der Hierarchy

- 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

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

BUILDING YOUR OWN BIG COMPANY

- Metapher
    - 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

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

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

SCHEMA.ORG: EVOLUTION OF STRUCTURED DATA ON THE WEB & OPEN DATA

STANDARDS

- MCF (Meta Content Framework) führte Ideen von „Wissens“ Präsentationen ein -> beschriftete Graphen.
- Zwischen 1997 und 2004 wurden verschiedene Standards für die Syntax und Datenmodelle entwickelt (RDF, RDFS, and OWL)

SCHEMA.ORG

- Wurde 2011 von Yahoo, Google, Bing und später Yandex erstellt.
    - -> Ziel: Schema zu erstellen welches möglichst Breit gefächert ist.
    - 297 Klassen und 178 Beziehungen in Hierarchischer Anordnung
- Schema.org Markup wird heutzutage unteranderem für Emails verwendet.
    - Abrufen der strukturierten Daten ist möglich -> zb.: Notifikation am Handy

DESIGN DESICIONS

Hauptgrund für das Design ist die Erleichterung der Veröffentlichung von Daten.

- Syntax
    - Mehrere Syntaxen werden angeboten
        - Microdata wurde entwickelt um mit der Komplexität von RDFa klarzukommen.
        - JSON-LD -> Verwendung : Javascript Seiten und Emails
- Polymorphism
    - Viele Systeme haben eine einzige Dömane und Beziehung für die Beziehungen.
        - Probleme mit der Klassen Hierarchy
        - Lösung -> Mehrere Dömanen wurden implementiert um geteilte „supertype’s“ zu vermeiden
- Incremental Complexity
    - Muss die Balance zwischen dem hinzufügen von Schemata welche sich überlappen und die Koordination des Ganzen

NUTZEN VON SCHEMA.ORG

Da sich die größten Suchmaschinen der Welt hinter diesem einheitlichen Standard stehen, können Webmaster sichergehen, dass das Markup mit itemprop und itemtype auch von diesen Suchmaschinen korrekt ausgelesen werden kann. Das verwendete Markup bietet Webseitenbetreibern die Chance, dass Inhalte von Suchmaschinen besser indexiert und für Rich Snippets oder den Knowledge Graph verwendet werden können.

OPEN DATA

Wenn es um Open Data geht stellt sich erstmal die Frage was ist Open Date. Aufgrund des Wortes Selber würde man von „offenen Daten“ ausgehen doch kann man sich darunter nur wenig vorstellen. Um Open Date vollends verstehen zu können muss man sich erst die Definition anschauen.

offene Daten sind Daten, die von jedermann frei benutzt, weiterverwendet und geteilt werden können - die einzige Einschränkung betrifft die Verpflichtung zur Nennung des Urherbers

Hier kristallisieren sich 3 Punkte heraus: Der Freie Zugang, Die Weitergabe und Wiederverwendung und zum Schluss die Beteiligung

Der Kernpunkt ist, dass der Fokus bei der Offenlegung von Daten auf nicht-personenbezogenen Daten liegt , also die keine Informationen über einzelne Personen enthalten.

Hierzu wurde 2017 vom Bundestag das Open-Data-Gesetzt beschlossen, nach dem Daten von Bundesbehörden maschinenlesbar und entgeltfrei öffentlich zugänglich gemacht werden sollen. Allerdings wird kritisiert, dass es keinen Rechtsanspruch auf die Daten gibt und es weitreichende Ausnahmetatbestände für die Veröffentlichung gibt. Nicht nur Deutschland verwendet Offene Daten.

Das Offene Datenportal der EU wurde Ende 2011 in Betrieb genommen und enthält hochwertige Daten aller Art aus allen Eu-Politikbereichen. Das wurde zu der G8- Charta festgestellt. Neben dem Zugriff auf Datensätze bietet das Portal einfachen Zugang zu einer Reihe von Visualisierungsanwendungen, die auf EU-Daten aufbauen.

Abschließend kann ich zu Open Data sagen das es etwas gutes ist, Daten frei zugänglich zu machen, doch die Kontrolle dieser von der Seite des Staates kann Problematisch sein wenn gewisse Fälle von Zensur anfallen. Gutes Besipiel ist hierfür die Türkei.

FAZIT

THEORETISCHER TEIL

Der Theoretische Teil des Modules Frameworks Daten und Dienste im Web hat viele Informationen vermittelt. Die Literatur welche uns zur Verfügung gestellt wurde war sehr Informationsreich und hat die Themen gut bearbeitet um ein Tieferes Verständnis zu ermöglichen.

Das einzig Negative wäre die Menge in der uns die Literaturaufgaben präsentiert wurden.

PRAKTISCHER TEIL

Der Praktische Teil hat sehr viel Spaß gemacht und hat mir geholfen meine stärken und schwächen im Team zu erkennen und an diesen zu arbeiten. Die Zusammenarbeit in der Gruppe lief gut, und durch die Unterstützung meiner Kameraden war es mir möglich neue Skills anzueignen und vorhandene zu raffinieren.

Die Projektarbeit lief meist reibungslos ab doch war die Vermittlung der Aufgabenstellung meiner Meinung nach Lückenhaft und hat viele Fragen offen gelassen welche zum Teil immer noch vorhanden sind. Dementsprechend sollten sie eventuell bei dem nächste Mal drauf achten ein wenig mehr auf die Studenten einzugehen.