StackOverflow: Social Network Analysis

1 Inhalt

2 Einleitung

StackOverflow ist Teil des StackExchange Netzwerks und bietet eine Plattform für Fragestellungen im Bereich Softwarentwicklung. Im Rahmen des Moduls «Social Network Analysis» der FHNW Technik führen wir eine Analyse von Fragen, Antworten und Tags von StackOverflow aus.

Wir haben eine Anwendung zur automatischen Abfrage, Aufbereitung und Speicherung dieser Daten entwickelt. Die Software bietet Exportfunktionen an, die es erlauben, die gesammelten Daten in geeigneter Form in SNA-Tools wie Gephi weiter zu analysieren.

Die zentralen Fragestellungen, auf die wir in dieser Arbeit eingehen, sind folgende:

- Was sind die populärsten Technologien?
- Wie verhält sich Popularität einer Technologie auf dessen Antwortrate?
- Sind Bounties wirklich nützlich?
- Wie sieht das Kotlin-Ökosystem genau aus?
- Frontend Face-off: Angular vs. React vs. Vue
- Welche (Frontend) Technologien werden zu Arbeits- und welche zu Hobby-Zwecken verwendet?

3 Datenbeschaffung

Die benötigten Daten sind alle frei auf StackOverflow.com verfügbar. Wir können den StackExchange-weiten REST Endpoint verwenden, um die Daten im json Format zu scrapen – wir sind hier bezüglich der Datenmenge zwar auf 10’000 API Requests (300, wenn nicht registriert) pro Tag eingeschränkt, dies ist aber mehr als ausreichend für dieses Projekt. Pro Request bekommen wir bis zu 100 Datensätze.

3.1 API Einschränkungen

Eine Einschränkung ist, dass wir die Throttle Violation beachten. Wir dürfen nicht mehr als 300 (10'000 wenn registriert) tägliche und 30 sekündliche API Requests absenden. Zusätzlich muss der «backoff» Parameter respektiert werden, welcher bei einer Response mitgeliefert werden kann. All dies wurde bei der Implementierung unseres Scrapers beachtet und stellt keinerlei Probleme dar. Wir erhalten pro Request bis zu 100 Datensätze, können also täglich bis zu einer Million Datensätze pro IP scrapen. Wir aggregieren die Datensätze zusätzlich in einer MongoDB und werden so definitiv genug Daten zur Verfügung haben.

Zusätzliche Infos hier: https://api.stackexchange.com/docs/throttle

3.2 Softwarearchitektur & Datenspeicherung

Die verwendeten Technologien beschränken sich auf Kotlin und Spring Boot. Mit einem eigenen Websraper fragen wir die Daten im json Format vom REST Endpoint ab, mappen sie zu Mongo Documents und speichern sie in einer MongoDB zwischen. Von dort werden die Daten dann transformiert und zu einem Graph in CSV Format umgewandelt, der dann in Gephi (oder auch weiteren Tools) importiert werden kann.

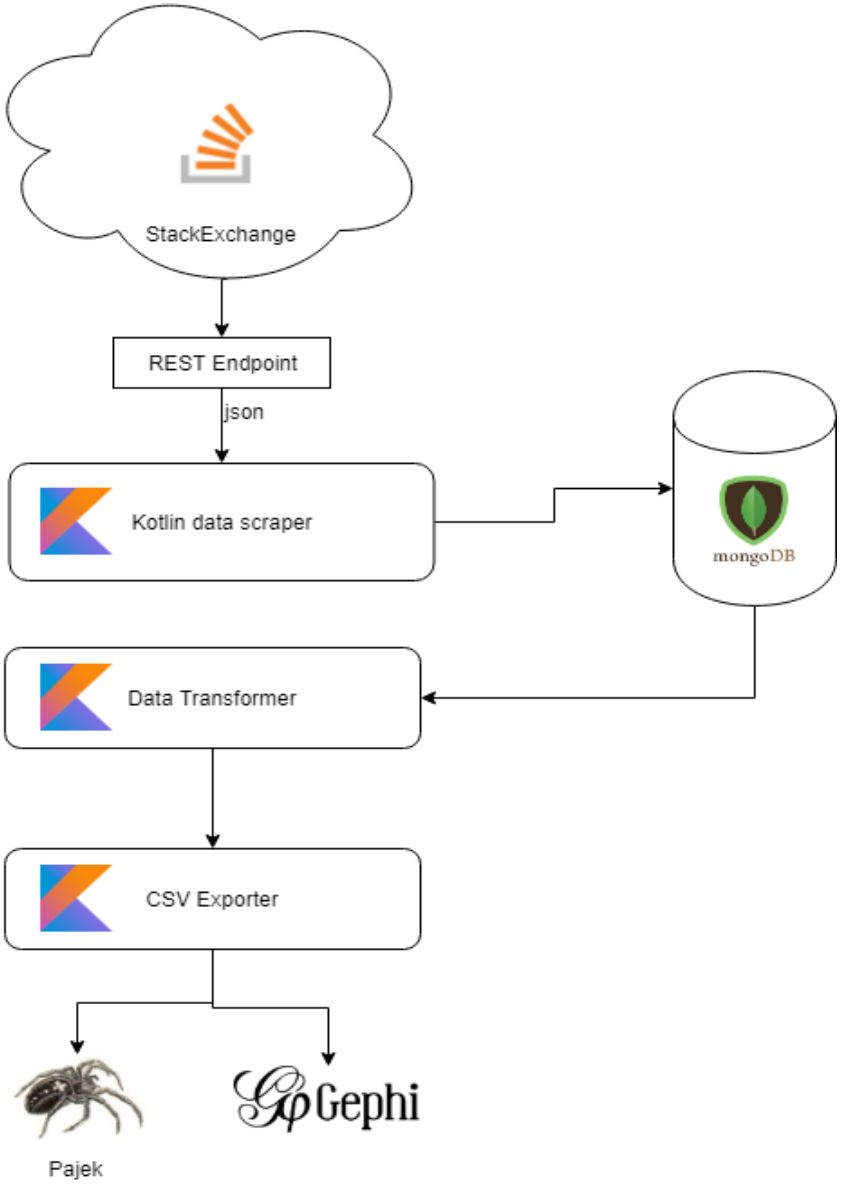

Abbildung 1: Software Architektur

3.3 How to & Setup

Unsere Software ist auf Github verfügbar: https://github.com/patricsteiner/sna_soscraper

Am einfachsten klont man das Projekt und importiert es in IntelliJ. Von dort wird gradle automatisch alle nötigen Schritte vornehmen, um die Software zum Laufen zu bringen. Im Readme findet man zusätzliche Informationen: https://github.com/patricsteiner/sna_soscraper/blob/master/README.md

Wir haben während mehreren Tagen Daten gesammelt und diese in einer MongoDB (gratis hosting auf mlab.com) gespeichert. In der Version, die wir zu diesem Dokument mitliefern, sind die nötigen Credentials bereits konfiguriert, die Applikation kann also as-is gestartet und verwendet werden. Man

kann natürlich auch eine eigene MongoDB verwenden, um Daten zu speichern. Dazu muss einfach die Datei appliaction.properties entsprechen angepasst werden.

Wenn man nur eine kleinere Menge an Daten abfragen und gleich weiterverwenden möchte ist es einfacher, die Daten einfach temporär im Memory zu halten, anstatt zu persistieren. Wir bieten dazu entsprechende Repository Klassen an (InMemoryQuestionRepository und InMemoryTagRepository). Um diese zu verwenden, muss einfach die @Repository Annotation dort gesetzt und beim MongoRepository entfernt werden.

4 Datenanalyse

Im Rahmen dieser Arbeit verwenden wir insgesamt 77587 Datensätze (Fragen), die wir während mehreren Tagen gescraped haben. Im Schnitt hat jede Frage 2.92 Tags.

4.1 Rohdaten

Wie die Rohdaten im Detail aussehen, kann der Klasse Question entnommen werden: https://github.com/patricsteiner/sna_soscraper/blob/master/src/main/kotlin/ch/fhnw/sna/soscraper/domain/Question.kt

Die für uns wichtigsten Attribute zu jeder Frage sind nachfolgend aufgelistet:

- Id
- Label
- Titel
- Erstellungsdatum
- Anzahl Antworten
- Hat akzeptierte Antwort
- Score
- Anzahl Views
- Tags

Die Daten kommen garantiert immer genau gleich an und es sind somit keine Bereinigungsschritte zwingend. Wir werden aber während der Datentransformation gewisse Attribute aggregieren um neue Features und somit zusätzliche Informationen aus den Daten herausziehen können. Mehr dazu im Abschnitt 4.3.

4.2 Analyse der Rohdaten

Eine erste Analyse der relevanten Attribute der Rohdaten liefert folgende Ergebnisse:

Attribut	**Min**	**Max**	**Average**	**Kommentar**
CreationDate	2018-10-09 06:18:44	2018-10-21 15:45:30	-	Täglich wurden also rund 6500 Fragen gestellt.
Views	2	11’233	39.87	Einige wenige Ausreisser nach oben («viral Questions»)

Answers	0	12	0.9	-
Comments	0	31	2.0	-
Favorites	0	12	0.08	Interessant wäre auch zu untersuchen, welche Fragen oft favorisiert werden.
Score	-11	77	0.13	Das einzige Attribut, das negativ sein kann.
Tags	1	5	2.92	Es muss mindestens ein und höchstens 5 Tags gesetzt werden.

Tabelle 1:Analyse der Rohdaten

In den folgenden Histogrammen sieht man zudem die Verteilung des Erstelldatums, Anzahl Antworten und der Score einer Frage.

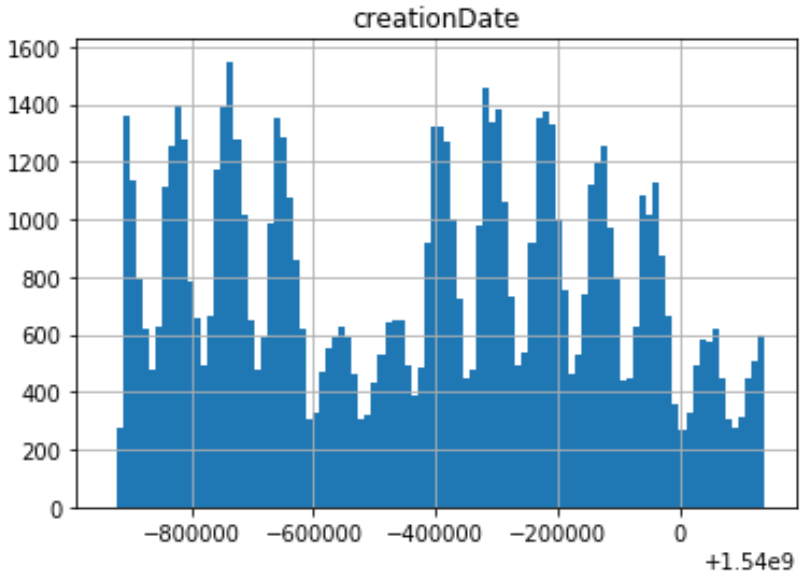

Abbildung 2: Histogramm des Erstelldatums

Die X-Achse ist ungünstig formatiert, da es sich um einen Timestamp handelt. Man sieht aber sehr deutlich, das an Wochentagen ungefähr doppelt so viele Fragen gestellt werden, wie am Wochenende.

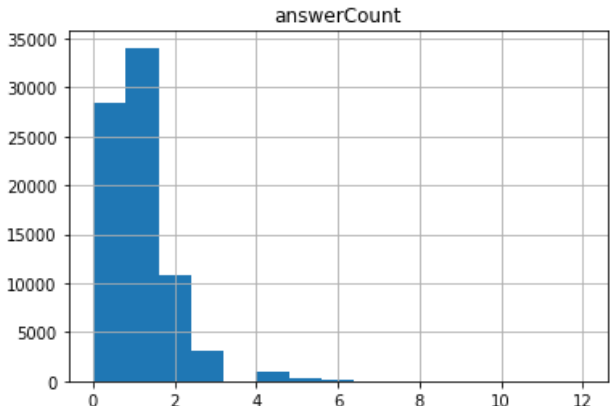

Abbildung 3: Histogramm der Anzahl Antworten

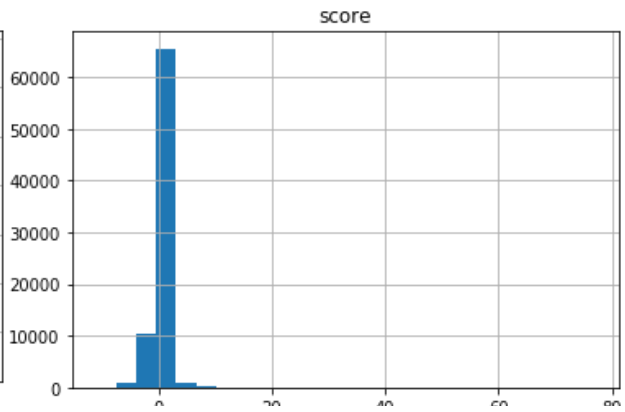

Abbildung 4: Histogramm der Score einer Frage

Diese erste Auswertung zeigt, dass wir es zwar nicht mit Normalverteilungen zu tun haben, die Daten aber durchaus realistisch sind und in dieser Form sehr wohl für unsere Zwecke verwendet werden können.

4.3 Transformation

Aus den gesammelten Daten kreieren wir zwei verschiedene Sammlungen von Entitäten. Einerseits «Question» und andererseits «Tag», wobei eine n-n Verknüpfung von Question zu Tag besteht. Es ist nötig, dass wir Tags (die einfach als String-Array einer Question geliefert werden) zu einer Entität mit eigener ID umwandeln, da das Tool Gephi eine id voraussetzt.

Wir erhalten also zwei Repositories mit folgenden Entitäten:

Abbildung 5: Entitätsmodell

Aus diesen zwei Repositories können wir nun ein 2-Mode Netzwerk generieren. Jede Question und jedes Tag werden jeweils zu einem Knoten mit dem dazugehörigen Typ. Für jedes Tag einer Question wird eine Kante zwischen den entsprechenden Knoten erstellt.

Für unsere Arbeit ist es jedoch nutzbringender und interessanter, dieses 2-Mode Netzwerk (Question zu Tag) in ein 1-Mode Netzwerk (Tag zu Tag) zu transformieren. So können wir bei der Transformation noch nachfolgend beschriebene zusätzliche Informationen gewinnen. Diese Attribute ermöglichen uns erweiterte Analysen und werden uns helfen, die Kernfragestellungen dieser Arbeit zu beantworten.

- **Occurence**: Anzahl Vorkommen in allen Fragen
- **Views**: Summe der Views aller Fragen, die dieses Tag beinhalten
- **Answered**: Summe der beantworteten Fragen, die dieses Tag beinhalten
- **Unanswered**: Summe der unbeantworteten Fragen, die dieses Tag beinhalten
- **Bounty**: Summe der Bounties aller Fragen, die dieses Tag beinhalten
- **AnsweredRatio**: Verhältnis von beantworteten zu unbeantworteten Fragen, die dieses Tag beinhalten
- **weekDay**: Anzahl Fragen gestellt unter der Woche
- **weekEnd**: Anzahl Fragen gestellt am Wochenende

Bei Ursprungsnetzwerk handelt es sich um einen gerichteten Graphen (Question zu Tag). Bei der Transformation behalten wir die Kantenrichtungen bei, operieren also weiterhin auf einem gerichteten Tag zu Tag Graphen. Man könnte die Kantenrichtungen auch ignorieren, würde dadurch aber potentiell

wertvolle Informationen verlieren. Darum tun wir das nur für gewisse Visualisierungen, bei denen die Kantenrichtungen irrelevant sind.

5 Soziale Netzwerkanalyse

Zur Analyse wurde der generierte gerichtete Graph in Gephi importiert. Alle Mehrfachkanten wurden zusammengefügt und bekamen die Summe der Anzahl zusammengeführter Kanten als Gewicht.

5.1 Kennzahlen zum Netzwerk

Die nachfolgenden Werte wurden direkt mit Gephi berechnet.

Kennzahl	**Wert**	**Kommentar**
Anzahl Knoten	16’777	Erstaunlich, dass es so viele Tags gibt!
Anzahl Kanten	239’908	Waren ursprünglich viel mehr, doppelte Kanten wurden zusammengefügt.
Average Degree (Kantenrichtung ignoriert)	3.484	Jedes Tag ist im Schnitt mit so vielen anderen Tags direkt verbunden.
Average Weighted Degree (Kantenrichtung ignoriert)	37.299	Diese Zahl sagt nicht viel aus, da die Streuung enorm gross ist (s. Abbildung 6)
Graph Density	~0	Die Dichte ist nicht genau 0, jedoch so klein, dass sie von Gephi als 0 ausgegeben wird.
Connected components	9307	Der Giant Component beinhaltet 44% aller Nodes und 99.66% aller Edges. Die restlichen Components bestehen jeweils nur aus 1 oder 2 Tags.

Tabelle 2: Kennzahlen zum Netzwerk

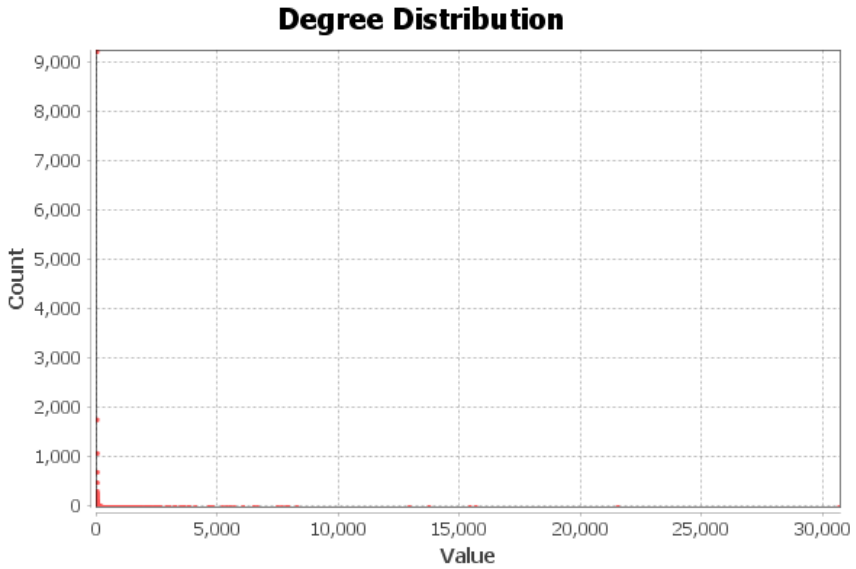

Abbildung 6: Verteilung der gewichteten Knotengrade

Weitere Metriken können in diesem riesigen Netzwerk nur mit enorm hohen Rechenaufwand berechnet werden, darum werden wir in den weiteren Analysen nur bestimmte Teilnetzwerke genauer untersuchen.

5.2 Kennzahlen zu Knotenattributen

Diese Kennzahlen werden innerhalb unserer Webanwendung berechnet. Die Berechnung dauert mehrere Sekunden, da sehr viele Daten geladen werden müssen und kann unter dem Rest Endpoint http://localhost:8080/stats abgerufen werden.

Attribut	Min	Max	Average	Kommentar
Occurence	1	8471	13.52	-
Views	3	345844	554.97	-
Answered	0	4290	5.84	-
Bounty	0	500	0.3	Wird nur bei sehr wenigen Fragen gesetzt. Siehe https://stackoverflow.com/help/bounty für weitere Infos.

Tabelle 3: Kennzahlen zu Knotenattributen

5.3 Die populärsten Tags

Die erste und interessanteste Frage ist: Was sind die beliebtesten/häufigsten Tags und deren Verbindungen? Gefiltert nach den 10 meistvorkommenden Tags erhält man folgendes Bild:

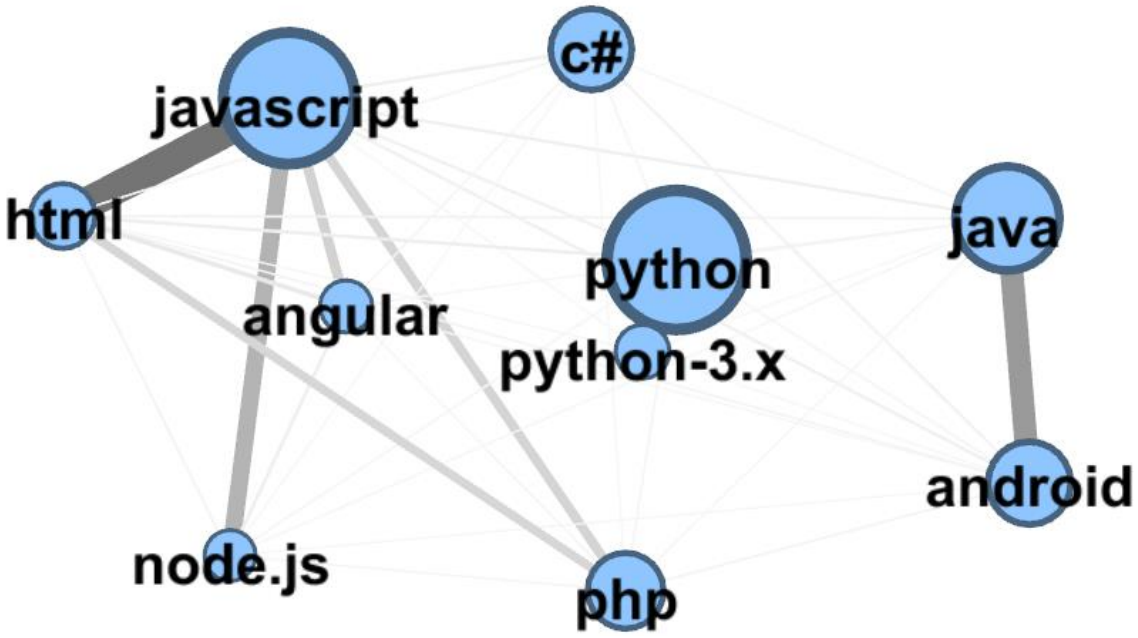

Abbildung 7: Top 10 Tags

Die Knotengrösse verhält sich hier wie auch bei allen folgenden Visualisierungen jeweils proportional zum Attribut Occurence. Wenn man dieses Resultat mit der Tag-Statistik von der gesamten StackOverflow Database (siehe https://stackoverflow.com/tags, Stand 22.12.2018) vergleicht, stellt man schnell fest, welche Technologien heutzutage wovon abgelöst wurden:

Rang	Tag	Popularität
1	JavaScript	1494422
2	Java	1494422
3	C#	1270228
4	PHP	1249791
5	Android	1159674
6	Python	1080161
7	jQuery	938323
8	HTML	793431
9	C++	598303
10	iOS	585958

Tabelle 4: All-time top 10 Tags auf StackOverflow

StackOverflow gibt es seit 10 Jahren, wobei sich die Popularität der Tags natürlich im Verlaufe dieser Zeit stark verändert hat. Wir analysieren nur die Daten der aktuellsten 2 Wochen und ziehen daraus folgende Schlüsse:

- **JavaScript** ist nach wie vor der Leader
- **Python** scheint heutzutage viel beliebter als auch schon zu sein, was wohl mit dem Data Science Hype zu tun hat
- Mit dem aufkommen von Single Page Applications hat **Angular** weitgehend **jQuery** abgelöst und auch **node.js** in den Vordergrund gerückt
- **PHP** ist nach wie vor relevant, aber definitiv am nicht mehr so sehr wie auch schon

Betrachtet man zusätzlich noch die 40 nächst-meistvorkommenden Tags, ist gut erkennbar, wie sich die heute prominenten Libraries und Frameworks um die dazugehörigen Technologien anordnen:

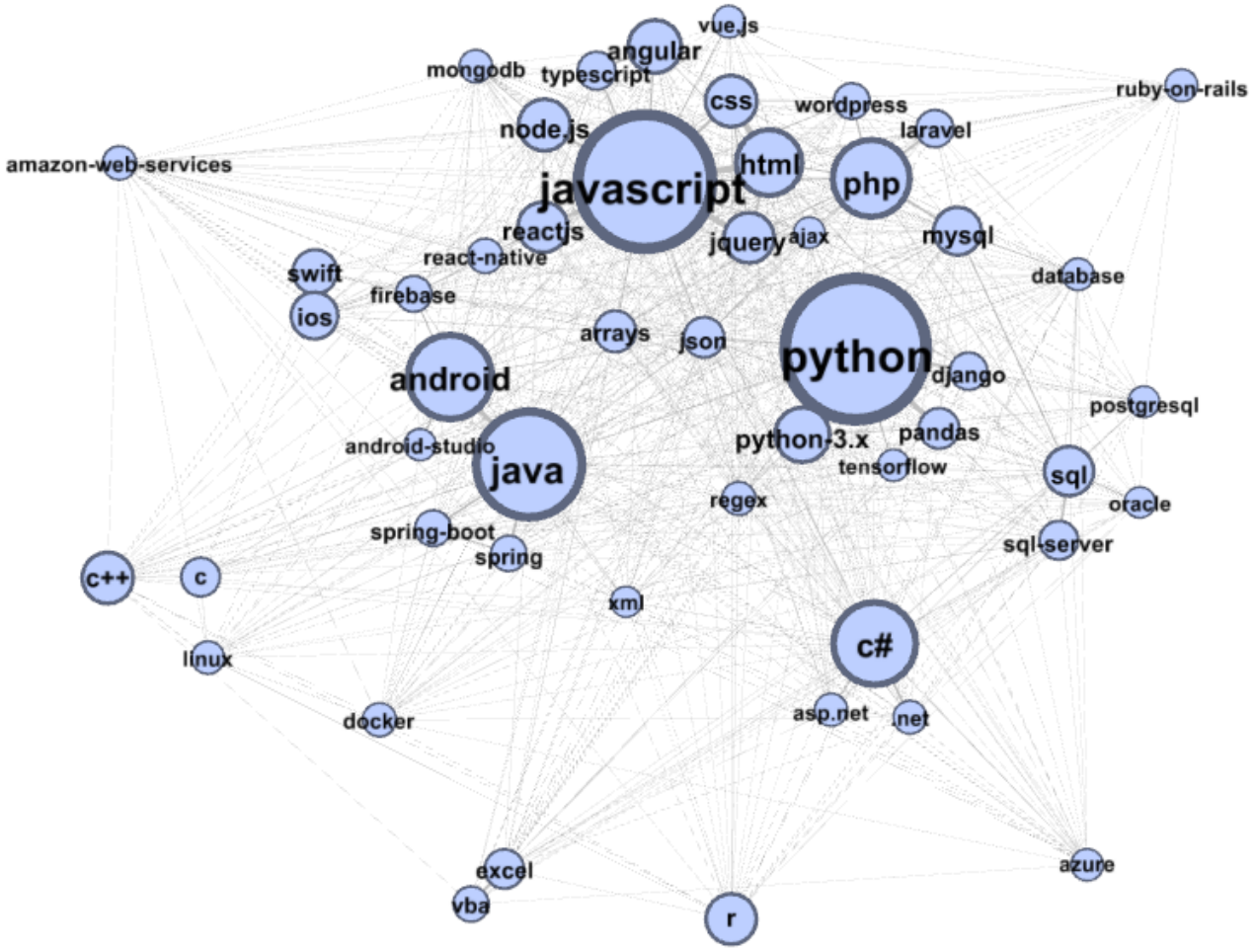

Abbildung 8: Top 50 Tags

5.4 Korrelationen zwischen Popularität und Knotenattributen

In diesem Abschnitt wird untersucht, wie die Popularität mit verschiedenen Knotenattributen korreliert. Um die Visualisierungen zu vereinfachen, verwenden wir hier die 3000 meist vorkommenden Tags. Als erstes wollen wir herausfinden, wie stark die Anzahl Views mit der Vorkommnis eines Tags korreliert. Es bestätigt sich hier die Erwartung eines sehr startken Zusammenhangs dieser beiden Attribute, wie nachfolgendem Chart entnommen werden kann. Der Pearson Korrelationskoeffizient ist 0.99.

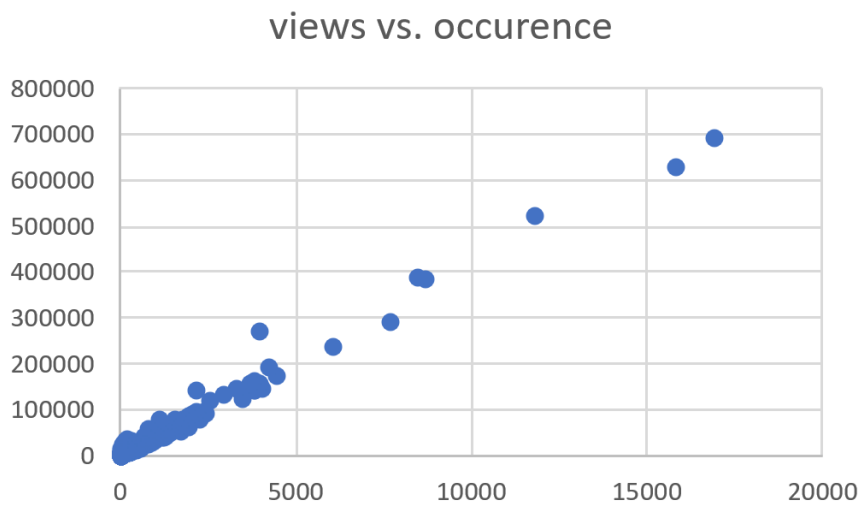

Abbildung 9: Scatterplot der Anzahl Views vs. Occurence

Viel interessanter ist aber die Frage, ob die Chance, dass eine Frage beantwortet wird, positiv korreliert mit der Popularität eines Tags. Wenn wir dazu wieder die 3000 häufigsten Tags verwenden, ist das zuerst überhaupt nicht ersichtlich:

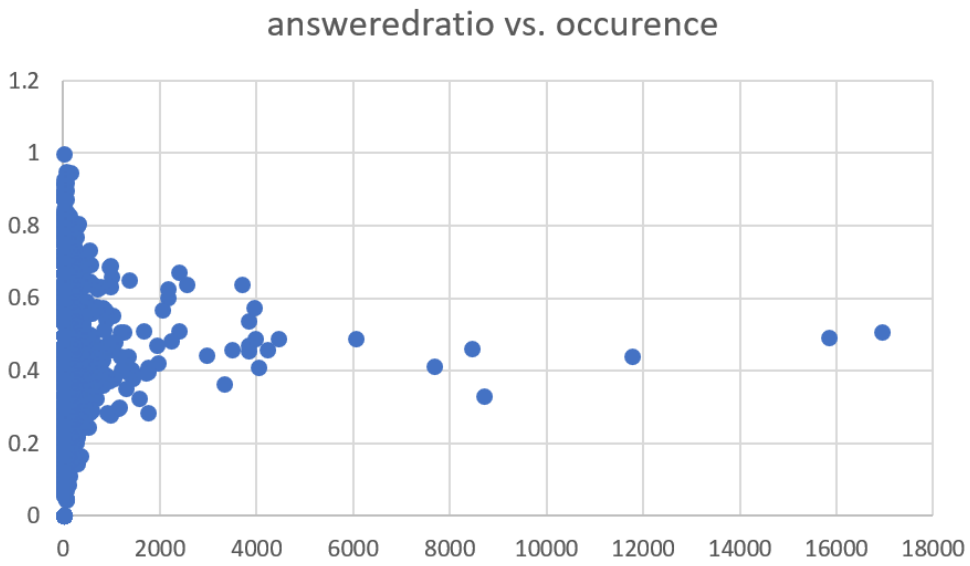

Abbildung 10: Scatterplot des AnsweredRatio vs. Occurence

Wenn die 200 höchsten Ausreisser entfernt werden, ist das Resultat immer noch nicht sehr klar. Es sieht leicht danach aus, als ob der AnsweredRatio mit erhöhter Occurence sogar sinkt. Der Person Korrelationskoeffizient von 0.05 ist sehr gering und bestätigt diese Vermutung nur teilweise. Wir erachten dies als keine relevante Korrelation, da es plausibel ist, dass mit höherer Popularität eines Tags auch mehr Unbeantwortete Fragen existieren, zum Beispiel auf Grund von Verlinkung auf Duplikate.

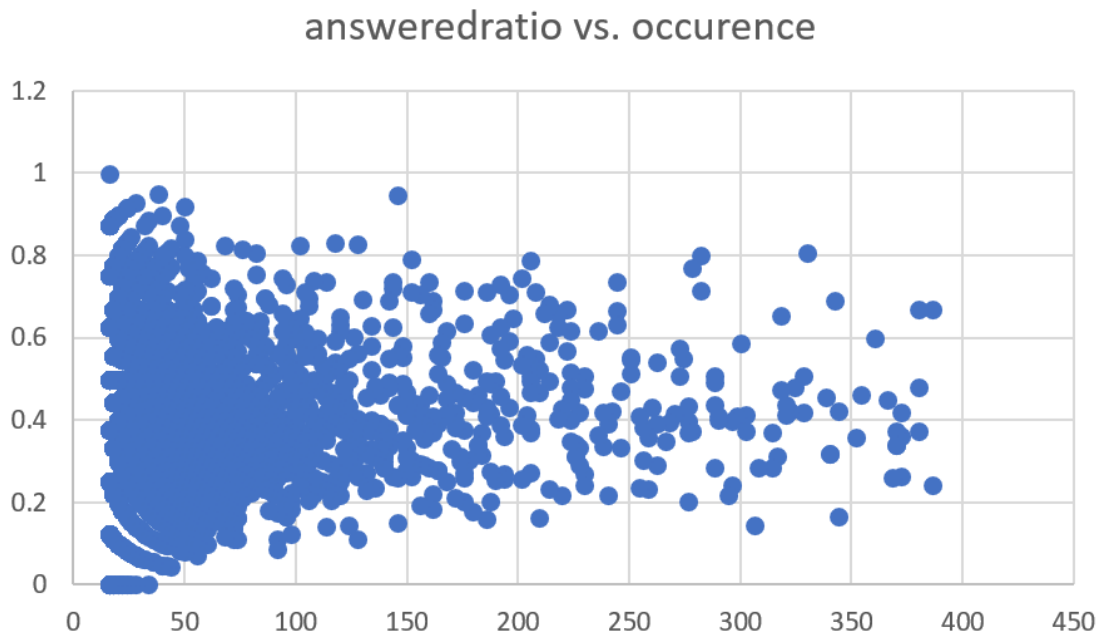

Abbildung 11: Scatterplot des AnsweredRatio vs. Occurence ohne Ausreisser nach oben

5.4.1 Bounties

Die Idee von Bounties ist, dass man einen Teil seiner StackOverflow Reputation als Prämie (Bounty) offeriert für den, der die Frage beantwortet. Reputation ist keine leicht verdiente Ressource auf StackOverflow, und eine Bounty der Grössenordnung 100 ist sehr viel mehr Wert als zum Beispiel eine akzeptierte Antwort auf eine Frage. Wir wollen herausfinden, ob sich die ausgesetzte Prämie tatsächlich für den Fragesteller lohnt und vergleichen den durchschnittlichen answeredRatio von Fragen mit Bounties mit dem von Fragen ohne Bounties.

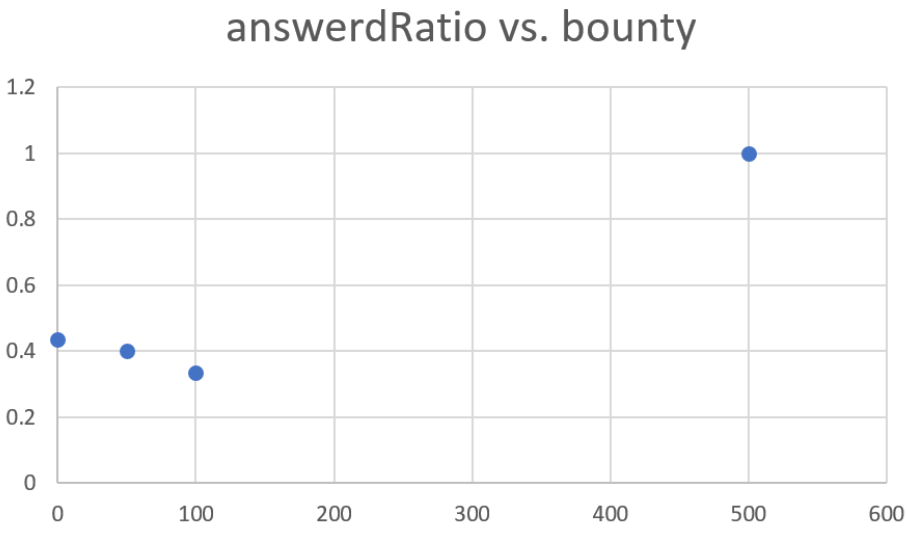

Abbildung 12: ScatterPlot des AnsweredRatio vs. Bounty

Entgegen den Erwartungen nimmt hier der AnsweredRatio mit Erhöhung des Bounties sogar ab (Ausnahme bei Bounty=500). Dies hat wohl einerseits damit zu tun, dass Fragen mit Bounty enorm schwere, wenn nicht sogar unbeantwortbare Fragen sind und andererseits dass unser Dataset nicht gross genug ist, um eine wirklich repräsentative Auswertung zu Bounties zu machen, da Bounties nur sehr selten vergeben werden.

5.5 Local View auf Kotlin

Die Software für diese Arbeit wurde in Kotlin geschrieben. Kotlin ist eine moderne JVM Sprache und wir untersuchen in diesem Abschnitt das Netzwerk rund um Kotlin. Dazu wurden alle Nodes aus dem Netzwerk entfernt, die nicht mit Kotlin verbunden sind.

Das resultierende Teilnetzwerk hat 386 Nodes und 2264 Edges. Ein Ausschnitt daraus sieht man in der folgenden Abbildung:

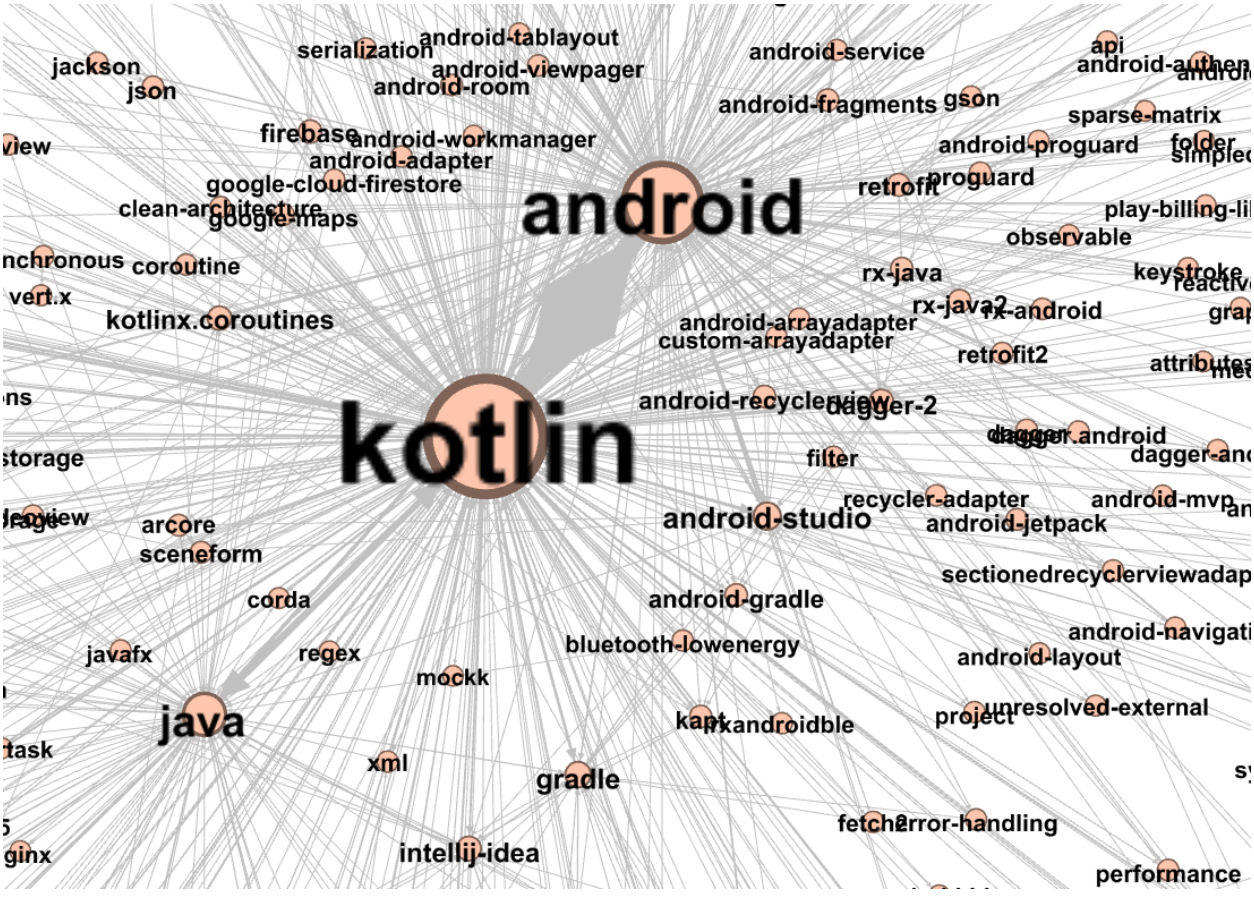

Abbildung 13: Auschnitt aus Teilnetzwerk von Kotlin

Das Netzwerk ist immer noch relativ gross und unübersichtlich. Wir fügen darum eine weitere Filterbedingung hinzu, dass nur Knoten erscheinen mit Knotengrad grosser 15. Die Knotengrösse bei der Visualisierung entspricht hier der Degree Zentralität, nicht der Occurrence wie bisher.

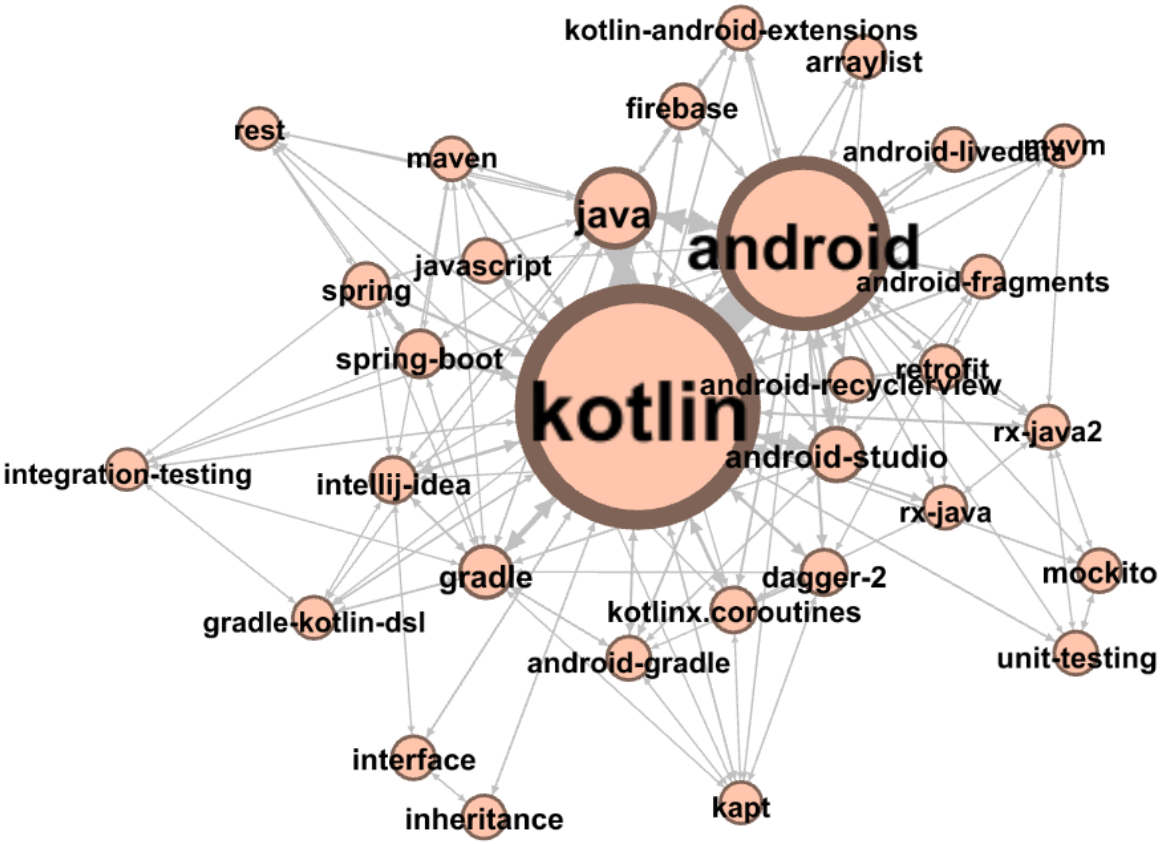

Abbildung 14: Kotlin Teilnetzwerk

5.5.1 Prestigemasse

Innerhalb dieses Teilnetzwerks berechnen wir nun normalisierte prestigemasse für alle Knoten und analysieren eine Auswahl davon:

Knoten	Indegree Prestige	Proximity Prestige	Kommentar
Android Studio	0.33	0.6	Kotlin hat mittlerweile Java abgelöst als Hauptsprache für Android-Entwicklung. Nicht verwunderlich, dass Android daher die höchst angesehene IDE für Kotlin ist.
Intellij-idea	0.3	0.58	Das zweitgrösste Ansehen geniesst IntelliJ von JetBrains, welche selber auch Kotlin entwicken. Eclipse erscheint im Netzwerk nicht mal mehr.

Spring	0.23	0.56	Kotlin kann sowohl für Frontend (JavaScript) als auch Backend (z.B. mit Spring) verwendet werden. Ersteres scheint populärer zu sein.
JavaScript	0.06	0.51	In diesem Teilnetzwerk hat JavaScript die niedrigste Prestige von allen! Verwunderlich, denn Kotlin kann zu JavaScript kompiliert werden. Wegen dem Vormarsch von TypeScript hat sich das aber trotz hervorragender Funktionalität doch noch nicht durchgesetzt.
Mockito	0.13	0.53	Wie auch in Java hat auch in Kotlin mockito nach wie vor das grösse Ansehen bei den Mock-Test Frameworks. Ktor (ein weiteres bekanntes Kotlin Test Framework) ist hier gar nicht erst ersichtlich.
Maven	0.2	0.55	-
Gradle	0.43	0.64	Gradle macht gebrauch von Kotlins modernen DSL (Domain Specific Language) Features und ist sicher zumindest teilweise deswegen für diese Sprache beliebter als Maven.

Tabelle 5: Kotlin Teilnetzwerk Prestigemasse

5.6 Analyse von Frontend Competitors (Angular, React, Vue)

Im Frontend Bereich besteht derzeit eine harte Konkurrenz zwischen den drei Frameworks/Libraries Angular, React, und neu auch Vue. In diesem Abschnitt untersuchen wir die drei Technologien genauer und Filtern das Netzwerk dazu in zwei Schritten:

1. Alle Kanten entfernen, die nicht mit einem der 3 Knoten Angular, Reactjs oder Vue.js verbunden sind
2. Alle Knoten entfernen, deren gewichteter Knotengrad kleiner als 100 ist

Wir erhalten folgendes Resultat:

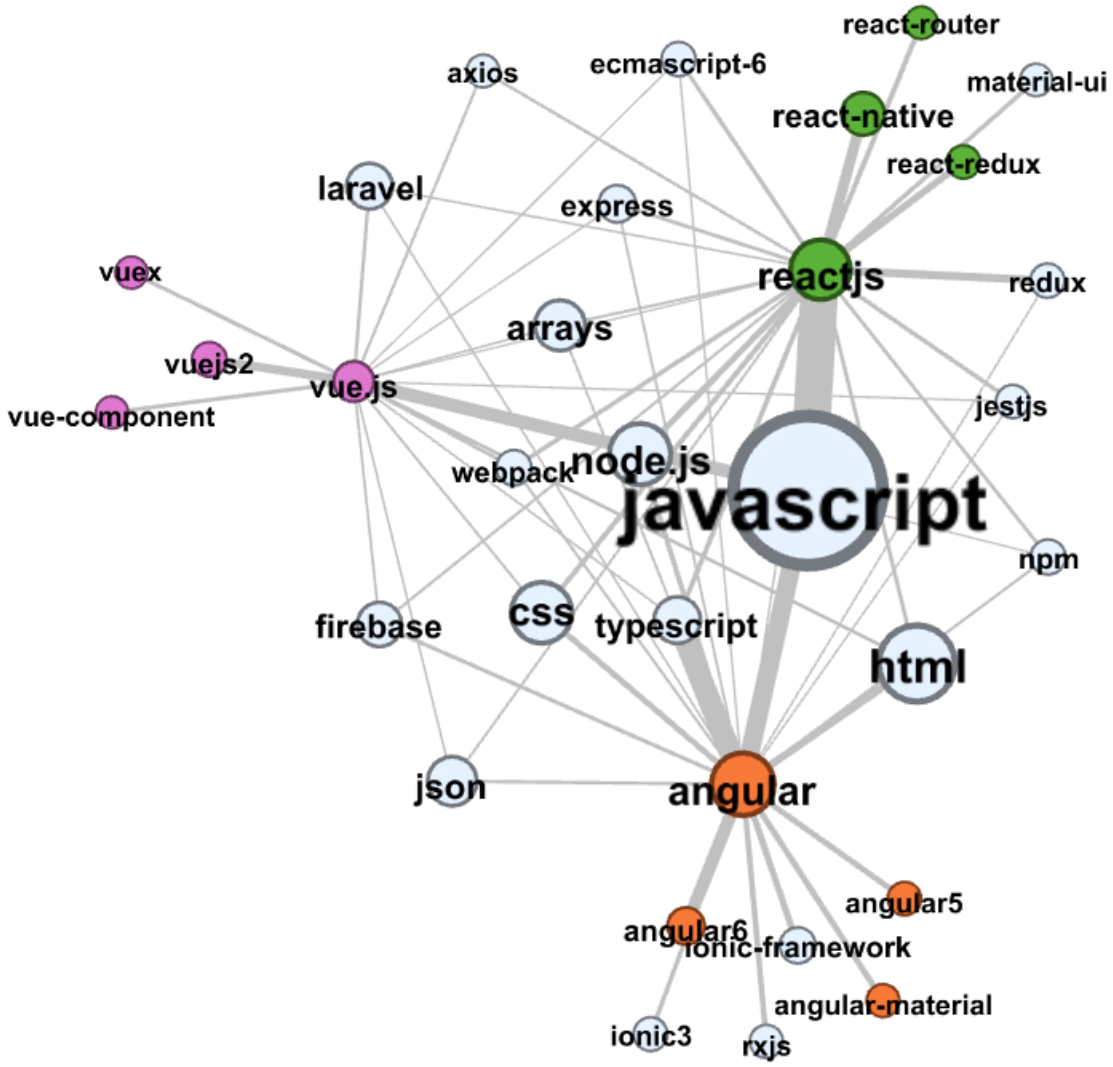

Abbildung 15: Teilnetzwerk von Frontend Technologien

JavaScript steht natürlich nach wie vor im Zentrum dieses Teilnetzwerks, da alle 3 Technologien darauf basieren. Es ist aber klar zu sehen, dass die Verbindung von Angular zu Typscript stärker ist als zu JavaScript. Das hat den Grund, das die neuen Angular Versionen auf Typescript basieren.

Weiter sind auch node.js und webpack zentral, was natürlich auf die direkte Abhängigkeit von Angular, React und Vue dazu zurckzuführen ist.

Zusätzlich ist besonders interessant, dass hier ausgerechnet Laravel als grösstes und sogar fast einziges (neben express) Backend Framework übrig bleibt – und Laravel ist ein PHP Framework. Viel eher wäre die Vermutung hier bei Spring oder Ruby on Rails gelegen. Wenn man das aber mit *Abbildung 8: Top 50 Tags* vergleicht, ist es durchaus plausibel, denn auch dort sind die beiden letzgenannten Frameworks auch nicht grösser als Laravel – erstaunlich!

5.6.1 Hobbymässig vs. beruflich verwendete Frontend Technologien

Angular wurde im Jahr 2010 erstmalig released und erfreut sich daher der grössten Popularität, gefolgt von React im Jahr 2013 und Vue in 2014. Relativ zur Popularität möchten wir hier untersuchen, wie etabliert die drei Technologien in Unternehmen sind in Vergleich zu ihrer hobbymässigen Verwendung. Diese Untersuchen basieren auf der vagen Annahme, dass Fragen zu beruflich verwendeten Technologien unter der Woche gestellt werden und die restlichen am Wochenende. Die Resultate sind daher mit einer gewissen Vorsicht zu betrachten.

Tag	**Fragen gestellt unter der Woche**	**Fragen gestellt am Wochenende**	**Verhältnis**
Angular	1190	3024	0.39
React	1094	2736	0.40
Vue	352	846	0.41

Tabelle 6: Frontend Technologien am Wochenende und unter der Woche

Die Unterschiede der Verhältnisse sind sehr klein und es ist sogar genau das umgekehrte der Fall, als angenommen wurde: Fragen zu Vue warden verhältnismässig häufiger unter der Woche gestellt als React und Angular. Und das obwohl das die neuste und somit anzunehmenderweise wenigst-etablierteste der drei Technologien ist. Wir schliessen daraus, dass Vue sehr wohl reif genug ist, um im beruflichen Umfeld eingesetzt zu werden.

Eine kurze Recherche plausibilisiert diese Schlussfolgerung:

Evan You during his tenure at Google worked in a number AngularJS projects. He realized if he could extract the best parts of AngularJS, he would build something really lightweight without all the extra concepts involves. He came up with Vue.js, a perfect frontend development framework.

Quelle: https://www.peerbits.com/blog/vuejs-is-growing-international-development-community.html

Der Vue Gründer bringt also bereits massenhaft Erfahrung von Angular ins Projekt und kannte so schon viele der Stolperfallen, konnte diese vermeiden und Vue ser schnell markttauglich machen.

6 Fazit & Ausblick

Obwohl unser Netzwerk enorm gross ist, handelt es sich doch nur um Daten von rund zwei Wochen – was wiederum nur ein Bruchteil der 10 Jahre der insgesamt verfügbaren Daten ist. Das ist aber nicht weiter tragisch, im Gegenteil: Wir haben unsere Analysen extra so gewählt, dass aktuelle Daten aussagekräfiter sind, als wenn wir auch noch historische Daten betrachten würden.

Einen Aspekt, den wir von Anfang an bewusst weggelassen haben, sind die Benutzer. Diese sind lediglich unbekannte Pseudonyme, also sicher weniger interessant, als eine Analyse von Technologien, die jedem ein Begriff sind. Es wäre aber durchaus denkbar, unsere Software so zu erweitern, dass wir auch Benutzer in das generierte Netzwerk einbeziehen und so nicht mehr Question-Tag oder Tag-Tag Netzwerke sondern User-Question und User-Tag Netzwerke bilden. So könnte man beispielsweise User suchen, die am besten auf bestimmte Job-Anforderungen treffen.