frag.extract

Beginner Practical by

Florian Rauls, Simon Gimmini, Tobias Neuschäfer at Heidelberg University

Überblick

- Aufgabe
- Projektzusammenfassung
- Ablauf
- Chrome Extension
- Extraction Model
- Ausblick
- Lehre

Aufgabe

- Entwicklung von
 - Google Chrome Erweiterung zum Erstellen von Code Fragmenten aus einem Fragen Thread und Speichern von diesen in einer Datenbank (über Frag Edit VSC Erweiterung)
 - Neuronalem Netz zum Determinieren des besten Codes zu bestimmter Intention

Projektzusammenfassung

Lines of Code:

Teil	Zeilen
Chrome Extension Skripte	537 (158 comments)
Chrome Extension HTML	125
Extraction Model	81 (71 comments)
Total inkl. ReadMe, JSON (Model)	913

- Pair Programming Sessions: eine (zur Installation des Native Messaging Hosts),
 sonst oft "soft" Pair Programming über Discord, Telegram
- >13 integration tests

Ablauf

Erste Struktur von PopUp entwickelt

Kommunikation zwischen Chrome und VSC aufgebaut

Button zum manuellen Auswählen von Fragmenten eingebaut

Design überarbeitet

Funktionalität/Logik der Fragmente in Design umgesetzt

Nutzung des Models zum Voraussagen von Fragmenten

Finales Testen der Funktionen

Chrome Extension - Design

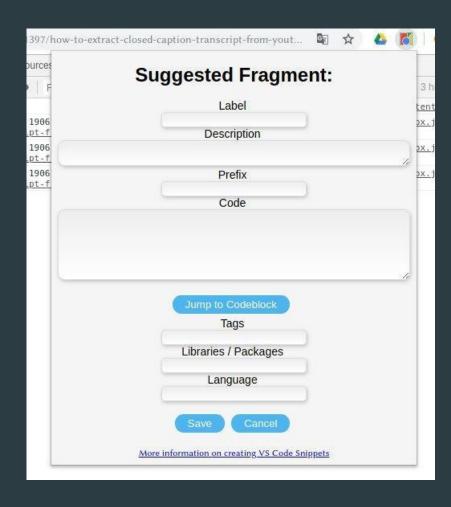
Designfragen:

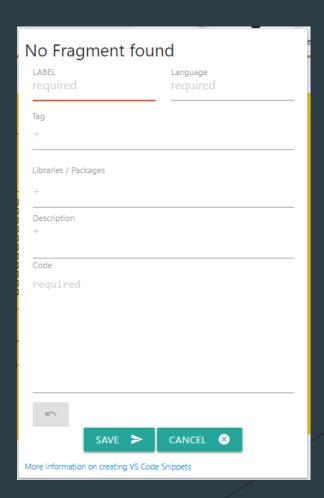
- Wie breit darf das Popup sein?
- Wie werden die Komponenten dargestellt?
 - Chips ("Tags"),
 Auswahldialoge, andere
 Schriftarten
- Wie kann man im Design Funktionale Anforderungen Umsetzen

Entscheidungen:

- Nicht breiter als der "Werberand"
- Tags als Tags
- Sprache als Auswahl
- Code in monospace
- Label und Code als verpflichtende Felder um leere und nicht gelabelte Fragmente zu verhindern
- Tags können durch einfaches "x-en" entfernt werden
- Konträre Icons statt langer Texte

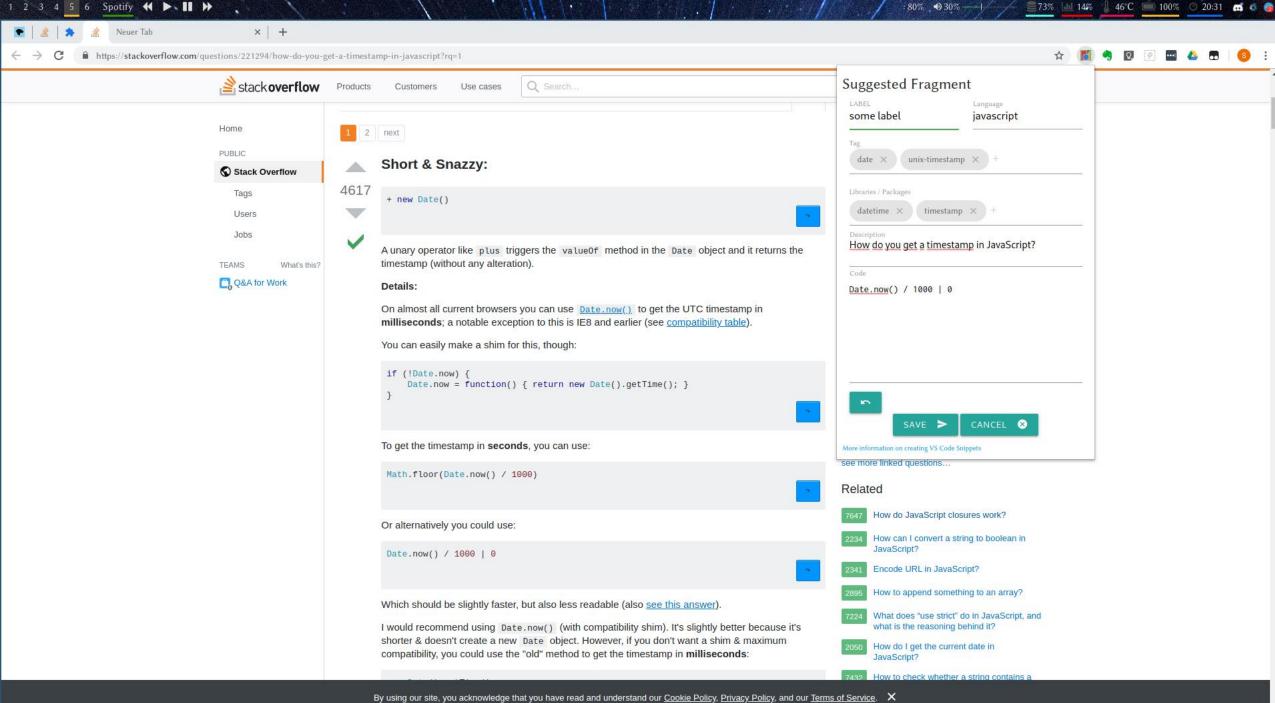
Chrome Extension - Design





Chrome Extension - Funktionalität

- Inhalt des Popups wird automatisch gespeichert und bei erneutem Öffnen geladen
- Alle Fragment Attribute werden entweder automatisch befüllt oder es werden Vorschläge gemacht
- Man kann zum ausgewählten Codeblock auf der Seite scrollen und jeden Codeblock manuell auswählen
- Native Messaging Protocol zur Übertragung der Fragmente
- NM-Host wird durch VSC-Erweiterung installiert
- Host empfängt Fragmente aus Google Chrome und verarbeitet diese für die Datenbank



Extraction Model

Grundidee:

- Datensatz (CoNaLa-Corpus) mit folgendem Inhalt:
 - "Intent" in natürlicher Sprache
 - Zugehöriges Code Fragment
- Datensatz als Trainingsgrundlage für ein Neuronales Netz
- NN unterstützt den User bei der Auswahl der Fragmente

Probleme:

- Ein in bspw. Python oder R trainiertes Model nicht nur in JavaScript, sondern gleich in einer Chrome Extension einbinden
- Training in JavaScript theoretisch möglich, aber wirft Probleme im Bereich Performanz auf
- Preprocessing muss in JavaScript reproduzierbar sein, um Daten zur Bewertung einzuspeisen
- Komplexes Problem erfordert ein komplexes Model -->
 Trotzdem muss die Auswertung am Ende performant sein

Export nach JavaScript

Theorie:

- Training des Neuronalen Netzes in Python (bspw. mit Keras)
- Abspeichern des Models in .h5-Format
- Konvertieren in TensorFlow.js-Format
- Laden des Models in der Extension
- Predictions in der Extension möglich

Export nach JavaScript

Auf TensorFlow.js übertragbar

Praxis:



https://me.me/i/if-javascript-was-a-car-hmmm-934b5ba923ae40abbad65a53f5e0f02e

Export nach Javascript

Probleme -> Lösungen:

- Laden des Models erweist sich als sehr trickreich, da dies ausschließlich über eine URL möglich ist -> Hosting per GitHub-Pages
- TensorFlow.js lange nicht so ausgereift wie Keras/TensorFlow (Layer fehlen)
 - -> Langes herumprobieren mit verschiedenen Layern und Stöbern in der Dokumentation
- Laden von Objekten erzeugt Promises, welche abgewartet werden müssen
 - -> Nutzen von async-Funktionen
- Preprocessing -> Siehe nächster Abschnitt

Preprocessing

Beispielhafter Ablauf:

- Intent: "How to loop in Python?"
- Code: "for i in range(0, 10): print("Do stuff")"



-> "how to loop in python <u>border_symbol</u> for i in range 0 10 print do stuff"



• [2, 3, 4, 6, 12, 1, 44, 77, 32, 98, 555, 0, 0, 0, 0, 0, 0]



75,32156 %iges Match

Preprocessing

Problem:

- Daten müssen in derselben Form zum Zeitpunkt der Prediction dem Neuronalen Netz vorliegen, wie sie beim Training vorlagen
 - -> Tokenization muss reproduzierbar sein

Lösung:

- Beim Erstellen der Tokens in Python wird ein Dictionary erstellt, welche Worte zu ihrem Index mappt
 - -> Abspeichern im JSON-Format
- Laden beim Start der Extension und algorithmisch den Vorgang der Tokenization nachvollziehen (inklusive Padding etc.)

Performanz

Problem:

- Teilweise über 30 Codeblöcke auf einer Stackoverflow Seite vorhanden
- Auswertung erfordert viel Zeit, wodurch die Erfahrung des Endnutzers verschlechtert wird

Lösung:

- Ausschließlich Betrachtung einer festen Anzahl an Codeblöcken (zurzeit die obersten 5), welche in Betracht gezogen werden
- Dadurch entsteht einheitliche Nutzererfahrung, egal welche Seite er öffnet
- Zudem besteht die Annahme, dass wenn sich die gesuchte Antwort nicht bereits in den 5 Antworten mit den meisten Votes wiederfindet, dass das eigene Problem zu speziell ist bzw nicht auf den Titel der Page passt und ohnehin manuell gesucht werden muss

Ausblick

- Parametrisierungsalgorithmus von Frag Edit funktioniert gut in Tests mit Fragmenten, die aus Stackoverflow erstellt wurden
 - -> zukünftig möglicherweise automatische Anwendung des Algorithmus auf neue Fragmente, die von der Chrome Erweiterung kommen
- Aufgrund der gegebenen Daten (geringe Qualität) ist das Model noch unzuverlässig, auch wenn es immer wieder gute Ergebnisse liefert
 - -> qualitativ hochwertigere Daten würden die Zuverlässigkeit steigern
- Infrastruktur bereits gegeben
 - -> es könnte bspw. ein Crawler eingesetzt werden, der mehr Daten generiert
- Workarounds im Bereich TensorFlow.js könnten Zugriff auf ein hochwertigeres
 Netz bieten
 - -> Würde die Zuverlässigkeit ebenfalls steigern

Lehre

- Besser in neue Sprache einarbeiten anstatt nur von Beispielen ausgehend nach Vermutungen zu coden
- Probleme nicht stur von nur einer Seite aus angehen
 - Nutzung einer festen Bibliothek für Tokenization in Python ist nicht reproduzierbar in JavaScript
 - Schreiben eines einfachen, eigenen Ansatzes wäre effektiver und zeitsparender gewesen
- Nicht auf die Dokumentation von Bibliotheken verlassen
 - Materialize nicht aktuell und l

 ückenhaft
 - TensorFlow.js nicht sauber strukturiert/vollständig
 - Googles Native Messaging Beispiel nicht funktionsfähig

Referenzen

- Github Repository
- <u>Materialize Library</u>
- <u>TensorFlow.JS</u>