Deep Learning for TextImaging
Gesamtdokumentation

SoSe 2021

29. September 2021

Inhaltsverzeichnis

Teil I.
reCaptcha

1. Gruppenmitglieder

- Robin Bayval
- Vedat Yildiz

2. Aufgabenkurzbeschreibung

Im Rahmen des Deep Learnings Praktikums des TTLabs stellte sich uns die Aufgabe ein reCaptcha zu entwickeln. Es ist auf beliebigen Seiten integrierbar und schützt somit die Seite von automatisierten Zugriffen. Gleichzeitig nutzen wir die Eingaben zum Crowdsourcing, um mit Hilfe des TextAnnotators des TTLabs die Annotationen zu verbessern.

3. Beschreibung der Tasks

Erstellung einer graphischen Benutzerschnittstelle für das reCaptcha

Wir haben eine Website als Schnittstelle zur Nutzung des reCaptcha erstellt. Diese ist aufrufbar über den Link: https://vesternesse.hucompute.org/recaptcha/reCAPTCHA/
Wir haben versucht die Seite intuitiv zu gestalten, sodass es ohne weitere Erklärung einfach zu nutzen ist. Auch haben wir die Webseite so gestaltet, dass sie über den PC, Tablet oder Handy aufrufbar ist.

Verbindung mit dem Websocket des Textannotators

Durch diesen beziehen wir die Daten(Annotationen), die wir nutzen um einen Test zu gestalten, der zum jetzigen Zeitpunkt nur durch einen menschlichen Nutzer lösbar sein sollte. Falls die erste Stufe gelöst wurde, dann werden die Benutereingaben im nächsten Schritt zum Crowdsourcing genutzt und die Daten(Annotationen) werden dafür an den Textannotator geschickt.

Eine unkomplizierte Möglichkeit bieten, dass reCaptcha auf seiner Website einzbinden

Falls das reCaptcha vom Benutzer gelöst worden ist muss die Webseite, die das reCaptcha nutzt dies mitbekommen. Zur Kommunikation zwischen den Websiten nutzen wir cross window communication.

4. Benutzung

4.1. Use-Case Diagramm

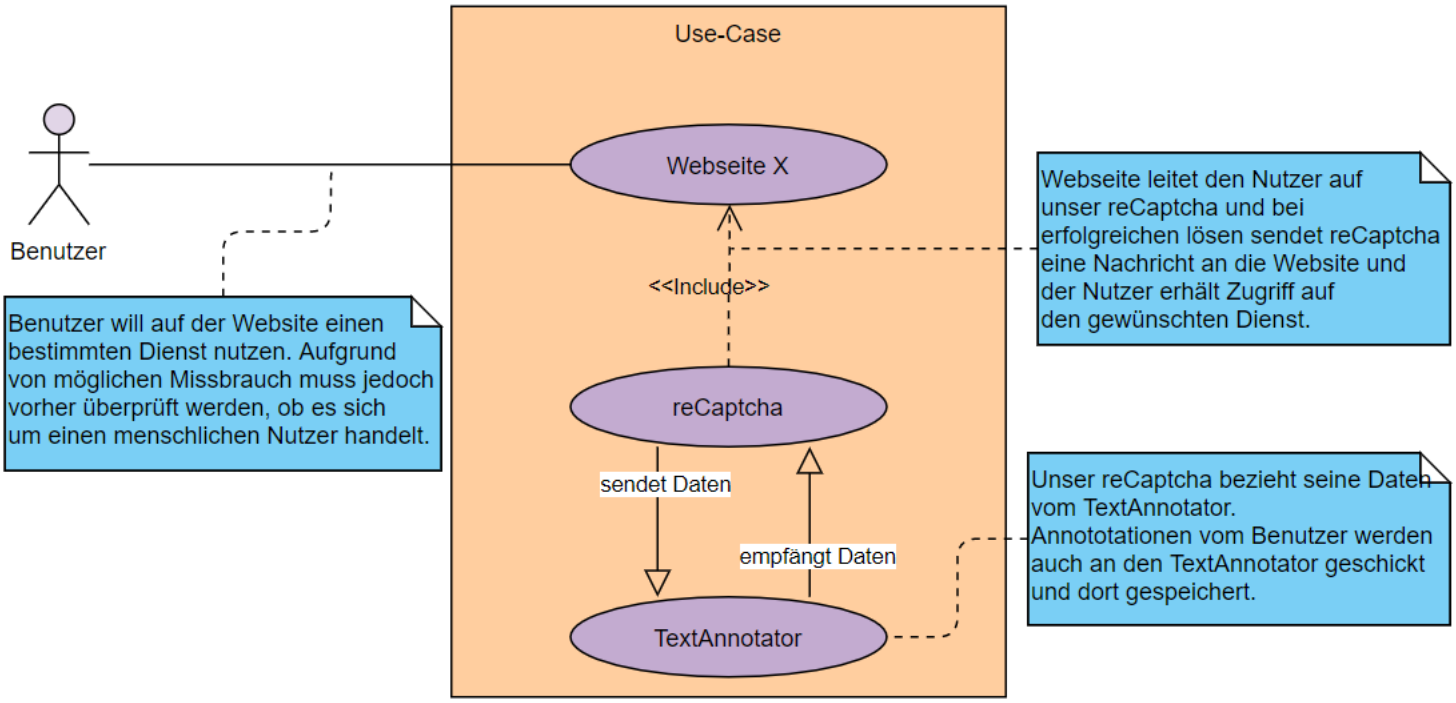


Abbildung 1: Use-Case Diagramm

4.2. Anleitung zur Nutzung

Unser Ziel war es den Betreibern von Webseiten eine simple Möglichkeit zu bieten unser reCaptcha zu nutzen. Dafür haben wir ein kleines Script erstellt den man auf seiner Webseite einfach einbinden muss. Zur Kommunikation zwischen unser reCaptcha und der Webseite die es nutzt, benutzten wir cross-window communication. Dies ermöglicht eine sichere Kommunikation zwischen window Elementen mit wenig Aufwand für den Nutzer. Zur Sicherheit sollte immer die Quelle der Nachricht berücksichtigt werden, da sonst cross site scripting Attacken auf der Webseite durchgeführt werden können.

Achtung:
Funktioniert nur bei HTML5 fähigen Browsern, da cross-window-communication angewendet wird.

Sie können beispielsweise ein Button mit der id="open" auf Ihrer Webseite erstellen,
wobei sich nach dem Klicken das reCaptcha öffnet.

```
const recaptchaURL = "https://vesternesse.hucompute.org/recaptcha/reCAPTCHA";

// Hier eine Möglichkeit wie das Recaptcha geöffnet werden kann.
const buttonID = "open"
document.getElementById(buttonID).addEventListener("click", function(){
   window.open(recaptchaURL); // Das reCaptcha muss von ihrer Seite aus geöffnet werden
}, false);
```

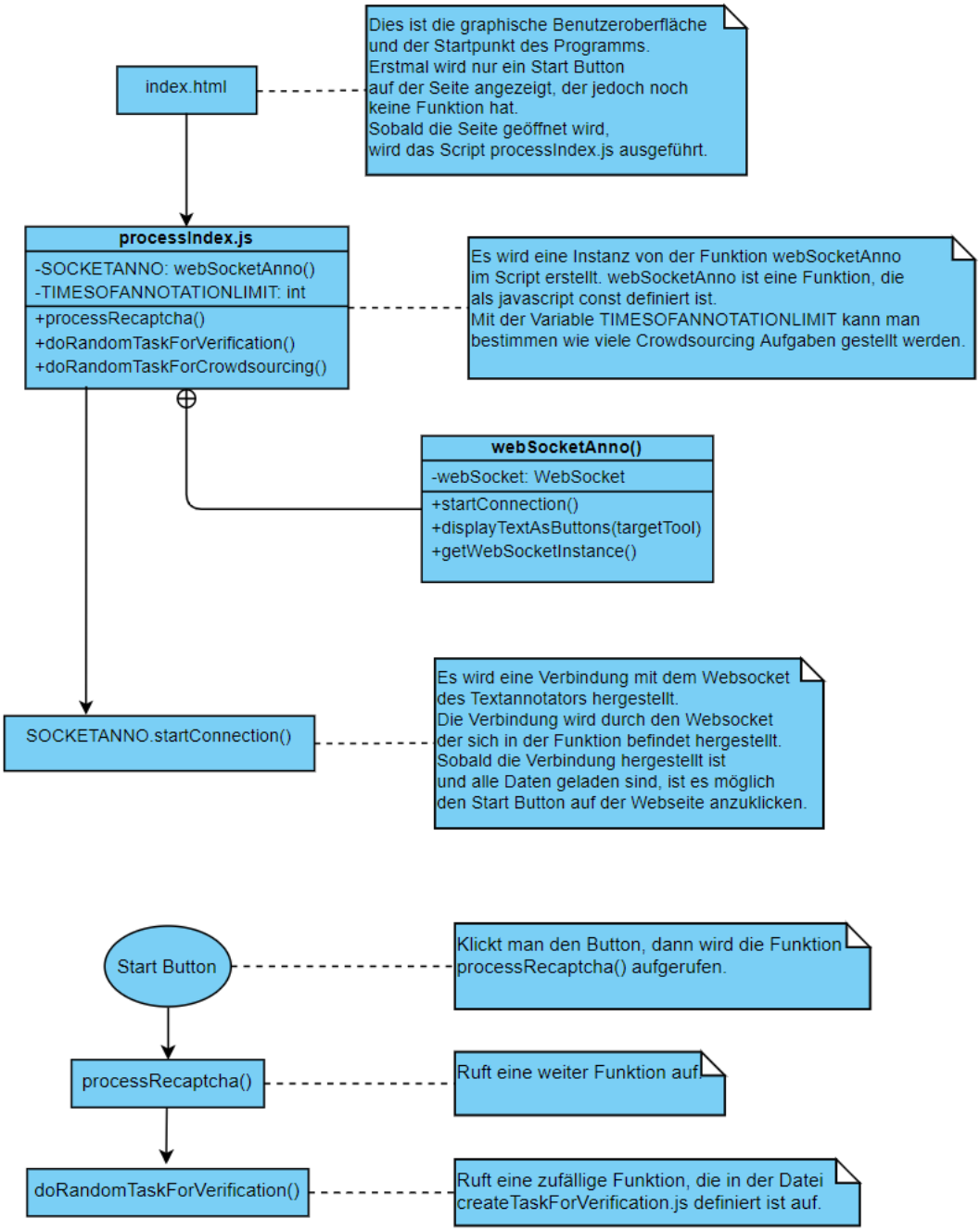
Binden sie in ihrer Seite folgendes Script ein.
Sobald das reCaptcha gelöst ist, erhält ihre Website eine Nachricht.

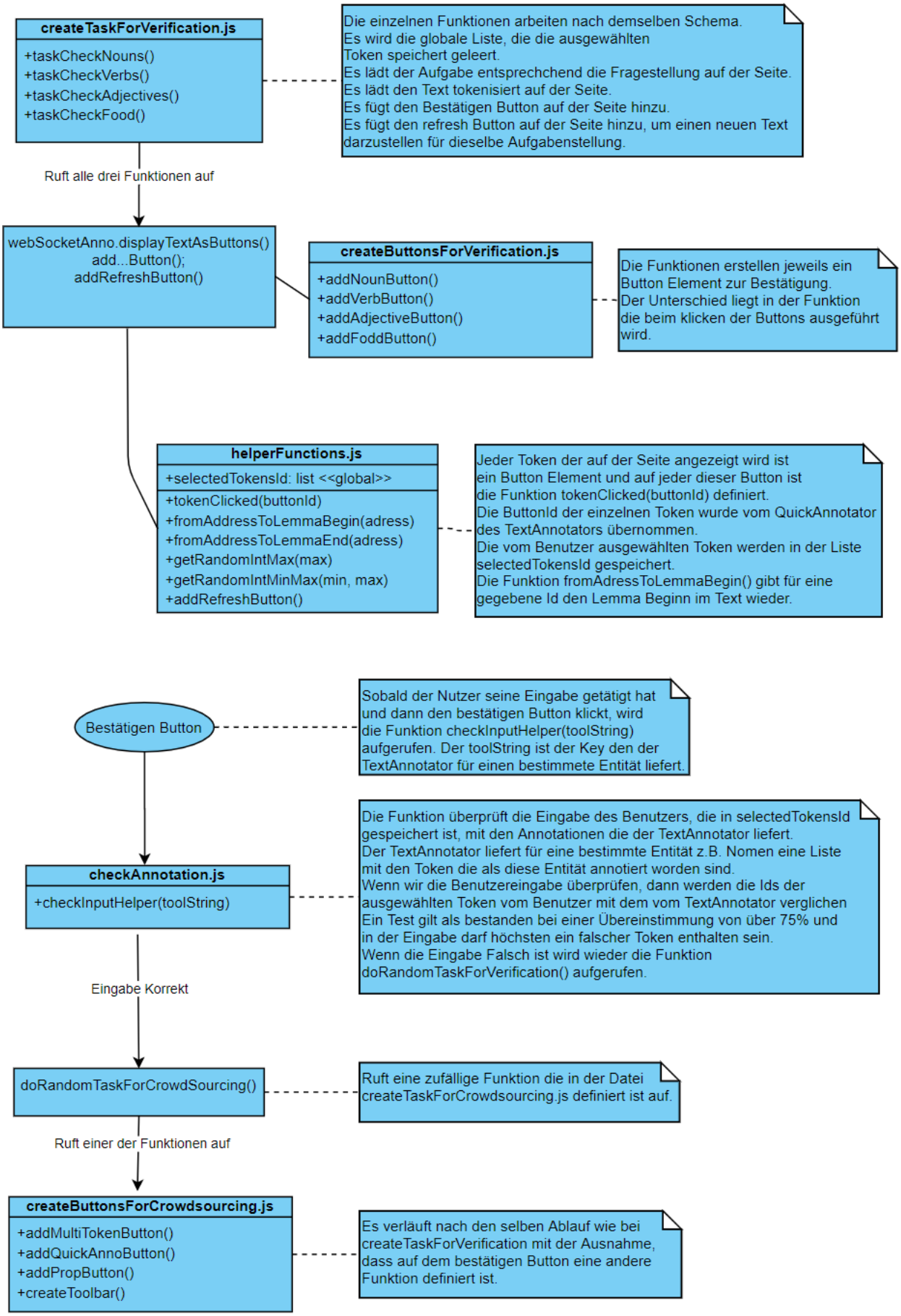
```
const recaptchaURL = "https://vesternesse.hucompute.org/recaptcha/reCAPTCHA";

// Reagiert auf erfolgreiches Captcha
window.addEventListener("message", function(e) {
    if (e.data == "solved"){
        console.log("Recaptcha erfolgreich gelöst");
        // Hier kommt Ihr Code hin
    }

}, false);
```

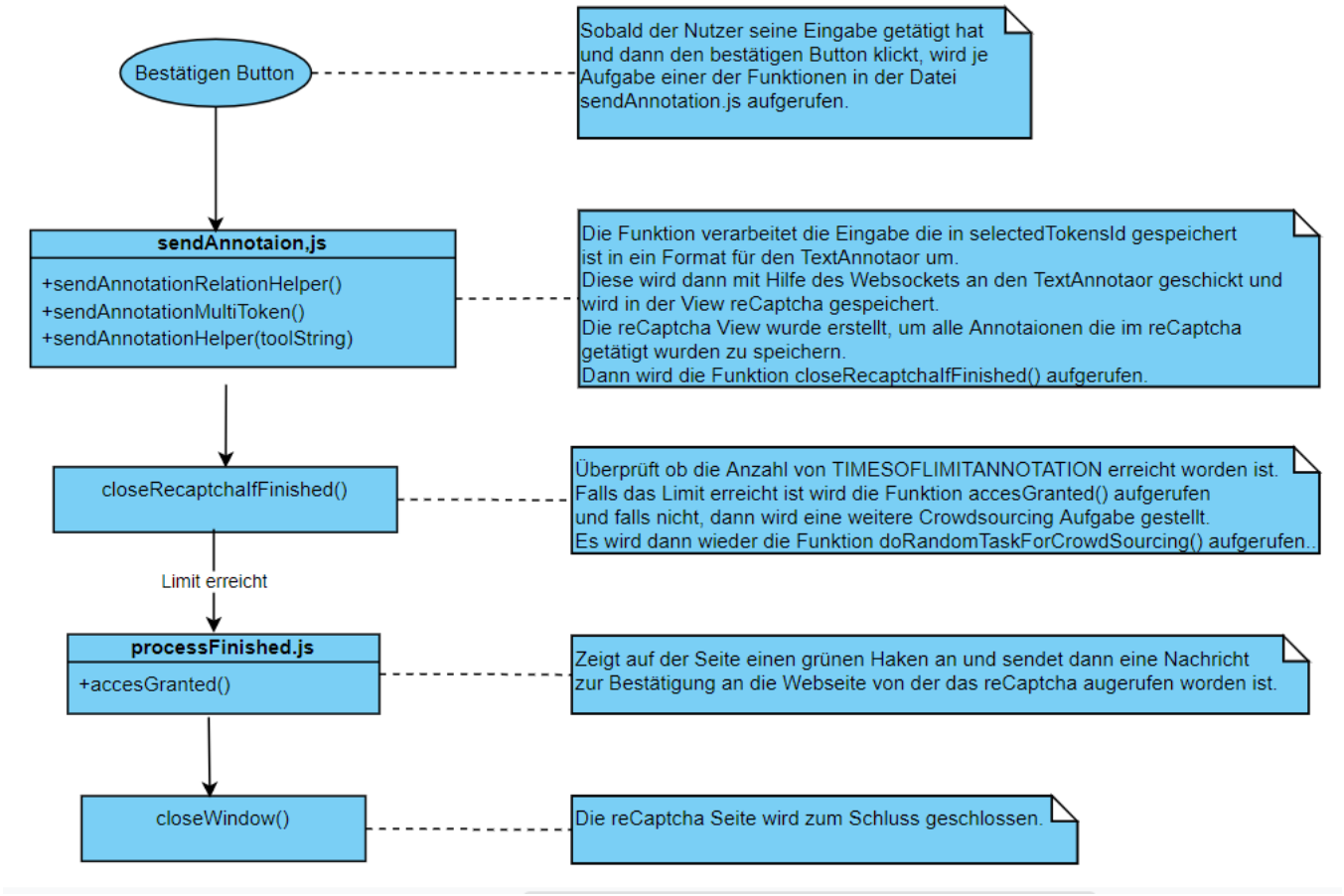
5. Technische Erklärung

Das Projekt wurde komplett mit javascript implementiert zusätzlich wurde Node.js genutzt.

5.1. Technischer Ablauf




createTaskForVerification.js
+taskCheckNouns()
+taskCheckVerbs()
+taskCheckAdjectives()
+taskCheckFood()
Die einzelnen Funktionen arbeiten nach demselben Schema.
Es wird die globale Liste, die die ausgewählten
Token speichert geleert.
Es lädt der Aufgabe entsprechchend die Fragestellung auf der Seite.
Es lädt den Text tokenisiert auf der Seite.
Es fügt den Bestätigen Button auf der Seite hinzu.
Es fügt den refresh Button auf der Seite hinzu, um einen neuen Text
darzustellen für dieselbe Aufgabenstellung.
Ruft alle drei Funktionen auf
webSocketAnno.displayTextAsButtons()
add...Button();
addRefreshButton()
createButtonsForVerification.js
+addNounButton()
+addVerbButton()
+addAdjectiveButton()
+addFoddButton()
Die Funktionen erstellen jeweils ein
Button Element zur Bestätigung.
Der Unterschied liegt in der Funktion
die beim klicken der Buttons ausgeführt
wird.
helperFunctions.js
+selectedTokensId: list <<global>>
+tokenClicked(buttonId)
+fromAddressToLemmaBegin(adress)
+fromAddressToLemmaEnd(adress)
+getRandomIntMax(max)
+getRandomIntMinMax(min, max)
+addRefreshButton()
Jeder Token der auf der Seite angezeigt wird ist
ein Button Element und auf jeder dieser Button ist
die Funktion tokenClicked(buttonId) definiert.
Die ButtonId der einzelnen Token wurde vom QuickAnnotator
des TextAnnotators übernommen.
Die vom Benutzer ausgewählten Token werden in der Liste
selectedTokensId gespeichert.
Die Funktion fromAdressToLemmaBegin() gibt für eine
gegebene Id den Lemma Beginn im Text wieder.
Bestätigen Button
Sobald der Nutzer seine Eingabe getätigt hat
und dann den bestätigen Button klickt, wird
die Funktion checkInputHelper(toolString)
aufgerufen. Der toolString ist der Key den der
TextAnnotator für einen bestimmete Entität liefert.
checkAnnotation.js
+checkInputHelper(toolString)
Die Funktion überprüft die Eingabe des Benutzers, die in selectedTokensId
gespeichert ist, mit den Annotationen die der TextAnnotator liefert.
Der TextAnnotator liefert für eine bestimmte Entität z.B. Nomen eine Liste
mit den Token die als diese Entität annotiert worden sind.
Wenn wir die Benutzereingabe überprüfen, dann werden die Ids der
ausgewählten Token vom Benutzer mit dem vom TextAnnotator verglichen
Ein Test gilt als bestanden bei einer Übereinstimmung von über 75% und
in der Eingabe darf höchsten ein falscher Token enthalten sein.
Wenn die Eingabe Falsch ist wird wieder die Funktion
doRandomTaskForVerification() aufgerufen.
Eingabe Korrekt
doRandomTaskForCrowdSourcing()
Ruft eine zufällige Funktion die in der Datei
createTaskForCrowdsourcing.js definiert ist auf.
Ruft einer der Funktionen auf
createButtonsForCrowdsourcing.js
+addMultiTokenButton()
+addQuickAnnoButton()
+addPropButton()
+createToolbar()
Es verläuft nach den selben Ablauf wie bei
createTaskForVerification mit der Ausnahme,
dass auf dem bestätigen Button eine andere
Funktion definiert ist.



6. Ergebnisse

Das Ziel unseres Projektes war es ein reCaptcha zu erstellen, dass Websiten vor automatisierten Zugriff schützt sowie mit Hilfe von Crowdsourcing die Annotation von Texten verbessert. Dies haben wir mit dem reCaptcha so umgesetzt, dass eine Vertrauensaufgabe gestellt wird, um zu verifizieren das es sich um einen menschlichen Nutzer handelt. Wenn diese erfolgreich gelöst wurde, kann man davon ausgehen, dass es sich um einen menschlichen Nutzer handelt und seine weiteren Eingaben im reCaptcha werden dann zum Crowdsourcing genutzt. Die Eingaben werden dann Textannotator gespeichert. Diese Annotaionen können dann dazu genutzt werden, um die Annotationen von Texten zu verbessern. Die Güte dieses Crowdsourcing Verfahrens können wir noch nicht bestimmen, da das reCaptcha noch nicht auf Webseiten genutzt wurde und deshalb noch keine Ergebnisse davon vorliegen.

7. Future Work

Das reCaptcha hat noch viel Raum zur Weiterentwicklung. Es bietet sich immer die Möglichkeit weitere Aufgaben für das reCaptcha zu erstellen.

Diese Aufgaben könnten das Crowdsourcing verbessern. Eine weitere Idee könnte sein, inspiriert von Googles noCaptcha, das Nutzerverhalten zu analysieren und so eine Entscheidung zu fällen, ob es ein menschlicher Nutzer ist. Dann müsste man den Nutzer keine Vertrauensaufgabe stellen und könnte ihn dafür mehr Aufgaben zur Crowdsourcing stellen. Dadurch würde man mehr Daten generieren können pro Nutzer. Die Ergebnisse des Crowdsourcing werden im Textannotator in eine view namens "recaptcha"gespeichert, jedoch werden diese noch nicht weiterverarbeitet. Zukunftige Arbeit könnte sein diese Daten entsprechend auszuwerten und so NLP-Verfahren zu verbessern. Eine Möglichkeit das reCaptcha noch sicherer zu gestalten könnte sein bei den Crowdsourcing Aufgaben den Inter-Annotator-Agreement zu berechnen und dies beim Test, ob es ein menschlicher Nutzer ist zu berücksichten. Zu jetzigen Zeitpunkt reicht es wenn der Nutzer nur die Testaufgaben löst um das reCaptcha zu lösen. Dies würde Webseiten, die das reCaptcha nutzen noch sicherer machen.

Literatur

Abrami, Giuseppe, Alexander Mehler, Andy Lücking, Elias Rieb und Philipp Helfrich (o.D.). „TextAnnotator: A flexible framework for semantic annotations“. In:

Brain, Daniel (o.D.). js window references. URL: https://bluepnume.medium.com/every-known-way-to-get-references-to-windows-in-javascript-223778bede2d.

cross-window-communication (o.D.). URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage.

Mehler, Alexander, Andy Lücking und Giuseppe Abrami (2014). „WikiNect: Image Schemata as a Basis of Gestural Writing for Kinetic Museum Wikis“. In: Universal Access in the Information Society, S. 1–17. DOI: 10.1007/s10209-014-0386-8.

websocket (o.D.). URL: https://javascript.info/websocket.