Universität Leipzig

Fakultät für Mathematik und Informatik
Institut für Informatik

– Bachelorarbeit –

Entwicklung einer Browser-Extension zur Anzeige von Datenschutzinformationen im Play Store und Evaluierung von Caching-Methoden

Autor
Alexander Prull
ap62puny@studserv.uni-leipzig.de
Institut für Informatik

Betreuender Hochschullehrer	**Erster betreuender wissenschaftlicher Mitarbeiter**	**Zweiter betreuender wissenschaftlicher Mitarbeiter**
Prof. Dr. Gerhard Heyer	Sascha Ludwig	Dr. Thomas Efer
heyer@informatik.uni-leipzig.de	ludwig@informatik.uni-leipzig.de	efer@informatik.uni-leipzig.de
Institut für Informatik	Institut für Informatik	Institut für Informatik

22. März 2019

Abstract

Diese Arbeit beschäftigt sich mit der Umsetzung einer Extension für den Google Chrome Browser. Inhalt dieser Erweiterung ist die Darstellung von datenschutzrelevanten Informationen auf der Internetseite des Google Play Store.

Die Aufgabe der Extension ist es, Verbraucher auf mögliche Schwachstellen einer App aufmerksam zu machen, bevor diese installiert wird. Als Informationsquelle dienen dabei ausgewertete Daten im Rahmen des Forschungsprojekts „Privacy-Guard “. Dabei wird die Internetseite um optische Indikatoren sowie abrufbare Informationen über Handlungsempfehlungen und Vor- beziehungsweise Nachteile zu den konkreten Bedenken erweitert.

Weiterhin werden Caching-Methoden evaluiert, um die Zahl an Serveranfragen zu minimieren und dadurch die Performanz der Extension zu erhöhen. Getätigte Messungen ergeben, dass die Methoden *indexedDB* und *localStorage* durch das Abspeichern von Datensätzen redundante Anfragen verhindern und die Ladezeiten der Browser-Extension damit um bis zu 67% reduzieren.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Laut der Datenschutz-Grundverordnung unterliegt jeder Dienstanbieter bei Erhebung von personenbezogenen Daten der Informationspflicht gegenüber der betroffenen Person (vgl. § 13 Absatz 1 Satz 1 DSGVO[14]). Dazu muss der Verantwortliche geeignete Maßnahmen treffen, *„um der betroffenen Person alle Informationen [...] und alle Mitteilungen [...], die sich auf die Verarbeitung beziehen, in präziser, transparenter, verständlicher und leicht zugänglicher Form [...] zu übermitteln;“*(§12 Absatz 1 Satz 1 DSGVO[13]).

Jedoch sind diese Dokumente im Regelfall sehr umfangreich und wichtige Informationen nicht auf Anhieb erkennbar. Das Ziel des Forschungsprojekts „Privacy-Guard“ ist es, den Verbrauchern wesentliche Punkte zu präsentieren und somit den Selbstdatenschutz zu vereinfachen. Dazu analysieren Juristen und Informatiker Datenschutzerklärungen von Apps und annotieren mögliche Bedenken.

Um diese Informationen bereits vor der Installation zur Verfügung zu stellen, entstand die Idee, eine Browser-Extension zu entwickeln, welche den Verbraucher bereits auf der Einkaufsseite vor möglichen Bedenken warnt.

1.2 Aufgabenstellung

Die Arbeit befasst sich mit den folgenden Aufgaben:

1. **”Programmierung einer Browser-Extension zur Anzeige von Datenschutzinformationen im PlayStore”**
2. **”Evaluierung von Caching-Methoden einer Browser Extension”**

Als Grundlage für diese Aufgaben dient eine Recherche um die Themengebiete einzugrenzen. Auf Basis dieser Zwischenergebnisse werden genaue Szenarien

formuliert und eine Vorgehensweise erläutert. Die auf diese Weise erlangten Endergebnisse werden präsentiert und abschließend diskutiert.

1.3 Gliederung

Die Arbeit ist in vier Teile untergliedert. Am Anfang steht eine umfassende Recherche, in der geklärt wird, was eine Browser-Extension ist, ob die genannte Aufgabenstellung bereits durch eine andere Erweiterung erfüllt wurde und welche Plattform am besten für die Umsetzung geeignet ist. Anschließend wird das Forschungsprojekt „PrivacyGuard “ vorgestellt und erläutert, auf welcher Grundlage die Informationen für die Extension basieren. Danach werden spezielle Richtlinien, Funktionen und die Darstellung im Browser erläutert. Der letzte Abschnitt beschäftigt sich mit der Aufstellung bestimmter Anforderungen an den Cache der Extension und stellt mögliche Methoden vor.

Auf Basis dieser Recherche erfolgt im zweiten Teil die Umsetzungen der ersten Aufgabe. Dazu wird ein Anwendungsszenario definiert und funktionale sowie nicht-funktionale Anforderungen erstellt. Anschließend überprüft der Abschnitt den Aufbau der Website. Darauf aufbauend werden Programmaufbau und das Ergebnis vorgestellt. Abschließend findet eine Diskussion über aufgetretene Probleme bei der Umsetzung statt.

Der nächste Teil der Arbeit behandelt die Evaluation der recherchierten Caching-Methoden. Einleitend werden grundlegende Fragen zum Speichern der Informationen geklärt und Kriterien für die Auswahl der geeigneten Methoden aufgestellt. Im Anschluss erläutert dieser Teil die Vorgehensweise der Messungen und deren Ergebnisse. Auch in diesem Teil findet eine Diskussion über mögliche Verbesserungen statt.

Im letzten Teil werden noch einmal alle Zwischenresultate zusammengefasst und ein Ausblick für diese Arbeit erstellt.

Kapitel 2

Vorarbeit

2.1 Recherche zu Browser-Extensions

2.1.1 Extension-Programmierung allgemein

Unter einer Extension versteht man ein Programm, welches den Browser um neue Funktionen ergänzt. Durch eigene Oberflächen oder Manipulation der Website erleichtern diese Erweiterungen das Nutzen des Browsers.

Im Gegensatz zu Plug-Ins haben Extensions Zugriff auf Browser-spezielle Funktionen und sind in der Lage über die Webseite hinaus zu agieren. Plug-Ins werden direkt in eine Webseite eingebettet und sind auf diese beschränkt. Der Oberbegriff „Add-on“ wird heutzutage hauptsächlich als Synonym für Extension verwendet.

Jeder größere Browser stellt eine Plattform zur Verfügung, auf denen Extensions angeboten und installiert werden können. In der Regel sind diese kostenlos. Wird eine Applikation nicht auf der Plattform angeboten oder dient sie zu Entwicklungszwecken, kann diese auch manuell aus externen Quellen installiert werden.

Extensions werden in HTML, JavaScript und CSS implementiert. Dabei können alle Bibliotheken verwendet werden, welche den Browserstandards für Extensions entsprechen. Kapitel 2.3.2 befasst sich genauer damit, welche Bedingungen für diese Bibliotheken in Google Chrome gelten.

Bekannte Beispiele sind Werbeblocker wie UBlock Origin und VPN-Anwendungen wie Hola.

2.1.2 Existenz vergleichbarer Extensions

Gesucht wurde nach einer Extension, die auf der Play Store-Seite dem Nutzer datenschutzrelevante Informationen zu den angebotenen Apps liefert, eine Datenschutzwertung im Play Store vergibt oder den Nutzer Apps nach Berechti-

gungen die Apps vorschlägt. Extensions werden nach ihrer Kurzbeschreibung in den Suchergebnissen überprüft und bei nicht eindeutiger Aufgabenbeschreibung die Infoseite aufgerufen. Nur deutsche und englische Ergebnisse werden berücksichtigt.

Die Recherche hat ergeben, dass unter den genannten Suchkriterien keine Chrome- oder Firefox-Extension gefunden wurde, die den Aufgabenbereich der geplanten Extension abdeckt. Einige aufgeführte Beispiele implementieren einen Teil der geplanten Funktion (Umsortierung, Tracker checken), aber keine Extension erfüllt alle gewünschten Aufgaben.

2.1.3 Vergleich führender Browser als Plattform für die Extension

Die getroffene Auswahl des Browsers als Plattform für die Entwicklung der Extension basiert hauptsächlich auf den aktuellen Marktanteilen. Google Chrome führt mit ca. 71%, gefolgt von Mozilla Firefox mit 9,5%, Microsoft Internet Explorer mit ca. 5,7%, Apple Safari mit ca. 5%, Microsoft Edge mit 4,4% und Opera mit ca. 2,4%.

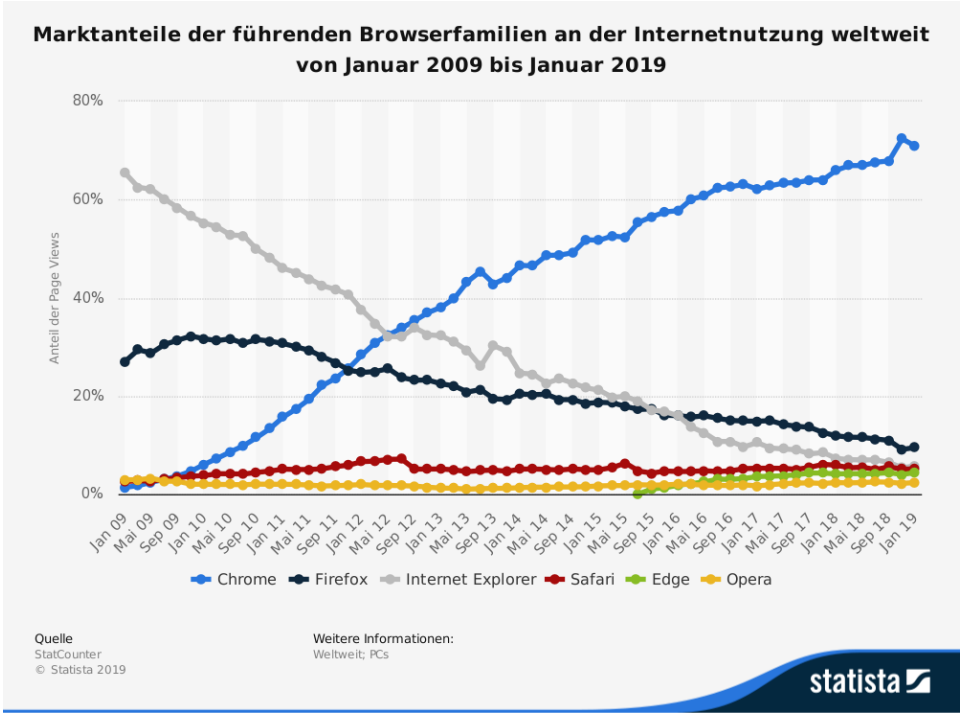

Abbildung 2.1: StatCounter. n.d. Marktanteile der führenden Browserfamilien an der Internetnutzung weltweit von Januar 2009 bis Januar 2019[31]

Aufgrund mangelnder Relevanz der Extension für Safari-Nutzer sowie der Obsoleszenz des Internet Explorers wurden diese Browser nicht weiter berücksichtigt.

Google Chrome ist den verbleibenden Alternativen Mozilla Firefox, Microsoft Edge

und Opera im Punkt Marktanteile weit voraus und somit die gewählte Plattform zur Entwicklung der Extension.

Unabhängig der Implementierung bieten sowohl Mozilla[24] als auch Edge[21] eine intuitive Lösung zur Portierung der fertigen Google Chrome-Extension.

2.2 PrivacyGuard

2.2.1 Vorstellung

Das Forschungsprojekt PrivacyGuard[11] wurde im Januar 2016 durch das Bundesministerium für Bildung und Forschung ins Leben gerufen. Das Institut für Angewandte Informatik[18], die mediaTest digital GmbH[1], die Quadriga Hochschule Berlin[28] und die Selbstregulierung informationswirtschaft e.V.[30] haben gemeinsam Möglichkeiten entwickelt, Verbraucher auf die Verarbeitung ihrer Daten durch Handy-Applikationen aufmerksam zu machen. Dabei werden Vor- und Nachteile einzelner Aspekte der Datenverarbeitung erläutert und, bei Bedarf, Gegenmaßnahmen empfohlen.

„Ziel […] ist die Erleichterung des Selbstdatenschutzes für Verbraucher auf mobilen Endgeräten." (vgl. [11])

Die Ergebnisse des Projekts teilen sich in drei Kategorien ein. Im Rahmen der Datenbeschaffung entstanden verschiedene Werkzeuge um Informationen über Apps zu extrahieren. Besonderer Wert wurde hier auf verlinkte Datenschutzerklärungen im Play Store und in der später installierten App auf dem Handy gelegt. Desweiteren wurden die Datenschutzerklärungen mittels eines entwickelten Pre-Tagging Tools, aber auch manuell annotiert, um die Verarbeitung der Texte zu optimieren.

Zur Datenverarbeitung entstand ein Backend[10], welches auf Anfrage der Bundle-ID einer App alle analysierten Daten übergibt. Dieses Backend bildet die Grundlage für die Datenvisualisierung von PGuard und ist die Schnittstelle der Informationen für die Browser-Extension.

Mit dem Projektabschluss im Juni 2018 stellte PrivacyGuard eine Webseite zur Analyse von Datenschutzerklärungen und Apps zur Verfügung[27]. Als Prototypen entstanden zusätzlich eine App zur Analyse aller auf dem Handy installierten Applikationen auf ihren Datenschutz und die in dieser Arbeit behandelte Browser-Extension zur Visualisierung der durch das Projekt gewonnenen Informationen über Apps im Play Store.

2.2.2 API-Anbindung für die Extension

Sämtliche, von der Extension visualisierte Daten werden über die Schnittstelle des Backends angefragt. Dazu benötigt die API mindestens eine Bundle-ID der App, Priorität und die Ausführlichkeit der Antwort.

Je höher die Anfrage priorisiert ist, um so eher wird sie vom Backend verarbeitet. Bei einer hohen Ausführlichkeit umfassen die angeforderten Informationen komplette Datenschutzerklärungen und alle Metainformationen zur Datenbeschaffung. Dagegen beinhaltet eine Antwort mit niedriger Ausführlichkeit lediglich Sprache, Quelle und Extraktionsdatum der Datenschutzerklärung sowie die Nummer der geltenden Infofelder mit jeweiligen Textpassagen aus der Datenschutzerklärung.

Für den Anwendungsfall der Browser-Extension besteht eine hohe Priorität um Wartezeiten möglichst gering zu halten. Dagegen reicht eine niedrige Ausführlichkeit zur Darstellung der nötigen Informationen für den Verbraucher. Zusätzlich bleibt so der Datenverkehr der Extension eher gering.

Die Infofelder sind der Hauptinformationsträger der Schnittstelle. Sie sind in 31 Eigenschaften unterteilt und können eine sogenannten „rote Linie “ sein. Das bedeutet, dass bei Besitz dieser Eigenschaft die entsprechende App potentiell gegen ein Gesetz verstößt. Folgende Infofelder sind im Rahmen des PrivacyGuard Forschungsprojekts zur Beurteilung von Apps entstanden(siehe Abbildung 2.1 und Abbildung 2.2).

2.3 Implementierung einer Google Chrome Extension

2.3.1 Eigenschaften

Die Architektur einer Google Chrome Extension stellt ein Paket aus mehreren Dateien dar und ist vergleichbar mit anderen Web-Technologien wie zum Beispiel Webseiten(vgl. [19],K. 2).

Grundvoraussetzung für eine funktionierende Extension ist die *manifest.json*, welche die nötigen Informationen für den Browser bereitstellt und festlegt mit welchen Dateien und Rechten die Extension aufgebaut ist. Hinzu kommt mindestens eine HTML-Datei zur Darstellung der Inhalte und mindestens ein Skript zur Umsetzung der Funktionalität. Erweitert werden diese oft durch CSS-Dateien.

Externe Bibliotheken wie JQuery können ebenfalls eingebunden werden, müssen aber aufgrund der Policies[4] von Google Chrome vollumfänglich lokal vorliegen.

Die Manifest-Datei ist im JSON-Format aufgebaut und beinhaltet sämtliche Informationen über die Extension. Wichtige Punkte sind Name der Extension, Beschreibung, Rechte und Aufbau. Unter Rechten oder *permissions* werden alle APIs aufgelistet, welche die Extension benötigt um ordnungsgemäß zu funktionieren. Bevor ein Nutzer später die Extension installiert, muss er diesen Rechten zustimmen.

Der Aufbau wird im Punkt *content scripts* in drei Eigenschaften unterteilt: unter welchem URL sind die Skripte aktiv, welche Skripte sind dort aktiv und welche CSS-Dateien werden dort von der Extension eingesetzt.

HTML-Dateien werden als „User-Interface Elemente“ zusammengefasst und beinhalten im Normalfall eine *popup.html* zur Darstellung des Fensters der Extension in der oberen rechten Ecke des Browser-Fensters. Je nach Funktionsumfang können weitere UI-Elemente eingebunden sein, um beispielsweise die besuchte Webseite zu erweitern.

Die vorhandenen Skripte werden normalerweise in zwei Kategorien eingeteilt. Das sogenannte „Background-Skript“ dient als Event-Handler und kommuniziert zwischen Extension und Browser. Alle restlichen Skripte sind „Content-Skripte“. Sie beinhalten die eigentliche Funktionalität der Extension.

2.3.2 Funktionsumfang und Richtlinien

Google setzt verschiedene Qualitätsansprüche an die Entwicklung einer Extension. Den Leitfaden bildet dabei das sogenannten „single purpose“-Prinzip[7]. Das heißt, jede Anwendung muss auf sich entweder auf ein bestimmtes Thema fokussieren, wie zum Beispiel Datenschutzerklärungen und darf zu diesem Thema verschiedene Funktionen anbieten. Oder die Extension konzentriert sich auf eine bestimmte Funktion des Browser, wie zum Beispiel die Startseite und darf dort Inhalte zu mehreren Themen implementieren.

Ein weiterer Aspekt, der mit dem „single purpose“-Prinzip zusammenhängt, ist die Reichweite der Extension. Google unterscheidet hier zwischen „page-action“ und „browser-action“.

„page-action“ bedeutet, dass das Icon der Extension lediglich auf einer bestimmten Seite aktiv ist und den Nutzer darauf hinweisen soll, wo sich die Erweiterung einschaltet. Icons mit „browser-action“ sind permanent aktiv und zeigen somit, dass die Extension seitenübergreifende Funktionen hat.

Auch inhaltlich gibt es bestimmte Richtlinien[4] zu beachten. Die „Content Policies “ untersagen die Einbindung von Material mit sexuell expliziten Inhalten, Gewaltdarstellungen, „Hate Speech “, Identitätsbetrug, Urheberrechtsverletzung, Schadsoftware, Glücksspiel, Spam oder anderen illegalen Aktivitäten.

Ist Werbung ein Bestandteil der Erweiterung, unterliegen diese Inhalte ebenfalls strengen Auflagen. Der Nutzer muss in der Lage sein, die Werbung ohne Probleme zu entfernen, zum Beispiel durch Deinstallation der Extension. Werbeanzeigen dürfen keine Programmfunktionen versperren, imitieren oder andere schädliche Absichten verfolgen. Sind Anzeigen von externen Webseiten geschalten, muss das klar ausgewiesen sein.

Erhebt, speichert oder verarbeitet die Extension sensitive Nutzerdaten, muss der Entwickler das entsprechend deklarieren und trägt Verantwortung für die Sicherheit der Daten. In diesem Zusammenhang benötigt die Anwendung eine eigene Datenschutzerklärung.

2.3.3 Darstellung im Browser

Die Extension wird an mehreren Stellen im Browser integriert. Dabei besitzt jedes installierte Programm ein Icon in der Adresszeile des Browsers (Abbildung 2.2). Dieses Element dient als Steuerelement für Funktionen der Erweiterung, wie zum Beispiel das Aktivieren und Deaktivieren der Extension. Ein ausgegrautes Icon bedeutet dabei, dass die Extension inaktiv ist, bzw. das die zugehörige Webseite nicht in diesem Tab geöffnet ist.

Eine weitere Möglichkeit Inhalte und Funktionen der Erweiterung zu verwalten, ohne dafür Template des Icons zu verwenden, ist die dedizierte Optionenseite[6]. Diese kommt vor allem bei umfangreicheren Browser-Extensions vor.

Alle restlichen Elemente werden direkt in die angezeigte Webseite integriert und erweitern die Anzeige beliebig nach Funktion der Extension.

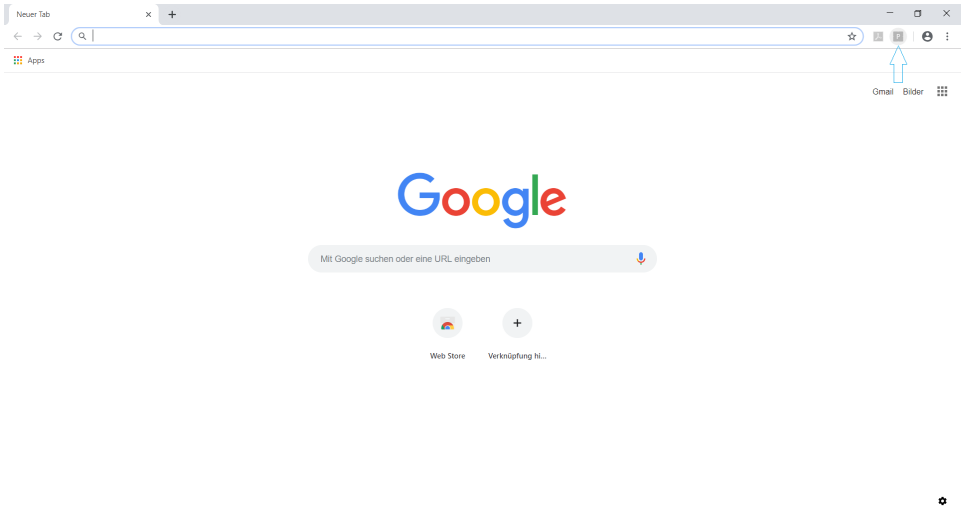

Abbildung 2.2: Browser-Extension Icon in der Adresszeile

2.4 Caching-Methoden

2.4.1 Anforderungen

Diese Arbeit beschäftigt sich damit, welche Caching-Methoden für Browser- Extensions geeignet sind. Der Fokus liegt hierbei auf den in Aufgabe 1.2 beschriebenen Anwendungsfall. Darauß ergeben sich folgende Anforderungen an den Speicher:

Performanz spielt eine große Rolle, da dem Nutzer alle gespeicherten Daten der Extension zeitgleich zum Abschluss der Ladezeit der Webseite zur Verfügung stehen sollen. Werden Daten mit Verzögerung auf der Webseite dargestellt oder

verlängert der Prozess sogar die initiale Ladezeit, hat das eine beträchtliche Auswirkung auf die Nutzererfahrung. So zeigt eine Studie von Google Research auf dem Jahr 2017[32], dass die Absprungrate sich bei Ladezeiten von über zwei Sekunden stark erhöht.

Aufgrund der Schnelllebigkeit der Informationen müssen diese nicht über einen längeren Zeitraum abgespeichert werden. Außerdem verarbeitet die Browser-Extension keine Daten weiter. Verlorene Einträge können so durch eine erneute Anfrage verlustfrei ersetzt werden.

Lastverteilung ist eine der Hauptanforderungen an die Caching-Methoden in dieser Arbeit. Da alle Daten über ein zentrales Backend abgerufen werden, besteht die Gefahr des Flaschenhalseffekts. Zudem sollen allgemein möglichst wenig Anfragen an externe Quellen gestellt werden, um so das Risiko zu vermeiden, auf Antworten von Servern zu warten.

Alle Daten, die im Cache abgelegt werden, sind durch PrivacyGuard gewonnene Informationen über die einzelnen Applikationen und beziehen sich nicht auf den Nutzer der Extension. Es werden also keine sensiblen Daten im Cache abgelegt. Somit besteht kein Bedarf an Sicherheitsmaßnahmen wie etwa die Verschlüsselung der Daten.

Die Datenvisualisierung der Extension ist überschaubar und auch die zu Grunde liegende Datenstruktur der API ist einfach aufgebaut. Dadurch kann auf komplexe Datenbankstrukturen oder zusätzliche Frameworks verzichtet werden.

2.4.2 Caching-Methoden und deren Eigenschaften

Um zu evaluieren, welche Caching-Methode am besten für den beschriebenen Anwendungsfall geeignet ist, befasst sich dieses Kapitel mit der Auflistung aller möglichen Speicherfunktionen für Browser-Extensions. Die dazu aufgelisteten Eigenschaften dienen als Vergleich für die spätere Vorauswahl der Methoden.

Alle Daten der Extension sollen im Browser gespeichert werden. Datenbanken auf externen Servern bieten keinen Mehrwert, da die Informationen vom Backend nicht noch verarbeitet, personalisiert, oder gesichert werden müssen. In diesem Fall dient das Backend als Datenbank, die durch lokale Speicherprozesse entlastet werden soll.

Cookies sind kleine Textspeicher, die normalerweise von Webseiten in Browsern genutzt werden, um Informationen über den Besucher zu speichern. Sie besitzen meist eine kurze Lebensdauer und werden über HTTP-Header übertragen. Auch Browser-Extensions können Cookies nutzen, um Informationen zu speichern(vgl. [3], K. 2; [5]). Das Buch verweist darauf, dass man bis zu 50 Cookies mit einer Gesamtgröße von 4KB pro Domäne ohne bedenken nutzen kann. Jedoch genießen Cookies, unter anderem aufgrund der EU-Richtlinie[16], keinen guten Ruf. Webseiten müssen sich in der Regel vor Gebrauch die Genehmigung des Nutzers

einholen. Dies könnte also auch bei Extension zu rechtlichen Problemen führen. Außerdem blockieren viele Nutzer diese Cookies eben aus datenschutzrechtlichen Gründen.

Die Web Storage API ist eine einheitlich Methode von mehreren Browsern, Daten lokal zu speichern(vgl. [3], K. 3; [26]). Pro Domäne verfügt der Web Storage über 5-10MB, je nach Browser. Dabei besteht jede Einheit aus einem key-value-Paar, welches Daten in Form von Strings speichern kann. Die Web Storage API ist unterteilt in „local storage“ und „session storage“. Während beim session storage alle Daten nur solange gespeichert sind, bis der Browser wieder geschlossen wird, bleiben die Daten beim local storage solange bestehen, bis sie manuell überschrieben oder gelöscht werden.

Mit der IndexedDB-API ist die Extension in der Lage eine domainspezifische Datenbank als Cache zu erstellen(vgl. [3], K. 4; [22]). Diese Datenbank besteht aus sogenannten „Object Stores“, welche Tabellen mit flexiblen Datentypen beinhalten können. Die Speicherkapazität wird nicht exakt definiert und ist abhängig von der Größe der Festplatte[23]. So liegt der verfügbare Speicher zwischen 10MB und 50% des freien Festplattenspeichers. IndexedDB wird nicht noch komplett unterstützt und gilt als vergleichsweise komplexe API.

Ergänzend zu erwähnen ist die SQLite basierte Datenbank Web SQL(vgl. [3], K. 5). Seit 2010 wird diese Spezifikation von dem World-Wide-Web-Konsortium allerdings nicht mehr zur Implementierung empfohlen und gilt seit jeher als veraltet.

Nr.	Bezeichnung	rote Linie
1	Ihre Zahlungsdaten werden unverschlüsselt übermittelt	Ja
2	Die App hat Internetzugriff	Nein
3	Die App benennt keine Kontaktmöglichkeit für datenschutzrechtliche Anliegen	Ja
4	Ihre Login-Daten werden unverschlüsselt übermittelt	Ja
5	Die App stellt keine Datenschutzerklärung auf Deutsch bereit	Nein
6	Die Datenschutzerklärung verwendet ungenaue Formulierungen	Nein
7	Die Verschlüsselung der App ist unsicher	Nein
8	Die App nutzt Standortdaten	Nein
9	Die Datenschutzerklärung kann geändert werden, ohne Sie hierüber zu informieren	Ja
10	Das über Sie erstellte Profil wird durch öffentliche Informationen über Sie ergänzt	Nein
11	Ihre Daten werden durch Dienstleister verarbeitet	Nein
12	Die App integriert Werbenetzwerke	Nein
13	Die App erhebt eine Vielzahl an Geräteinformationen	Nein
14	Die App stellt die Datenschutzerklärung erst nach Start der App bereit	Nein
15	Die App hat Zugriff auf Ihr Adressbuch	Nein

Tabelle 2.1: Übersicht der Infofelder von PrivacyGuard

Nr.	Bezeichnung	rote Linie
16	Die App enthält Malware	Ja
17	Die App übermittelt Daten an Dritte	Nein
18	Die App erhebt statische Gerätekennungen	Nein
19	Die App verarbeitet Daten, die für die Funktion der App nicht erforderlich sind	Nein
20	Die App verarbeitet Daten, die ausdrücklich ausgeschlossen wurden	Ja
21	Die App ermöglicht einer Vielzahl von Drittanbietern Zugriff auf Ihre Nutzungsdaten	Nein
22	Es wird ein Profil über Sie erstellt	Nein
23	Ihre Daten werden in der Unternehmensgruppe geteilt	Nein
24	Die App kann sich im Hintergrund unbemerkt aktualisieren	Nein
25	Die Herkunft der App ist unbekannt	Nein
26	Die App stellt keine Datenschutzerklärung bereit	Ja
27	Ihre Daten werden für personalisierte Werbung genutzt	Nein
28	Die App klärt nicht ordnungsgemäß über Datenverarbeitungen im Ausland auf	Nein
29	Die App stellt unterschiedliche Datenschutzerklärungen in der App und im App-Store bereit	Ja
30	Ihre Daten werden über die App veröffentlicht	Nein
31	Die Sprachsteuerung ist dauerhaft im Hintergrund aktiv	Nein

Tabelle 2.2: Übersicht der Infofelder von PrivacyGuard

Kapitel 3

Hauptteil

3.1 Aufgabe 1: Implementierung einer Browser-Extension zur Anzeige von Datenschutzinformationen im Play Store

3.1.1 Anwendungsszenario

Während vor einigen Jahren Applikationen hauptsächlich auf eigenen Webseiten zum Download angeboten wurden, haben sich die Appstores mittlerweile durchgesetzt. Vorteile für diese Plattformen sind unter anderem erleichterter Zugang, Vergleiche mit anderen Applikationen und individuelle Empfehlungen.

Bei der Wahl für eine bestimmte Applikation achten Nutzer auf Aspekte wie Preis, Anzahl der Downloads und Bewertungen von anderen Nutzern. Immer wichtiger wird aber auch die Frage: Welche Daten gebe ich der Applikation frei und wie werden diese verarbeitet? Der Play Store bietet zwar einen groben Überblick, welche Daten eine Applikation von dem Handy benutzt, aber nicht wie diese vom Anbieter weiterverarbeitet werden.

Außerdem sind Käufe von Apps in angelegten Benutzerkonten gespeichert. Mit diesem Benutzerkonto kann der Nutzer wiederum persönliche Daten für Login-Prozesse in Apps nutzen. So entsteht ein großes Netz an Informationen über das der Nutzer selbst keine Übersicht mehr hat.

Um Nutzern eine genaue Übersicht zum Datenschutz gewähren, betrachtet die Extension dabei Fragen,welche der Play Store nicht unmittelbar beantwortet:

1. **Handhabung der Daten**: Wie werden die Daten verarbeitet und an wen werden diese weitergeleitet? Wird ein Profil anhand der Daten erstellt? Welche Sicherheit besteht bei der Übertragung der Daten?
2. **Vor- und Nachteile der Datenverarbeitung**: Kann der Anbieter die App-

likation dadurch komfortabler gestalten? Wird Werbung in der Applikation personalisiert? Besteht Gefahr vor Missbrauch der Daten?

3. **Kontrolle über die Daten**: Welche Möglichkeiten stehen zu Verfügung im Falle von Nichteinverständnis? Ist der Umgang mit den Daten nach der Installation noch einschränkbar. Kann der Nutzer die Verwendung der Daten verbieten und trotzdem die App weiterhin nutzen?

Ziel ist es Verbrauchern diese Fragen mittels der Erweiterung des Google Play Stores durch eine Extension zu beantworten.

3.1.2 Anforderungsanalyse

3.1.2.1 Funktionale Anforderungen

Aus den Fragen, die bei dem Anwendungsszenario entstanden sind, werden funktionale Anforderungen gebildet um konkrete Aufgaben für die Extension zu schaffen(vgl. [15], S. 29-30).

/F10/ **Erweiterung der Informationen im PlayStore**: Der Nutzer hat die Möglichkeit im Browserfenster per Aktivierung bzw. Deaktivierung der Extension zusätzliche Datenschutzinformationen zu den angezeigten Applikationen ein- bzw. auszublenden.

/F20/ **Anzahl von bedenklichen Eigenschaften einer Applikation**: Zu jeder Applikation erhält der Nutzer ein Feedback von der Extension, wie viele Bedenken vorliegen.

/F30/ **Darstellung von kritischen Eigenschaften einer Applikation**: Eigenschaften einer Applikation, welche einen erheblichen Nachteil für den Nutzer darstellen oder einen möglichen Gesetzesverstoß beinhalten, werden hervorgehoben.

/F40/ **Abrufen von Details zu den Bedenken**: Wird ein Bedenken angezeigt, kann der Nutzer direkt Erläuterung, Handlungsempfehlung sowie Vor- und Nachteile zu diesem Bedenken abrufen.

/F50/ **Empfehlung bei Suchanfragen**: Basierend auf den Bedenken einer Applikation kann der Nutzer die Suchanfrage so anpassen, dass ihm unbedenkliche Applikationen priorisiert angezeigt werden.

3.1.2.2 Nichtfunktionale Anforderungen

Das Programm richtet sich in erster Linie an Nutzer, denen keine besonderen informatischen Kenntnisse abverlangt werden. Extensions zeichnen sich durch ihre Einfachheit aus. Nutzer wissen vor der Installation, welche Funktionen diese Programme haben. Die Extension soll auf den ersten Blick klar machen, welche Komponenten des Browsers erweitert oder verändert wurden(vgl. [29], S. 249-250).

/NF10/ **Darstellung und Einbindung der Informationen**: Darstellung und Einbindung spielen bei Browser-Extensions eine wichtige Rolle. Hier wird keine grundlegend neue Oberfläche gestaltet, sondern eine bereits vorhandene erweitert. Der Fokus fällt darauf, die bestehende Oberfläche so zu verändern, dass alle Elemente der Extension an der richtigen Stelle eingebaut werden. Der Nutzer soll auf den ersten Blick erkennen, welche neuen Informationen zu welchen bereits bestehenden Teilen der Website gehören.

/NF20/ **Persistenz der Website**: Im Gegenspiel zu NF10 darf die Website nicht so verändert werden, dass sie in ihrem Aussehen und ihren Funktionen zu stark von ihrem Originalzustand abweicht. Gerade bei Seiten, auf denen viele Elemente automatisch generiert werden, verursachen kleine optische Veränderungen schon Probleme beim Aufbau der Website. Entsprechend müssen Informationen so subtil wie möglich eingebettet werden. So wird verhindert, dass der Nutzer die Extension nur aufgrund der Optik wieder deinstalliert.

/NF30/ **Handhabung**: In der Extension werden viele und vor allem auch umfangreiche Informationen angeboten. Diese dürfen den Nutzer nicht überwältigen. Dennoch müssen sämtliche Informationen an der richtigen Stelle zur Verfügung stehen.

/NF40/ **Skalierbarkeit**: Hier bezieht sich der Begriff Skalierbarkeit vor allem auf Anfragen an das Backend. Angenommen die Extension erreicht eine hohe Nutzerzahl: Dadurch steigt das Risiko auf Überlastung des Servers. Um das zu verhindern, werden bei der Informationsgewinnung zwei Aspekte besonders wichtig: zum einen wie aktuell die Informationen sein sollen. Je aktueller, desto öfter müssen Anfragen gesendet werden. Zum anderen die Relevanz: Wie schnell müssen welche Informationen vorhanden sein und welche Informationen, die eine Analyse erfordern, werden erst auf spezielle Anfrage des Nutzers angefragt? Diese Anforderung stellt einen zentralen Punkt in der Entwicklung der Extension dar und wird in Aufgabe **??** detailliert behandelt.

/NF50/ **Datenschutz**: Bei allen Webdiensten spielt der Datenschutz eine wichtige Rolle. Auch in diesem Programm sollen Daten gespeichert werden um die Anforderung /NF40/ zu unterstützen. Um Datenschutzbedenken auszuschließen, muss das Format der Daten so gewählt werden, dass diese nicht personalisiert werden und nach Möglichkeit komplett lokal gespeichert werden.

/NF60/ **Korrektheit der Daten**: Alle Informationen zu Applikationen, die dieses Programm darstellt, werden extern von einem Server des privacy guard-Projekts eingespeist. Dieser gewinnt die Daten hauptsächlich auf automatischen Textmining-Verfahren. Ein Problem bei diesen Verfahren ist die fehlenden Validierung der Informationen. Ursachen wie das heterogene

Format von Datenschutzerklärungen und Mehrfachverlinkungen von Datenschutzinformationen können zu bei dieser Methode zu Fehlern oder Ungenauigkeiten führen. Aus diesem Grund muss dem Nutzer verdeutlicht werden, dass alle Angaben als Empfehlungen zu betrachten sind und keine verbindlichen Aussagen über Applikationen getätigt werden.

3.1.3 Aufbau der Website

Der Play Store, oder auch „Google Play“, ist die Haupt-Vertriebsplattform von Google für digitale Güter wie Apps, Filme und Serien, Musik und Bücher. Diese Arbeit und somit die Entwicklung der Extension beschränken sich auf die Kategorie „Apps“. Zu finden unter dem URL:

„https://play.google.com/store/apps“

Die Seite unterteilt sich durch ein linksbündiges Menü (siehe Abbildung 3.4 Nr.1) in die Bereiche „Einkaufen“ und „Meine Apps“. Der Bereich „Einkaufen“ ist in die drei Reiter (siehe Abbildung 3.4 Nr.2) „Startseite“, „Top-Charts“ und „Neuerscheinungen“ gegliedert. Links neben diesen Reitern befindet sich eine Auswahl für einzelne Kategorien.

Alle Auswahlpunkte zeigen im Darstellungsbereich (siehe Abbildung 3.4 Nr.3) Apps in der gleichen Struktur an. Wie in Abbildung 3.4 zu erkennen, werden zu einer Thematik mehrere Kacheln (siehe Abbildung 3.4 Nr.4) in einer Reihe dargestellt. Durch den Button „Mehr“ klappt die entsprechende Reihe aus. Aufgeklappte Themengebiete, Suchergebnisse und „Meine Apps“ werden als Raster dargestellt.

Durch den Klick auf eine Kachel lädt der Nutzer die Detailseite(Abbildung 3.5). Diese zeigt neben der ausgewählten App auch ähnliche Programme an der rechten Seite an. Auf der Basis aller hier gezeigten Informationen trifft der Nutzer die Entscheidung, ob die App installiert werden soll.

Für die Funktionen der Erweiterung sind die Kacheln mit Informationen zu einzelnen Apps ausschlaggebend. Insgesamt unterscheidet die Website in drei Arten von Kacheln:

- **Kleine Kachel**: Wie in Abbildung 3.4 zu sehen, füllen kleine Kacheln alle Übersichtsseiten und sind als Reihe oder Raster angeordnet. Sie bestehen aus drei Bereichen. Das ist Vorschaubild oben, der Titel der App mit Herausgeber in der Mitte und die Bewertung unten zusammen mit dem Kaufpreis, falls vorhanden.

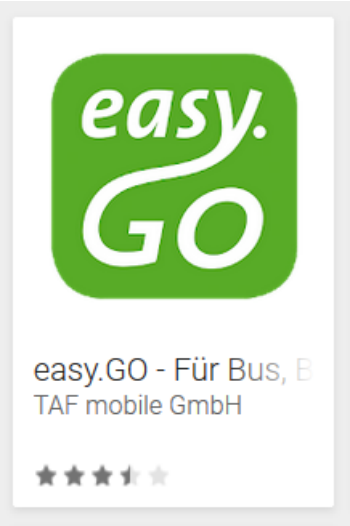

Abbildung 3.1: Kleine Kachel

- **Mittlere Kachel**: Als Raster unter dem Menü-Punkt „Meine Apps“ und als vertikale Reihe neben einer großen Kachel in der Detailansicht (siehe Abbildung 3.5) werden mittlere Kacheln eingesetzt. Diese, im Querformat dargestellte, Variante nimmt mehr Platz ein und bietet mehr Informationen. Das Vorschaubild ist links. In der rechten Hälfte oben befindet sich der Titel mit Herausgeber, darunter die Kurzbeschreibung der App. Am rechten unteren Rand sitzt die Bewertung mit dem Kaufpreis.

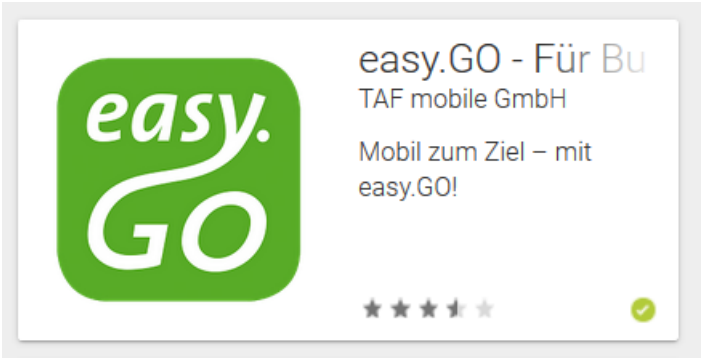

Abbildung 3.2: Mittlere Kachel

- **Große Kachel**: Jede App besitzt eine Detailansicht auf einer separaten Seite (siehe Abbildung 3.5). Diese Details werden in der großen Kachel dargestellt. Der obere Teil ist nahezu identisch aufgebaut wie die mittlere Kachel. Darunter folgen Vorschaubilder, eine ausführliche Beschreibung,

Nutzerbewertungen, der Updateverlauf und zusätzliche Informationen in Steckbrieffform.

Abbildung 3.3: Große Kachel

Der in diesem Kapitel beschriebene Stand der Website bezieht sich auf den Zeitraum von März 2018 bis März 2019. Die Planung und Implementierung der Extension ist auf diesen optischen und technischen Stand der Website angepasst. Mögliche Probleme, die ein veränderter Stand der Website mit sich bringt, werden in Abschnitt 3.1.6 diskutiert.

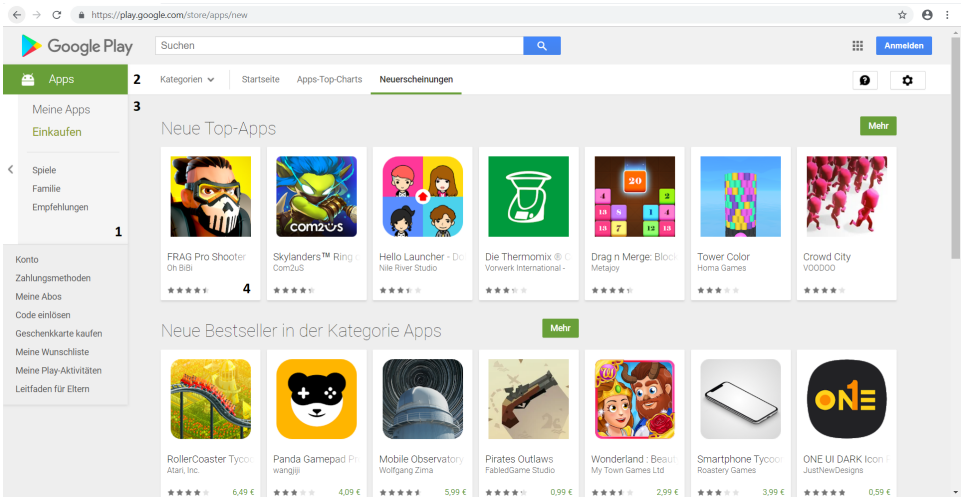

Abbildung 3.4: Kategorie Apps im Google Play Store (1: Apps-Menü; 2: Reiter; 3: Anzeigebereich der Apps; 4: Einzelne Kachel)

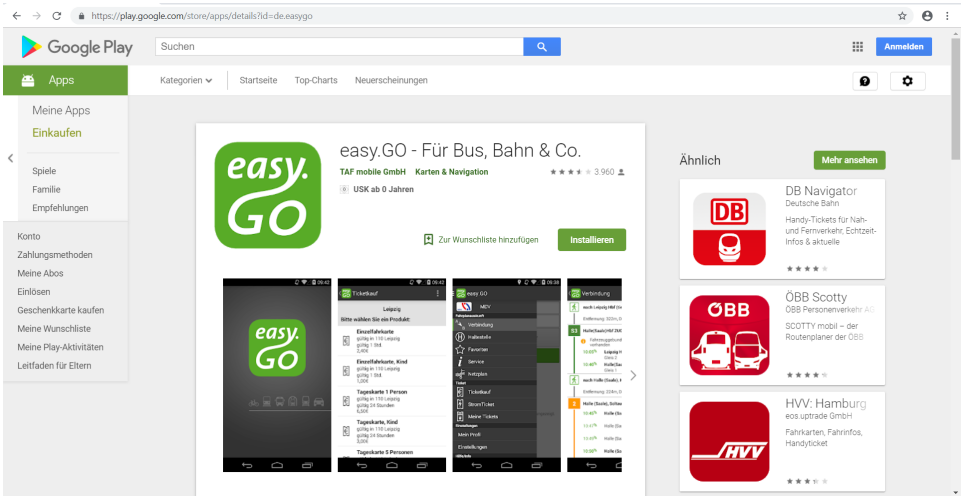

Abbildung 3.5: Detailansicht einer App

3.1.4 Programmaufbau

Die Implementierung des Programms orientiert sich an den beschriebenen Richtlinien in Kapitel 2.3.2. Dieser Abschnitt befasst sich mit der Umsetzung der Anforderungen aus Kapitel 3.1.2.1. Dabei werden relevante Ausschnitte des Quellcodes betrachtet und getroffene Entscheidungen begründet. Außerdem werden kritische Stellen beleuchtet und in Kapitel 3.1.6 diskutiert.

Das Manifest stellt die grundlegenden Zusammenhänge der Extension dar, un-

ter anderem den gewählten Entwicklungsnamen der Extension „PrivacyGuard App-Rating“ und die entsprechende Beschreibung. Weiterhin sind die nötigen Berechtigungen aufgeführt:

- **storage**: Berechtigung zum Zugriff auf Speicherplatz, um Informationen aus Backend-Anfragen zu speichern. Details zu konkreten Möglichkeiten der Speicherplatznutzung behandelt Kapitel 3.2.1.
- **declarativeContent**: Bereitstellung von Events, wie Seitenaufruf oder Seitenänderung und damit zusammenhängende Regeln, wie das Ausführen von Content-Skripten. Diese API wird vom Background-Skript genutzt,welches die genannten Aufgaben umsetzt.
- **activeTab** und **tab**: Gibt an, ob sich der Nutzer gerade in einem Tab befindet auf dem die Extension aktiv ist.

Außerdem werden alle Dateien ihren Rollen zugewiesen:

- **Content-Skript**: Unter dem Punkte „content scripts “ wird festgelegt, welche Skripte unter welchem URL aktiv sind. Durch die Eigenschaft „run_at: document_end “ wartet die Extension ab, bis die Seite fertig geladen ist, bevor das Skript startet.

  Der Ausdruck „*://play.google.com/store/apps*“ bedeutet, dass die Extension auf jeder Play Store-Seite der Kategorie Apps und deren Unterverzeichnis aktiv ist. Da es sich um eine „page action“ Extension handelt, wird lediglich eine Website als „match “ aufgeführt. Die beiden wichtigen Dateien hier sind „pguard.js“ als das Content-Skript für sämtliche Funktionen die die Erweiterung der Website betreffen und „popup-controller.js“ für alle Funktionen des Popups. Hinzu kommen sämtliche Bibliotheken, welche von den Content-Skripten benötigt werden.
- **Background-Skript**: Das Background-Skript „background.js“ fungiert als Eventhandler der Extension und ist deshalb separat im Manifest aufgeführt.
- **web_accessible_resources**: Diese Ressourcen sind Dateien, welche der Extension zur Verfügung stehen, aber selber keine Skripte sind. Sie beinhalten ausgelagerte Informationen wie Fließtexte und Templates zum Bauen von HTML-Elementen. Die „popup.html“ ist hier ein Sonderfall und wird direkt dem Popup zugewiesen.

```
{
  "name": "PGuard AppRating",
  "description": "rates PlayStore Applications based on 30 data safety criteria",
  "version": "0.1",
  "page_action":{
    "default_popup": "popup/popup.html"
  },
  "manifest_version": 2,
  "permissions": ["storage", "declarativeContent", "activeTab", "tabs" ],
  "content_scripts": [
    {
      "matches": ["*://play.google.com/store/apps*"],
      "js": ["lib/js/jquery-3.3.1.min.js", "lib/js/popper.js", "lib/js/bootstrap.min.js",
      "lib/js/fontawesome-all.js", "pguard.js", "popup/popup-controller.js"],
```

```
14        "css": ["lib/css/bootstrap.min.css", "lib/css/multiapp.css"],
15        "run_at": "document_end"
16      }
17    ],
18    "background": {
19      "scripts": ["lib/js/jquery-3.3.1.min.js","background.js"],
20      "persistent": false
21    },
22    "web_accessible_resources":[
23      "lib/data/*.json",
24      "lib/templates/*"
25    ]
26  }
```

Listing 3.1: Aufbau der manifest.json

Die Background.js besteht lediglich aus Callback-Funktionen der declarativeContent API. Hier wird zur Installation der Extension ein Listener eingebunden. Dieser funktioniert mit Regeln nach dem Konditionen-Aktionen-Prinzip. Zum Start des Aufrufs werden alle bereits vorhandenen Regeln des Listeners entfernt und anschließend die übergebenen Regeln installiert. Hier benötigt das Programm eine Regel. Die Kondition prüft, ob die passende URL aufgerufen wurde. Dieser stimmt mit dem String aus der Manifest-Datei überein. Ist die Kondition erfüllt, aktiviert sich das Popup.

Das Content-Skript „pguard.js“ bildet den Hauptteil der Extension und dient zur Erfassung aller auf der Webseite dargestellten Apps, der Einbettung von zusätzlichen Informationen durch das PGuard-Backend und optischen Anpassung der Webseite, um die neuen Inhalte ordentlich einzubinden.

Bei Skript-Aufruf werden zuerst die lokalen Bibliotheken der Extension geladen. Dazu gehören die IB texte.json sowie die HTML-Templates. Außerdem überprüft das Skript, ob lokaler Speicher zur Verfügung steht.

Anschließend prüft die Funktion „fillApps“ (siehe Listing 3.2), ob die aktuelle Seite eine Single-App-Page (Detailseite) oder Mutli-App-Page (z.B. die Startseite) ist und ermittelt sämtliche Kandidaten, welche für die Einbettung der Informationen in Frage kommen. Dabei liest der JQuery-Selektor alle Elemente mit dem entsprechenden Klassennamen aus.

Mit der Funktion „loadInfoPanels“ (siehe Listing 3.3) wird jeder so gefundene Kandidat auf seine APP-ID überprüft. Diese befindet sich entweder in dem Attribut „data-docid“ oder „href“. Mithilfe dieser ID durchsucht die Funktion „getStorageItem“ den lokalen Speicher auf Einträge. Der Eintrag ist valide, falls er nicht leer ist und vor weniger als drei Tagen angelegt wurde. Findet die Funktion keinen validen Eintrag im lokalen Speicher, wird eine neue Anfrage an das Backend für die entsprechende APP-ID erstellt.

Liefert das Backend eine Antwort mit mindestens einem Ergebnis, speichert die Funktion die Informationen im lokalen Speicher ab. Bei mehr als einem Ergebnis entscheidet eine Prioritätenliste abhängig von der zuverlässigsten Quelle, welcher Datensatz genutzt wird.

Der gewählte Datensatz wird der Funktion „createPanel“ übergeben. Handelt es sich bei dem Kandidaten um eine Detailseite, wird das Popover aus den Informationen direkt erstellt. Dazu wird der Datensatz mit Hilfe der IB texte.json in den entsprechenden Text umgewandelt und in die Html-Template einfügt. Bei kleinen App-Kacheln baut die Funktion einen Banner in die Kachel ein. Auf diesem Banner wird mittels JQuery ein Event geladen, welches bei einem Klick auf den Banner das Popover erstellt.

```
1  //Liesst alle geladenen Kacheln aus und übergibt diese zum Einfügen der Informationen
2  function fillApps(){
3          //Anfrage, ob Extension aktiv ist.
4          chrome.runtime.sendMessage(
5              {extensionState: "get"}, function (response) {
6
7                  if(response == "on"){
8                      console.log("PGuard-AppRating online");
9
10                     //Prueft, ob Single-App-Ansicht (Detailseite) oder Multi-App-Ansicht
11                     if(document.getElementsByClassName("card")[0]){
12                         $(".card").each(function () {
13                             loadInfoPanels(this, false);
14                         });
15                     } else {
16                         if(document.getElementsByClassName("JHTxhe")[0]){
17                             loadInfoPanels(document.getElementsByClassName("JHTxhe")[0], true);
18                         }
19                         $(".Vpfmgd").each(function (){
20                             loadInfoPanels(this, false);
21                         });
22                     }
23                 } else {
24                     console.log("PGuard-AppRating offline");
25                 }
26             }
27
28         );
29 }
```

Listing 3.2: Die Funktion *fillApps* in pguard.js

```
1  //Erstellt die Header fuer die Multi-App-Ansicht
2  function loadInfoPanels(parentNode, isSinglePage) {
3      var appID;
4      if(isSinglePage){
5          var currentURL = new URL(location.href);
6          appID = currentURL.searchParams.get("id");
7      } else {
8          if($(parentNode).attr("data-docid")){
9              appID = $(parentNode).attr("data-docid");
10         }else {
11             appID =
   $(parentNode.children[0].children[0].children[3].children[0]).attr("href").split("id=")[1];
12         }
13     }
14     //Wurde eine App-ID gefunden?
15     if (appID) {
16
17         var appDataString = getStorageItem(appID);
18
19         var useStorageItem = function(item) {
20             var appDataArray = [];
21             //Prueft ob bereits Daten im localStorage vorhanden und aktuell sind.
22             if(isStorageWorking && item && item.split(trennZeichen)[0]){
23                 var lastUpdate = new Date(item.split(trennZeichen)[0] * 86400000);
24                 if((lastUpdate.getTime() + 259200000) >= today.getTime()){
25                     appDataArray = item.split(trennZeichen);
26                 }
27             }
28             //Falls Daten vorhanden, baue das Element darauss
29             if(isStorageWorking && appDataArray.length === 1){
30                 createPanel(parentNode, appDataArray, false , isSinglePage);
31             } else if(isStorageWorking && appDataArray.length > 1){
32                 createPanel(parentNode, appDataArray, true , isSinglePage);
33             } else {
34
35                 //Anfrage an die Background.js zum Einholen der Informationen vom Backend
```

```
36                  chrome.runtime.sendMessage(
37                      {contentScriptQuery: "appIDQuery", appID: appID},function (response) {
38                          if (response.dses && response.dses.length > 0) {
39                              var storageString =
      castDseToStorageString(getNewestDseFromData(response));
40                              setStorageItem(appID,storageString);
41                              console.log("Neuer Eintrag angelegt: ", appID, storageString);
42                              appDataArray = storageString.split(trennZeichen);
43                              createPanel(parentNode, appDataArray, true, isSinglePage);
44                          } else {
45                              console.log("Keine DSE vorhanden für: ", appID);
46                              setStorageItem(appID,""+ Math.floor(today.getTime() /86400000));
47
48                              createPanel(parentNode, [], false, isSinglePage);
49                          }
50                      });
51              }
52          };
53
54          //Erzeugt Promise mit Storage-Element
55          if (typeof appDataString === 'object' && usedStorage === "indexedDB") {
56              appDataString.onsuccess = function() {
57                  if(appDataString.result){
58                      useStorageItem(appDataString.result.data);
59                  } else {
60                      useStorageItem();
61                  }
62              };
63
64          } else {
65              useStorageItem(appDataString);
66          }
67      }
68  }
```

Listing 3.3: Die Funktion *loadInfoPanels* in pguard.js

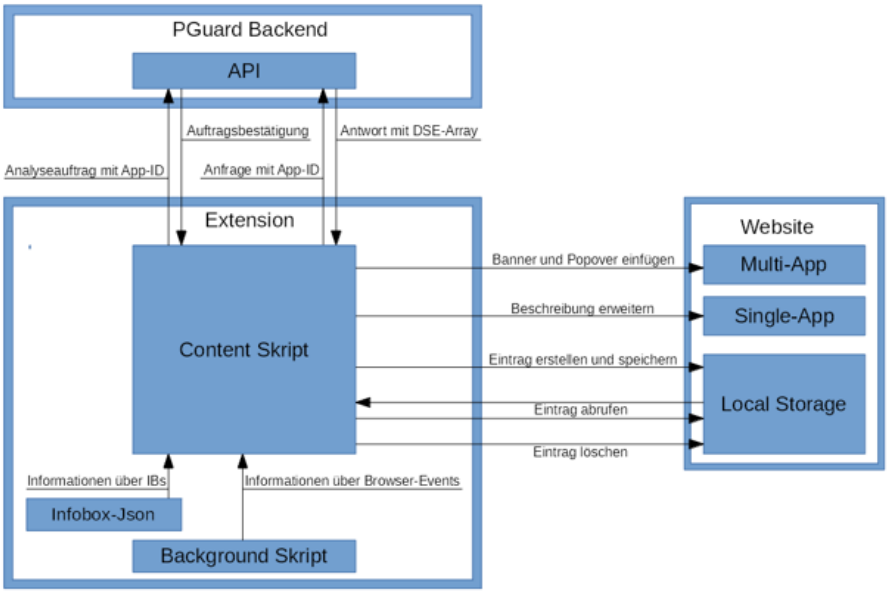

Abbildung 3.6: Aufbau und Interaktionen der Extension

3.1.5 Ergebnis

Die Browser-Extension stellt Datenschutzinformationen an mehreren Stellen der Webseite zur Verfügung. Bei kleinen und mittelgroßen Kacheln wird der erstellte Banner mit der Anzahl an gefundenen Infoboxen am oberen Rand eingeblendet (siehe Abbildung 3.7). Die Darstellung unterscheidet sich dabei je nach Status der Informationen:

- **„Keine Ergebnisse“**: Aufgrund von technischen Problemen liefert das Backend keine Datenschutzinformationen zu der angefragten Applikation.

- **Blauer Banner**: Informationen zum Datenschutz über die Applikation sind vorhanden, aber es liegt kein möglicher Gesetzesverstoß oder erheblicher Nachteil für den Nutzer vor.

- **Roter Banner („Rote Linie“)**: Informationen zum Datenschutz über die Applikation sind vorhanden und es liegt ein möglicher Gesetzesverstoß bzw. ein erheblicher Nachteil für den Nutzer vor.

Durch einen Klick auf den Banner erscheint das jeweilige Popover, bestehend aus einem Fenster mit der Liste an gefundenen Infoboxen. Diese Boxen sind aufklappbar und beinhalten die jeweilige Beschreibung, Handlungsempfehlungen, Vor- und Nachteile.

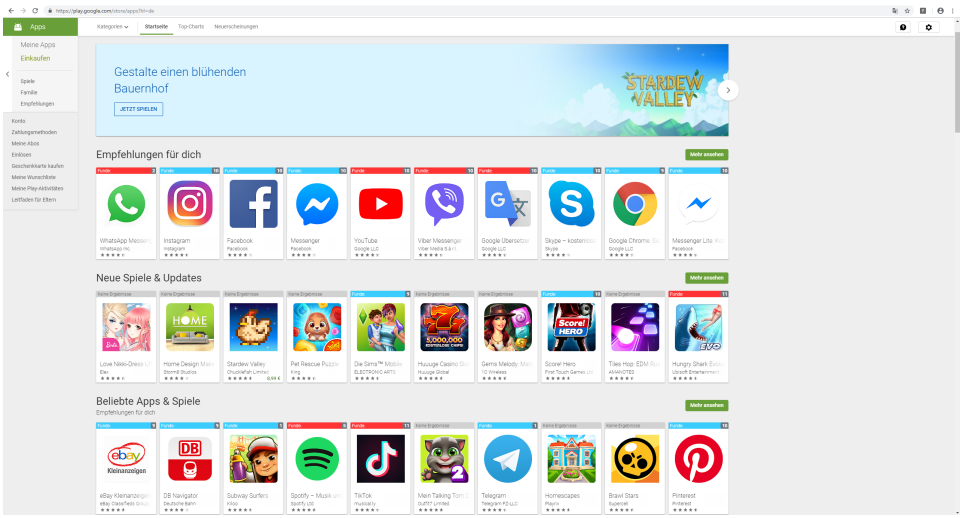

Abbildung 3.7: Ergänzung der Kacheln durch farbige Banner.

Auf den Detailseiten wird dieses Fenster direkt in die große Kachel zwischen der Programmbeschreibung und den Bewertungen eingebettet. Die Templates sind hier identisch zu denen im Popover.

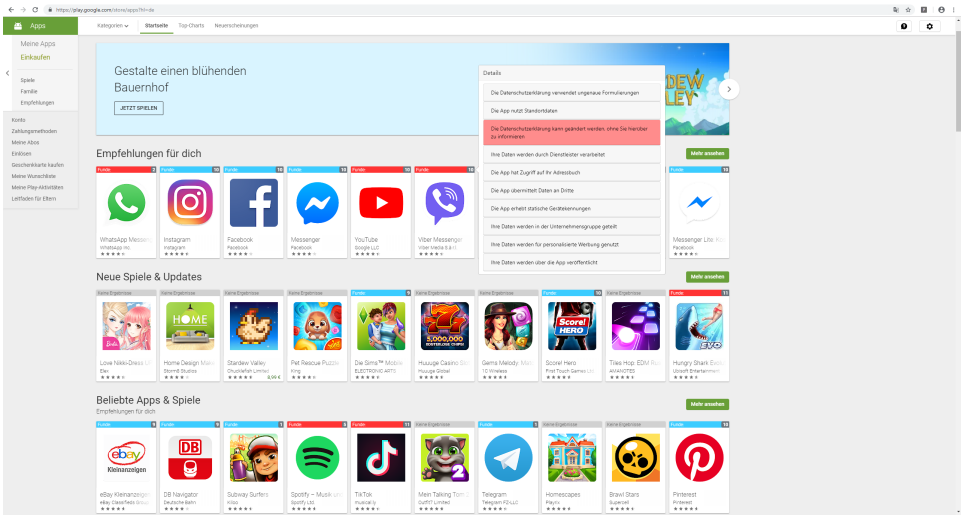

Abbildung 3.8: Popover mit der Auflistung an Infoboxen. „rote Linien“ sind entsprechend markiert.

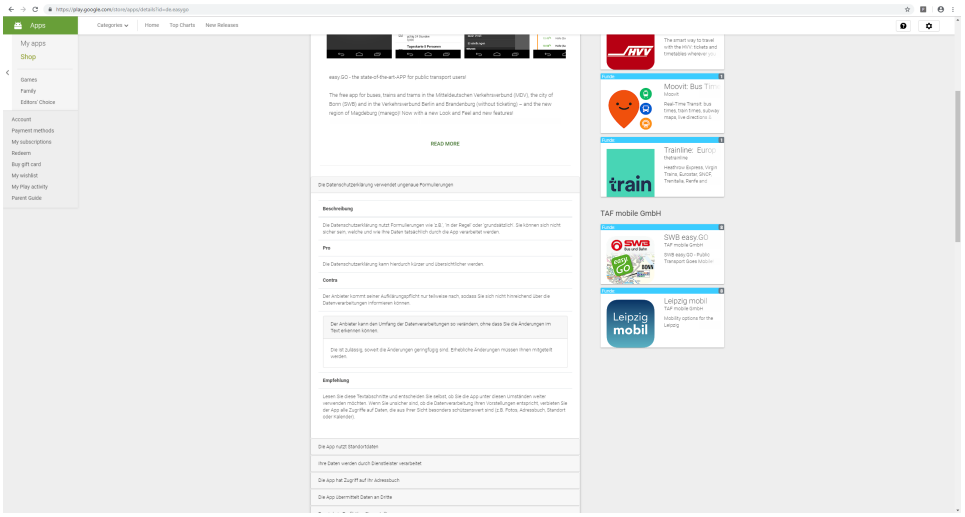

Abbildung 3.9: Detailseite mit eingefügten Infoboxen.

Alle vom Backend gewonnenen Informationen werden in eine lokale Datenbank abgespeichert und beim nächsten Auftauchen der gleichen Kachel wieder ausgelesen. Nach drei Tagen gilt die Information als veraltet und wird auf Abruf aktualisiert. Kapitel 3.2.1 befasst sich mit dem genauen Aufbau der Datenbank.

Das Icon besitzt ein Popup-Menü mit zwei Buttons zum Aktivieren/Deaktivieren der Extension und zum Löschen der angelegten Datenbank.

3.1.6 Diskussion

Im Verlauf der Programmierung sind technische und organisatorische Probleme aufgetreten. Hier werden einige davon erläutert und deren Konsequenzen aufgezeigt.

Da sich diese Browser-Extension sehr an der aktuellen Struktur der Website orientiert, können in Zukunft Fehler bei der Darstellung der Banner und Templates auf der Seite auftreten. Während der Implementierung traten bereits einige dieser Fehler auf.

Der Klick, welcher das Event zum Auf- und Zuklappen der Infoboxen auslöst, überlappt sich mit anderen Events der Webseite, auf die die Extension keinen Einfluss hat. Generell erschwert die Undurchsichtigkeit des technischen Aufbaus der Website die Anpassung der Funktionen einer Extension, sodass bei der Implementierung das Testen aller eingefügten Elemente manuell geschieht und mit großem Mehraufwand verbunden ist. Beispielsweise signalisiert die Website nicht, ob das DOM vollständig geladen ist. Dadurch wird das Content-Skript manchmal zum falschen Zeitpunkt ausgeführt. Ein Lösungsansatz ist das Implementieren eines „MutationOberservers“[25]. Das sorgt jedoch für Einbußen bei der Performanz.

Weiterhin überarbeitet Google die Website in regelmäßigen Abständen. Die in dieser Arbeit getroffene Aufteilung in Kacheln kann mit der nächsten Überarbeitung bereits obsolet sein. Aufgefallen ist das bei Betrachtung der Website in den letzten Monaten. Dabei haben sich bereits einige Klassennamen von Elementen der Kacheln geändert, sodass die Banner in falscher Größe oder gar nicht dargestellt wurden. Beispielweise der Klassenname „JHTxhe “ (siehe Listing 3.2, Zeile 16)

Im Allgemeinen ist die Browser-Extension in ihrem aktuellen Zustand nicht „marktreif “. Das heißt, es bedarf weiterer Überarbeitung und Organisation, bevor man die Extension im Web-Store von Google anbieten kann (siehe Ausblick, Kapitel 4.2).

Aufgrund der Evaluation von möglichen lokalen Datenspeichern für diese Extension sind sowohl „IndexedDB“ als auch „LokalStorage“ implementiert. Für spätere Verbraucher müsste eine der beiden Methoden entfernt werden, um unnötige Verwirrung im Umgang mit der Extension zu vermeiden.

Bereits in der Implementierungsphase sind inkonsistente Datensätze aufgefallen. Eine genauere Überprüfung ergab, dass die automatisierte Informationsgewinnung und Verarbeitung teilweise fehlerhaft ist. Dazu zählen das Crawlen von Datenschutzerklärungen aus unzuverlässigen Quellen und das Priorisieren der falschen Sprache. Dadurch kann die Richtigkeit der anzeigten Informationen nicht vollumfänglich garantiert werden. Auf dieser Basis wäre eine Funktion zur Empfehlung von alternativen Apps nicht sinnvoll.

Das Forschungsprojekt PrivacyGuard wurde im Juni 2018 beendet und somit auch

die Verwaltung aller in diesen Rahmen entstandenen, Produkte. Deshalb gibt es zum aktuellen Zeitpunkt keinen Verantwortlichen für die Veröffentlichung und Wartung des Backends und dieser Extension.

3.2 Aufgabe 2: Evaluierung von Caching-Methoden einer Browser Extension

3.2.1 Speichern von Informationen

Bevor diese Arbeit mögliche Methoden zur Speicherung von Datensätzen evaluiert, werden zuerst grundlegende Fragen beantwortet:

Warum benötigt die Extension einen Cache?: In Kapitel 3.1.3 zeigt die Abbildung 3.4 den Aufbau der Internetseite nach bestimmten Themen. Jedes Thema wird mit einer Reihe von Apps dargestellt. Je nach Bildschirmauflösung beinhaltet eine Reihe sieben bis zehn dieser Kacheln. Beim initialen Aufruf lädt die Seite acht Themen. Weitere Reihen werden dynamisch beim Scrollen nachgeladen. So findet die Extension zwischen 60 bis 80 App-IDs allein durch den Aufruf der Seite. Wird ein Thema durch den Button „Mehr“ aufgeklappt, lädt die Seite 120 bis 540 weitere Kacheln. Dabei sind Applikationen oft in mehr als einem Thema vorhanden und bereits auf der Startseite doppelt oder dreifach abgebildet.

Da die Extension pro gefundener App-ID eine Anfrage an das Backend schickt, entstehen pro Seitenaufruf mindestens 60 und pro Klick auf ein Themengebiet mindestens weitere 120 Anfragen. Also würde ein Nutzer bei einem Besuch der Seite grob geschätzt mehrere hunderte Backend-Anfragen auslösen.

Da diese Webseite als Such- und Einkaufsplattform fungiert, bleibt es in der Regel nicht bei einem einzigen Besuch. Hinzu kommt also eine hohe Redundanz der Anfragen bei erneutem Besuch der Seite. Denn viele Themen und Kategorien, wie zum Beispiel „Empfehlungen für dich “, sind personalisiert und bleiben über eine Vielzahl an Seitenaufrufen identisch.

Zusammenfassend entstehen bei der Nutzung des Google Play Stores also eine hohe Anzahl an benötigten Informationen mit einer Vielzahl an Redundanzen sowohl bei einem Besuch als auch über mehrere Sitzungen hinweg. Die Veröffentlichung der Extension würde also einen hohes Anfrageaufkommen an das Backend verursachen. Durch eine folgende Überlastung könnte die Extension keine Informationen mehr darstellen und hätte keinen Nutzen mehr.

Der Lösungsansatz ist ein vom Backend unabhängiger Speicher, welcher gewonnene Informationen für den Nutzer bereithält. Vor allem langlebige und redundante Daten stehen so ohne wiederholte Anfragen zur Verfügung.

Welche Informationen sollen im Cache gespeichert werden?: Damit nicht nur die Anzahl der Anfragen an das Backend, sondern auch der Umfang der Daten

möglichst gering bleibt, werden die nötigten Informationen in ihrer komprimierten Form angefragt. Dem Backend wird durch bestimmte Parameter signalisiert, nur die Kennzahl der zutreffenden Infobox (siehe Tabellen 2.1 und 2.2) zusammen mit der Extraktionsquelle und dem Datum der Extraktion zu der jeweiligen App-ID zu übermitteln.

So entsteht folgender key-value-Datensatz:
{

key: *App-ID*, **value**: *Extraktionstag in Tagen,Array von Infoboxen als Zahlen* }

```
{appID: "air.com.goblin.timetoescape", data: "17971|1|5"}
{appID: "air.com.goodgamestudios.empirefourkingdoms", data: "17971|1|3|6|9|12|17|23|27|30"}
```

Abbildung 3.10: Beispiel eines key-value-Datensatzes

Die gespeicherten Indizes der Infoboxen werden lokal von der Extension über die Datei „IB_texte.json “ (Auszug in Abbildung 3.4) auf ihre entsprechenden Texte gemappt, sodass diese nicht über das Backend abgefragt werden müssen. Der Extraktionstag dient zur Feststellung des Alters eines Datensatzes. Aktuell ist festgelegt, dass eine Information dann als veraltet gilt, wenn ihr Extraktionsdatum älter als drei Tage ist. Erst sobald die Zeitspanne überschritten wurde, sendet die Extension mit dem nächsten Aufruf der jeweiligen Applikation eine neue Anfrage.

```
1  {
2    "id": "19",
3    "is_red_line": "false",
4    "titel": "Die App verarbeitet Daten, die für die Funktion der App nicht erforderlich sind",
5    "description": "",
6    "pros": [
7      {
8        "first_layer": "Dies ermöglicht es, die App um Komfortfunktionen zu erweitern (z.B.
       Kalendereinträge, Kartendarstellung).",
9        "second_layer": ""
10     }
11   ],
12   "cons": [
13     {
14       "first_layer": "Es werden unnötige Daten erhoben.",
15       "second_layer": ""
16     },
17     {
18       "first_layer": "Ihre Daten können zu detaillierten Profilen zusammengeführt werden.",
19       "second_layer": ""
20     },
21     {
22       "first_layer": "Ihre Daten können missbraucht werden.",
23       "second_layer": ""
24     }
25   ],
26   "recommendations": "",
27   "actions": [
28   ]
29 },
```

Listing 3.4: Infobox 19 aus der IB_texte.json

Wie viele Informationen können gespeichert werden?: Für die oben gezeigte Struktur der Datensätze ergibt sich folgender *worst-case*:

key: { *max. 50 characters*[1] }

value: { *99999,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,*

18,19,20,21,22,23,24,25,26,27,28,29,30,31 }

Das entspricht einer Länge von 50 characters für den key und 89 characters für den value. In UTF-8 sind das umgerechnet ca. 139 Byte. Geht man in Google Chrome von einem Limit des Cache-Speichers auf 5200000 characters[2] aus, können so ca. 37410 Datensätze gespeichert werden.

3.2.2 Kriterien und Vorauswahl

Die Kriterien für eine geeignete Caching-Methode leiten sich aus den Anforderungen aus Kapitel 2.4.1 ab. Zur Auswahl stehen nach Abschnitt 2.4.2 Cookies, WebSQL, die Web-Storage-API und IndexedDB. Nur Kandidaten, welche alle Kriterien erfüllen werden, implementiert und darüber hinaus evaluiert.

Die Grundlage für einen geeigneten Cache ist ausreichend Speicherplatz. Während alle anderen Kandidaten mindestens 5MB Speicherplatz zur Verfügung haben, besitzen Cookies mit 4KB Gesamtspeicher pro Domäne, und somit nach der Berechnung in Abschnitt 3.2.1 mit ca. 28 Datensätzen, zu wenig Speicher für die Extension.

Das nächste Kriterium ist die Verfügbarkeit. Die Caching-Methode soll uneingeschränkt und überprüfbar zur Verfügung stehen, um die Speicherfunktion der Extension zu garantieren. Aufgrund der genannten EU-Richtlinie in Abschnitt 2.4.2 und dem Gebrauch von Cookies als Tracking-Methode beschränken oder blocken viele Nutzer diese Speichermethode. Das stellt eine Ungewissheit dar und widerspricht der uneingeschränkten Nutzung als Kriterium. WebSQL ist zwar weiterhin in einigen Browsern verfügbar, von einer Implementierung in neue Programme rät der W3C ab[34]. Durch diese Obsoleszenz erfüllt auch WebSQL das Kriterium nicht. IndexedDB wird als Nachfolger von WebSQL gesehen. Sowohl die Web-Storage-API als auch IndexedDB sind aktuelle Technologien und uneingeschränkt verfügbar.

Abschließend wird der Zugriff auf die Caching-Methoden betrachtet. Der Fokus liegt hier bei einem Zugriff mit möglichst geringer Wartezeit auf viele kleine Datensätze. Alle Kandidaten verfügen über asynchrone Aufrufe, sodass die Extension keine zusätzlichen Wartephasen verursacht. Durch eine fehlende Indexierung verliert die Web-Storage-API jedoch an Performanz, wenn es um das Auslesen von großen Datensätzen geht. Das angestrebte Key-Value-Prinzip wird von IndexedDB und der Web-Storage-API unterstützt. Bei WebSQL erfolgt der Zugrifft über die „Standard Query Language “ und die Datensätze lägen in Tabellenform vor, was die Handhabung mit den Datensätzen erschwert. Cookies speichern Informationen

[1] Gemessen durch Analyse aller App-ID-Einträge während der Evaluation

in Datenobjekte, welche die erwünschten Eigenschaften für den Zugriff ebenfalls erfüllen.

Weder Cookies noch WebSQL qualifizieren sich aufgrund der betrachteten Kriterien als geeignete Caching-Methoden. IndexedDB und die Web-Storage-API kommen beide in Frage und werden in den anschließenden Kapiteln anhand von Messergebnissen und ihren Eigenschaften bei der Implementierung evaluiert.

3.2.3 Vorgehensweise

Die Extension nutzt für die Evaluation den „localStorage“ der Web-Storage-API, da der „sessionStorage “ gespeicherte Daten nach Beenden der Sitzung löscht. Die vorgesehene Messung soll Daten über mehrere Sitzungen erfassen. *IndexedDB* wird ohne Framework implementiert.

In drei aufeinanderfolgenden Durchläufen werden hier der Aufruf der Website ohne Cache, mit vorhandenem *localStorage* und mit *IndexedDB* verglichen. Die Messung richtet sich nach einer normalen, kurzen Nutzung der Internetseite. Dazu gehört das Laden der Startseite, die Auswahl eines weiteren Reiters und das Öffnen einer Detailseite.

Gemessen wird dieser Prozess mittels der Browser-Konsole von Google Chrome im Reiter „Performance “. Dieser ist in der Lage, die Ladezeit einer Webseite mit Unterteilung in *Scripting* , *Rendering*, *Painting*, *Other* und *Idle* darzustellen. Die Messung fokussiert sich vor allem die Bereiche *Scripting* und *Rendering* bis zum Ende des *Painting*-Prozesses der Extension. Der zweite Messpunkt sind die Anzahl der Anfragen an das Backend. Diese werden im Reiter „Network “ausgelesen.

Durchführt wurden die Messungen auf der folgenden Plattform:

Komponente	**Eigenschaften/Version**
Prozessor	i7-6700K @ 4.00GHz (8CPUs)
Speicher	16384MB RAM
Grafik	GeForce GTX 1070
Auflösung	2560 x 1440
Betriebssystem	Windows 10 Education 64-Bit-Version (10.0, Build 17134)
Browser	Google Chrome Version 71.0.3578.80 64-Bit
Internetverbindung	Download: 100MBit/s , Upload: 6Mbit/s

Tabelle 3.1: Spezifikationen der Testumgebung

3.2.4 Ergebnisse

Die Ergebnisse der Messung stammen vom 17.03.2019. Jede Tabelle stellt den Aufruf einer Seite des Google Play Stores mit allen drei Durchläufen dar. Pro angewandter Methode wird die Gesamtladezeit der Extension in der oberen Reihe angegeben. Darunter befindet sich das Verhältnis von Anfragen an das Backend zu allen von der Extension gefundenen Apps auf der Seite.

Im ersten Durchlauf ist gut zu erkennen, dass die Ladezeit der Extension ohne gespeicherte Datensätze mit ca. 275 bis 295 Millisekunden immer ungefähr gleich bleibt. Dagegen unterscheiden sich die Zeiten im zweiten Durchlauf schon deutlich. Dadurch, dass im *localStorage* und der *indexedDB* bereits die Daten aller Apps aus dem vorherigen Aufruf abgespeichert sind, halbiert sich die Ladezeit der Extension mit *localStorage*. Mit *indexedDB* verringert sich die benötigte Zeit sogar um ca. 67%. Im letzten Durchgang bestätigen sich die Messungen, wobei die Extension hier mit beiden Caching-Methoden etwa gleich lange lädt.

Ursache für die kürzeren Ladezeiten sind hauptsächlich die 74 eingesparten Anfragen an das Backend ab dem zweiten Durchlauf. Weiterhin ist zu erkennen, dass beim Laden der Extension auf der Startseite unter gleichen Bedingungen eine Messschwankung von bis zu 20 Millisekunden auftritt.

Methode	**1. Durchlauf**	**2. Durchlauf**	**3. Durchlauf**
ohne Cache	292.8ms	292.0ms	294.8ms
	70/70	74/74	74/74
localStorage	283.6ms	123.0ms	108.5ms
	74/74	0/74	0/74
indexedDB	274.4ms	95.3ms	104.3ms
	74/74	0/74	0/74

Tabelle 3.2: Ladezeiten und Anfragen auf der Startseite

Der Wechsel zum Reiter „Top-Charts“ findet direkt nach dem Aufruf der Startseite statt. Der Vorteil dabei ist, dass die beiden Caches bereits 18 der 144 in dem Reiter gefundenen Apps von der Startseite gespeichert haben. Damit ist die Gesamtladezeit der Extension mit lokalem Speicher bereits im ersten Durchlauf geringer. Ungefähr 120 bis 160 Millisekunden, also bis zu 30%, werden dadurch eingespart. In den nächsten beiden Durchläufen halbiert sich die Ladezeit mit *localStorage* gegenüber der speicherlosen Variante, da erneut alle Applikationen bereits im Cache vorhanden sind und keine neuen Anfragen an das Backend geschickt werden müssen. Auffällig ist auf dieser Seite, dass *indexedDB* beide Male weniger Ladezeit benötigt als *localStorage* mit konstanten 67% Ersparnis.

Eine mögliche Ursache könnte die gestiegene Anzahl an abzuspeichernden Datensätzen sein. Außerdem sind die Schwankungen der Gesamtladezeit ohne Cache mit ca. 115 Millisekunden, also etwa 20%, hier wesentlich höher als auf der Startseite mit ca. 6%. Das deutet auf eventuelle Browser-seitige Optimierung für Erweiterungen hin.

Methode	**1. Durchlauf**	**2. Durchlauf**	**3. Durchlauf**
ohne Cache	559.8ms	469.8ms	445.6ms
	144/144	144/144	144/144
localStorage	439.5ms	213.4ms	214.7ms
	126/144	0/144	0/144
indexedDB	393.3ms	162.6ms	161.1ms
	126/144	0/144	0/144

Tabelle 3.3: Ladezeiten und Anfragen auf dem Reiter „Top-Charts “

Auf der Seite „Top-Charts “ wurde die App „WhatsApp “ ausgewählt und damit die Detailseite aufgerufen. Auf dieser Seite finden sich neben der ausgewählten App weitere Kacheln mit Empfehlungen. Von den acht angezeigten Apps, wurden alle bereits auf der Startseite oder unter „Top-Charts “ aufgelistet und somit von den Caching-Methoden erfasst. Lediglich beim ersten Durchlauf mit *indexedDB* wurden sechs neue Empfehlungen auf der Detailseite anzeigt. Ohne Cache schwankt die Ladezeit der Extension zwischen 61 und 71 Millisekunden. Durch die acht eingesparten Anfragen wird die Ladezeit mit Cache immerhin auf 43 bis 46 Millisekunden verringert. Das entspricht einer etwa 34% kürzeren Ladezeit der Extension.

Methode	**1. Durchlauf**	**2. Durchlauf**	**3. Durchlauf**
ohne Cache	61.1ms	69.5ms	71.1ms
	8/8	8/8	8/8
localStorage	44.2ms	44.0ms	46.0ms
	0/8	0/8	0/8
indexedDB	57.3ms	45.2ms	42.8ms
	6/8	0/8	0/8

Tabelle 3.4: Ladezeiten und Anfragen auf der Detailseite von „WhatsApp “

Zusammenfassend zeigen die Messwerte deutliche Unterschiede zwischen den Ladezeiten mit und ohne Cache. Während die Extension auf der Detailseite ca. 34% schneller lädt, verdoppelt bzw. verdreifacht sich die Geschwindigkeit durch den Gebrauch von *localStorage* oder *indexedDB*. In den hier gemessenen Durchläufen schlägt *indexedDB* den *localStorage* in puncto Ladezeiten. Gerade bei Seiten mit einer größeren Menge an Speicherzugriffen erzielt *indexedDB* bessere Ergebnisse.

Insgesamt konnten bereits beim ersten Durchlauf bis zu 26 der der 226 Anfragen eingespart werden und in darauffolgenden Durchgängen wurden keine Anfragen an das Backend mehr benötigt. Dadurch wird das Backend merklich entlastet.

3.2.5 Diskussion

Die gewonnenen Messwerte reichen aus um den Unterschied zwischen der Extension ohne Cache und mit Cache aufzuzeigen. Auch die einzelnen Caching-Methoden konnten unter diesen Bedingungen gut miteinander verglichen werden. Um genauere und umfassendere Ergebnisse zu erhalten, ist es natürlich möglich, weitere Szenarien auf der Website zu testen. Hinzu kommt, dass bei einer längeren Testphase mit mehreren Durchläufen die Ladezeiten weiter optimiert werden, da weitere Datensätze abgespeichert werden. In dem angewandten Szenario wurden 200 Einträge erstellt und die Kapazität der lokalen Speicher nicht ansatzweise erreicht.

Erhält die Extension keine Informationen zur einer App, schickt sie dem Backend eine Aufforderung, welche den Analyse-Prozess des Backends für diese App startet. Diese Aufforderungen wurden für die Messung deaktiviert, da sie keinen Einfluss auf die Ergebnisse haben und die Gewinnung der Messwerte nur erschweren.

Weiterhin werden hier nur Optimierungen in der Extension betrachtet. Veränderungen am Prozess für Anfragen an das Backend könnten, gerade bei einer hohen Anzahl an Anfragen, zusätzlich positive Auswirkungen haben. Das gleiche gilt für die Struktur von Anfragen und Antworten.

Nicht betrachtet wurde die Leistung der Extension im Mehrbenutzerbetrieb. In diesem Fall wäre eine weitere Maßnahme zur Vermeidung von zu vielen Anfragen, die Anpassung der Tage nachdem eine Information als veraltete gilt. Drei Tage sind eine relativ kurze Zeitspanne, da Datenschutzerklärungen in der Regel nur alle paar Monate geändert werden.

Kapitel 4

Zusammenfassung der Arbeit

4.1 Zusammenfassung

Im Rahmen dieser Arbeit entstand eine Browser-Extension zur Erweiterung des Google Play Stores um datenschutzrelevante Informationen. Dabei werden mit Hilfe des Backends von PrivacyGuard angezeigte Apps um Infoboxen mit Vor- und Nachteilen sowie Handlungsempfehlungen erweitert. Zusätzlich zeigen eingefügte Banner die Anzahl der Funde an und warnen den Nutzer vor sogenannten „roten Linien“.

Zu Beginn wurden dabei Recherchen angestellt, die zeigen, dass der Google Chrome Browser durch seinen hohen Marktanteil als Plattform am besten geeignet ist und keine Erweiterung mit den genannten Funktionen bereits existiert. Der zweite Teil der Vorarbeit hat sich mit dem genauen Aufbau und der Implementierung einer Chrome-Extension beschäftigt und welche Richtlinien dabei eingehalten werden müssen.

Anschließend wurde das Forschungsprojekt *PrivacyGuard* vorgestellt zusammen mit dem Backend, welches als Quelle für Datenschutzinformationen dient. Um dieses Backend nicht zu überlasten, besteht der Bedarf eines lokalen Speichers in der Extension. Welche Methoden zur Verfügung stehen und sich für diesen Anwendungsfall eignen, wurde ebenfalls Teil der Recherche.

Aus den gewonnenen Informationen entstand eine Anforderungsanalyse mit funktionalen und nicht-funktionalen Nutzerszenarien für die Browser-Extension. Zur bestmöglichen Umsetzung wurde die Website des Google Play Stores betrachtet und eine möglichst einfache aber klare Darstellung der Informationen gewählt. Bis auf die Empfehlung bei Suchanfragen wurden alle Anforderungen umgesetzt und das Ergebnis anschließend betrachtet und einige Kritikpunkte diskutiert.

Nach der Implementierung wurden die recherchierten Caching-Methoden anhand bestimmter Kriterien verglichen und zwei Kandidaten für die weitere Evaluation

ausgewählt. Diese Methoden wurde in die Erweiterung integriert. Messungen ergaben, dass die Extension durch das lokale Speichern von Informationen bis zu 67% ihrer Ladezeit einsparen konnte, welche hauptsächlich durch Anfragen an das Backend beeinflusst wurde. Dabei lag die Caching-Methode *indexedDB* vorne. Abschließend wurden Möglichkeiten diskutiert, um die Messungen zu erweitern und die Gesamtperformanz der Browser-Extrension weiter zu steigern.

4.2 Ausblick

Während der Umsetzung der Aufgabenstellungen ergaben sich bereits Punkte, die aus zeitlichen oder organisatorischen Gründen nicht mehr umgesetzt werden konnten. Dieser Ausblick beschreibt eine mögliche Weiterentwicklung der Browser-Extension und neue Punkte zur Verbesserung der Performanz.

Der erste und wichtigste Punkt ist die Veröffentlichung der Extension. Um das zu ermöglichen bedarf es folgender Schritte:

- **Wartung und Support**: Sowohl das Backend als auch die Extension müssen fortlaufend gewartet werden. Im Rahmen der Umsetzung der App kam es öfter zu Ausfällen im Backend aufgrund von Fehlern oder Speicherauslastungen. Auch die Extension selbst musste im Rahmen der Arbeit mehrmals angepasst werden, um alle Kacheln mit Apps zu erkennen. Auch neue Browser-Events wurden erst nach und nach entdeckt, was zu Lücken beim Laden von Apps führte. Nach einer Veröffentlichung benötigt die Extension also eine verantwortliche Person zur Wartung dieser und weiterer auftretender Probleme.

- **Evaluierung der Datenqualität**: Die automatische Datenverarbeitung läuft nicht durchweg zuverlässig und es treten hin und wieder Fehler bei der Ausgabe von Datenschutzerklärungen auf. Um nach der Veröffentlichung rechtliche Konsequenzen zu vermeiden und das Vertrauen der Nutzer nicht zu verlieren, muss die Qualität der Daten noch einmal überprüft werden.

- **Empfehlung bei Suchanfragen /F50/**: Stimmt die Datenqualität, kann auch das Nutzerszenario zur Empfehlung von alternativen Apps implementiert werden. Dadurch bekommt der Verbraucher Apps mit möglichst wenigen datenschutzrechtlichen Bedenken vorgeschlagen.

Optional kann die Extension auch auf andere Browser portiert werden. Dazu bieten Mozilla und Microsoft entsprechende Anleitung an([24, 21]).

Gegen Ende der Bearbeitungszeit wurde eine weitere Methode zur lokalen Datenspeicherung veröffentlicht: „File API “[33]. Diese ermöglicht es Dateien anzulegen und als lokalen Speicher zu nutzen. Aktuell wird die API allerdings nicht vollständig von allen Browsern unterstützt. Eine weitere Evaluation könnte zeigen, ob das Vorteile in puncto Performanz gegenüber den in dieser Arbeit vorgestellten

Caching-Methoden hat.

Weiterhin wurde *indexedDB* in dieser Arbeit ohne Frameworks umgesetzt. Unter Umständen bieten zum Beispiel „localForage “[17] oder „Dexie.js “[12] eine bessere Performanz.

Abschließend sei erwähnt, dass mit der Veröffentlichung von Google Chrome Version 73 am 12. März eine neue Richtlinie eingeführt wird. Die sogenannte „Cross-Origin-Resource-Policy “[8] soll „Spectre“-Angriffe[20] verhindern und vor kompromittierten Renderern schützt. Dazu können Http-Server über den Browser Anfragen von externen Seiten blockieren. Diese Richtlinie könnte auch Einfluss auf die Kommunikation zwischen der Extension und dem Backend haben und die API müsste entsprechend angepasst werden. Vorläufig wurde das Problem durch den Lösungsansatz von *Chromium.org*[9] behoben (siehe Listing 3.3, Zeilen 35 - 52).

Danksagung

An dieser Stelle möchte ich all jenen danken, die zum Gelingen dieser Bachelorarbeit beigetragen haben.

Ein besonderes Dankeschön geht dabei an Sascha Ludwig für seine hervorragende Betreuung. Nach erfolgreichem Abschluss des Moduls „Textmining“ bot mir Sascha eine Stelle als studentische Hilfskraft am Institut für Angewandte Informatik und dem Forschungsprojekt „PrivacyGuard“ an. Im Rahmen meiner Anstellung und in Absprache mit Prof. Dr. Gerhard Heyer entstand das Thema für diese Arbeit. An dieser Stelle möchte ich deshalb auch Prof. Heyer für das Ermöglichen dieser Bachelorarbeit danken.

Nach Abschluss des Projekts bis zur Abgabe übernahm Thomas Efer die Betreuung der Bachelorarbeit. Dafür möchte ich mich hier ebenfalls herzlich bedanken.

Ein weiteres Dankeschön geht an Dipl. Inf. Franziska Güttler und Nicole Mackus für den Workshop „Informatik Spezial“ über das Bearbeiten von wissenschaftlichen Arbeiten.

Nicht zuletzt danke ich Anna Bechert für das intensive Korrekturlesen dieser Arbeit.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Appvisory.com. *mediaTest digital GmbH*. URL https://appvisory.com/company. Stand: 03.2019.

[2] Arty.name. *Test of localStorage limits/quota*. URL https://arty.name/localstorage.html. Stand: 03.2019.

[3] R. Camden. *Client-Side Data Storage*. O'Reilly Media, 2016.

[4] Chrome.com. *Developer Program Policies*. . URL https://developer.chrome.com/webstore/program_policies. Stand: 03.2019.

[5] Chrome.com. *chrome.cookies*. . URL https://developer.chrome.com/extensions/cookies. Stand: 03.2019.

[6] Chrome.com. *Give Users Options*. . URL https://developer.chrome.com/extensions/options. Stand: 03.2019.

[7] Chrome.com. *Extensions Quality Guidelines*. . URL https://developer.chrome.com/extensions/single_purpose. Stand: 03.2019.

[8] Chromestatus.com. *Cross-Origin Resource Policy*. URL https://www.chromestatus.com/feature/4647328103268352. Stand: 03.2019.

[9] Chromium.org. *Changes to Cross-Origin Requests in Chrome Extension Content Scripts*. URL https://www.chromium.org/Home/chromium-security/extension-content-script-fetches. Stand: 03.2019.

[10] Datenschutz-scanner.de. *PrivacyGuard Backend*. . URL https://pgadmin.datenschutz-scanner.de/api/docs.html. Stand: 03.2019.

[11] Datenschutz-scanner.de. *DATENSCHUTZscanner by PrivacyGuard*. . URL https://datenschutz-scanner.de/das-projekt.html. Stand: 03.2019.

[12] Dexie.org. *dexie.js*. URL https://dexie.org/. Stand: 03.2019.

[13] Dsgvo-gesetz.de. *Transparente Information, Kommunikation und Modalitäten für die Ausübung der Rechte der betroffenen Person*. . URL https://dsgvo-gesetz.de/art-12-dsgvo/. Stand: 03.2019.

[14] Dsgvo-gesetz.de. *Informationspflicht bei Erhebung von personenbezogenen Daten bei der betroffenen Person.* . URL https://dsgvo-gesetz.de/art-13-dsgvo/. Stand: 03.2019.

[15] C. Ebert. *Systematisches Requirements Engineering.* dpunkt.verlag, 2014.

[16] Europa.eu. *Richtlinie 2002/58/EG des Europäischen Parlaments und des Rates vom 12. Juli 2002 über die Verarbeitung personenbezogener Daten und den Schutz der Privatsphäre in der elektronischen Kommunikation.* URL https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:de:HTML. Stand: 03.2019.

[17] Github.org. *localForage.* URL https://github.com/localForage/localForage. Stand: 03.2019.

[18] Infai.org. *Institut für Angewandte Informatik.* URL https://infai.org/. Stand: 03.2019.

[19] P. Mehta. *Creating Google Chrome Extensions.* Apress, 2016.

[20] Meltdownattack.com. *Meltdown and Spectre.* URL https://meltdownattack.com/. Stand: 03.2019.

[21] Microsoft.com. *Porting an extension from Chrome to Microsoft Edge.* URL https://docs.microsoft.com/en-us/microsoft-edge/extensions/guides/porting-chrome-extensions. Stand: 03.2019.

[22] Mozilla.org. *IndexedDB.* . URL https://developer.mozilla.org/de/docs/IndexedDB. Stand: 03.2019.

[23] Mozilla.org. *Browser storage limits and eviction criteria.* . URL https://developer.mozilla.org/de/docs/IndexedDB/Browser_storage_limits_and_eviction_criteria. Stand: 03.2019.

[24] Mozilla.org. *Porting a Google Chrome extension.* . URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Porting_a_Google_Chrome_extension. Stand: 03.2019.

[25] Mozilla.org. *MutationObserver.* . URL https://developer.mozilla.org/de/docs/Web/API/MutationObserver. Stand: 03.2019.

[26] Mozilla.org. *Web Storage concepts and usage.* . URL https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API. Stand: 03.2019.

[27] PGuard-tools.de. *PrivacyGuard Tools.* URL https://dseanalyser.pguard-tools.de/. Stand: 03.2019.

[28] Quadriga-hochschule.com. *Quadriga Hochschule.* URL https://www.quadriga-hochschule.com/. Stand: 03.2019.

[29] S. Robertson. *Mastering the Requirements Process: Getting Requirements Right*. Addison-Wesley Professional, 2012.

[30] Sriw.de. *selbstregulierung informationswirtschaft e.V.* URL https://sriw.de/. Stand: 03.2019.

[31] Statista.com. *Marktanteile der führenden Browserfamilien an der Internetnutzung weltweit von Januar 2009 bis Januar 2019*. URL https://de.statista.com/statistik/daten/studie/157944/umfrage/marktanteile-der-browser-bei-der-internetnutzung-weltweit-seit-2009/. Stand: 03.2019.

[32] Thinkwithgoogle.com. *Find out how you stack up to new industry benchmarks for mobile page speed*. URL https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/. Stand: 03.2019.

[33] W3.org. *File API*. . URL https://www.w3.org/TR/FileAPI/. Stand: 03.2019.

[34] W3.org. *Web SQL Database*. . URL https://www.w3.org/TR/webdatabase/. Stand: 03.2019.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Insbesondere versichere ich, dass alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche gekennzeichnet sind.

Alexander Prull, Leipzig, 22. März 2019