Universität Leipzig

Fakultät für Mathematik und Informatik
Institut für Informatik

– Bachelorarbeit –

Entwicklung einer Browser-Extention zur Anzeige von Datenschutzinformationen im PlayStore und Evaluierung von Caching-Methoden

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

First Supervisor	**Second Supervisor**	**External Supervisor**
Prof. Nummer 1	Prof. Nummer 2	Extern Nummer 1
ggg@informatik.uni-leipzig.de	ttt@uni-leipzig.de	rrr.eee@uuu.com
Fancy Computer Science	Institute of Rocket Science	Something AG

17. März 2019

Abstract

In dieser Arbeit wird sich mit den Eigenheiten der Browser Extension Programmierung auseinander gesetzt. Speziell geht es um Extension die Webseiten um bestimmte Informationen erweitern. Diese werden von einem Backend empfangen und zur Ladezeit der Seite eingespeist. Dabei setzt sich die Arbeit mit zwei Punkten auseinander. In erster Linie geht es darum die Ladezeit der Webseite durch das Anfordern von Informationen so wenig wie möglich zu beeinflussen. Also die Performance der Extension zu maximieren. Auf der anderen Seite wird durch die Nutzung der Extension von einer steigenden Nutzerzahl der Backend-Server mit einer steigenden Anzahl von Anfragen belastet. Um diese Probleme zu lösen werden in der Arbeit verschiedene Möglichkeiten zur Speicherung von Daten betrachtet und eine Auswahl der Methoden auf ihre Performance hin getestet.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Zeitalter der Digitalisierung werden große Mengen Informationen immer schneller und detaillierter verarbeitet. -?- Jeder, der im Internet unterwegs ist, hinterlässt dabei wertvolle, persönliche Daten. Dabei spielt der Datenschutz eine wichtige Rolle, denn nicht immer werden diese Daten freiwillig preisgegeben. So reguliert die Datenschutzerklärung, welche Nutzerdaten verarbeitet werden. Denn in dieser Erklärung muss jeder Dienstanbieter dem Nutzer zu Beginn des Nutzungsvorgangs über Art, Umfang und Zwecke der Erhebung und Verwendung personenbezogener Daten sowie über die Verarbeitung seiner Daten in Staaten außerhalb des Anwendungsbereichs [...] in allgemein verständlicher Form zu unterrichten"(Telemediengesetz Paragraph 13 1 1)

Das Projekt Privacy Guard hat sich damit beschäftigt, inwiefern diese Datenschutzerklärungen den Vorgaben entsprechen und im Speziellen analysiert, welche Applikationen unvollständige oder mangelhafte DSEs vorweisen. Im Rahmen dieses Projektes entstand die Idee, bereits vor der Installation von Anwendungen, deren Datenschutzerklärungen zu untersuchen und den Nutzer auf mögliche Bedenken hinzuweisen.

1.1 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"**

Hauptaugenmerk ist die Erläuterung von Browser-Extensions, Umsetzung eines Beispiels und Limitationen. Welche Arten von Speicher stehen einer Extension zur Verfügung und welche Performance-Ersparnisse kann durch Abspeichern von

Daten die die Extension wiederholt benötigt eingespart werden. Welche Entlastung erfährt der Server mit Backend. Aufbau und Einbindung des ausgewählten Kandidaten

1.2 Aufbau der Arbeit

Zu Beginn werden Recherche Ergebnisse vorgestellt und ausgewertet. Aus den dadurch gewonnenen Resultaten die Aufgaben genauer Definiert. Auf Basis der Recherche entsteht im 1. Teil eine Extension wobei der Fokus darauf liegt, dass diese möglichst übersichtlich bleibt und zur Evaluierung von Speichermethoden dient. Anschließend werden verschiedene Testläufe präsentiert bei denen bestimmte Methoden zur lokalen Speicherung von Daten unter den gleichen Rahmenbedingungen verwendet werden. Die Ergebnisse werden verglichen und den Erwartungen gegenübergestellt. Zuletzt wird ein Fazit gezogen.

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 Browser.

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 Quelle 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 Existieren bereits vergleichbare Extensions

Gesucht wurde nach einer Extension die auf der Play Store Seite den Nutzer datenschutzrelevante Informationen zu den angebotenen Apps liefert, Eine Datenschutzwertung im Playstore vergibt oder den Nutzer Apps nach Berechtigungen

die Apps vorschlägt. Extensions werden nach ihrer Kurzbeschreibung in den Suchergebnissen überprüft und bei nicht eindeutig Aufgabenbeschreibung die Infoseite aufgerufen(Bsp. Safe.ad im Web Store „ecosystem“).Nur deutsche und englische Ergebnisse werden berücksichtigt.

Die Recherche hat ergeben, dass unter den genannten Suchkriterien keine Chrome oder Firefox Extension gefunden wurde die 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%.

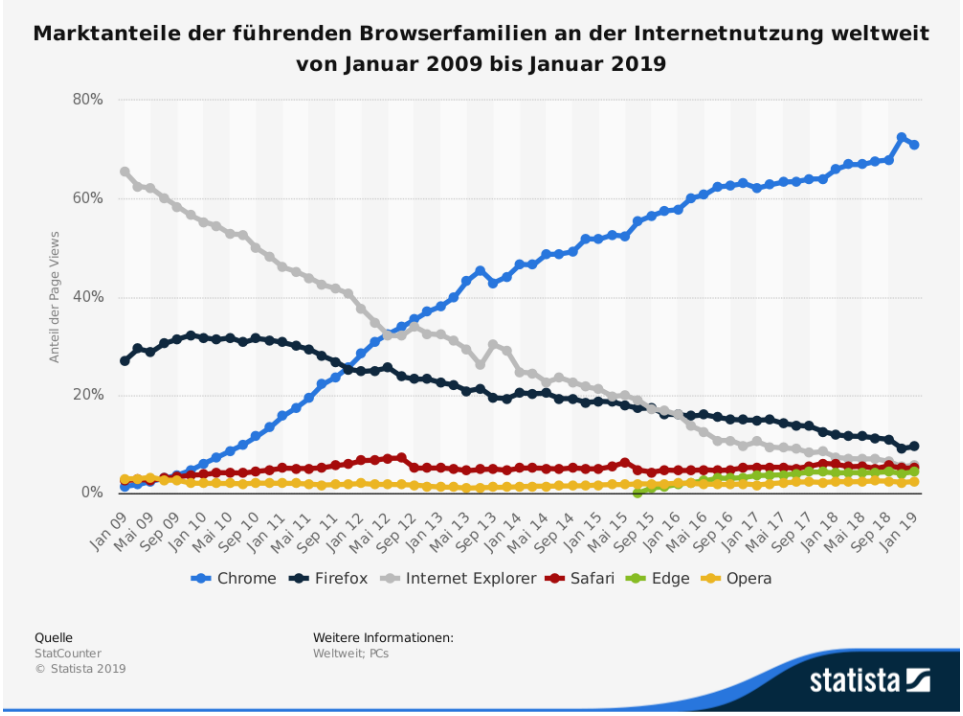

Abbildung 2.1: StatCounter. n.d. Marktanteile der führenden Browserfamilien an der Internetnutzung weltweit von Januar 2009 bis Januar 2019. Statista. Zugriff am 4. März 2019. Verfügbar unter https://de.statista.com/statistik/daten/studie/157944/umfrage/marktanteile-der-browser-bei-der-internetnutzung-weltweit-seit-2009/.

Aufgrund mangelnder Relevanz der Extension für Safari-Nutzer, sowie der Obso-

leszenz 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[1], als auch Edge[2] eine intuitive Lösung zur Portierung der fertigen Google Chrome-Extension.

2.2 PrivacyGuard

2.2.1 Vorstellung

Das Forschungsprojekt PrivacyGuard[3] wurde im Januar 2016 durch das Bundesministerium für Bildung und Forschung ins Leben gerufen. Das Institut für Angewandte Informatik[4], die mediaTest digital GmbH[5], die Quadriga Hochschule Berlin[6] und die selbstregulierung informationswirtschaft e.V.[7] 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.“(https://datenschutz-scanner.de/das-projekt.html,Stand 4.3.2019)

Die Ergebnisse des Projekts teilen sich in 3 Kategorien ein. Im Rahmen der Datenbeschaffung entstanden verschiedene Werkzeuge um Informationen über Apps zu extrahieren. Besonderer Wert wurde hier auf verlinkte Datenschutzerklärungen im Playstore 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[8], 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.

[1] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Porting_a_Google_Chrome_extension

[2] https://docs.microsoft.com/en-us/microsoft-edge/extensions/guides/porting-chrome-extensions

[3] https://datenschutz-scanner.de/das-projekt.html

[4] https://infai.org/

[5] https://appvisory.com/company

[6] https://www.quadriga-hochschule.com/

[7] https://sriw.de/

[8] https://pgadmin.datenschutz-scanner.de/api/docs.html

Mit dem Projektabschluss im Juni 2018 stellte PrivacyGuard eine Webseite zur Analyse von Datenschutzerklärungen und Apps zur Verfügung[9]. 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 PlayStore.

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 Ausführtlichkeit 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 Linießein. Das bedeutet, dass bei Besitz dieser Eigenschaft, die entsprechende App potentiell gegen ein Gesetz verstößt. Folgenden 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.

Grundvoraussetzung für eine funktionierende Extension ist die manifest.json, welche 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.

[9] https://dseanalyser.pguard-tools.de/

Externe Bibliotheken wie JQuery können ebenfalls eingebunden werden, müssen aber aufgrund der Policies[10] von Google Chrome vollumfänglich lokal vorliegen. Mehr dazu Im nächsten Abschnitt.

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 „permissions“ 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 (BILD?). Je nach Funktionsumfang können weitere UI-Elemente eingebunden sein, um zum Beispiel 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[11]. 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 soll den Nutzer darauf hinweisen, wo sich die Erweiterung einschaltet. Icons mit „browser-action“ sind permanent aktiv und zeigen somit, dass die Extension seitenübergreifende Funktionen hat.

[10] https://developer.chrome.com/webstore/program_policies
[11] https://developer.chrome.com/extensions/single_purpose

Auch inhaltlich gibt es bestimmte Richtlinien[12] 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.

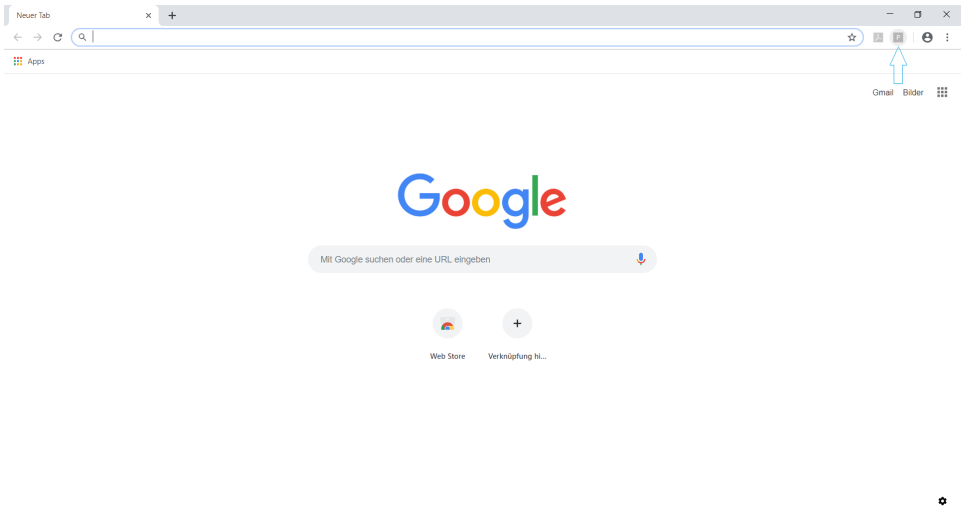

Abbildung 2.2: Browser-Extension Icon in der Adresszeile

[12] https://developer.chrome.com/webstore/program_policies

Eine weitere Möglichkeit Inhalte und Funktionen der Erweiterung zu verwalten, ohne dafür Template des Icons zu verwenden, ist die dedizierte Optionenseite[13]. 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.(REF)

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 1.1 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[14], 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 Server 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.

[13] https://developer.chrome.com/extensions/options

[14] https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/

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. Als Quelle für Teile der Recherche diente das Buch von Raymond Camden „Client-Side Data Storgae “

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[15], um Informationen zu speichern. 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[17] ist eine einheitlich Methode von mehreren Browser, Daten lokal zu speichern. 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[18] ist die Extension in der Lage eine domainspezifische Datenbank zu erstellen. 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[19]. So liegt der verfügbare Speicher zwischen 10MB und 50% des freien Festplattenspei-

[15] https://developer.chrome.com/extensions/cookies

[16] https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:de:HTML

[17] https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

[18] https://developer.mozilla.org/de/docs/IndexedDB

[19] https://developer.mozilla.org/de/docs/IndexedDB/Browser_storage_limits_and_eviction_criteria

chers. IndexedDB wird nicht noch komplett unterstützt und gilt als vergleichsweise komplexe API.

Ergänzend zu erwähnen ist die SQLite basierte Datenbank Web SQL. 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 PlayStore 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 PlayStore 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 Ap-

plikation 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 PlayStores 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.

/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.

/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 Punkte vgl pguard informationen an der richtigen Stelle zur Verfügung stehen.

/NF40/ Skalierbarkeit def 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 Ersten 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 2 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 Datenschutzinfor-

mationen 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(Abb. 3.1 Nr.1) in die Bereiche „Einkaufen“ und „Meine Apps“. Der Bereich „Einkaufen“ ist in die drei Reiter(Abb. 3.1 Nr.2) „Startseite“, „Top-Charts“ und „Neuerscheinungen“ gegliedert. Links neben diesen Reitern befindet sich eine Auswahl für einzelne Kategorien.

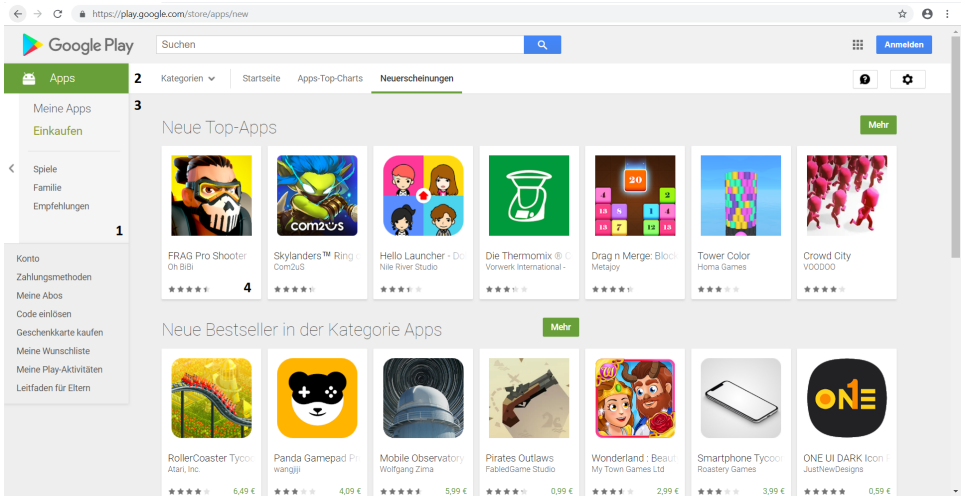

Abbildung 3.1: Kategorie Apps im Google Play Store (1: Apps-Menü; 2: Reiter; 3: Anzeigebereich der Apps; 4: Einzelne Kachel)

Alle Auswahlpunkte zeigen im Darstellungsbereich(Abb. 3.1 Nr.3) Apps in der gleichen Struktur an. Wie in Abbildung 3.1 zu erkennen, werden zu einer Thematik mehrere Kacheln(Abb. 3.1 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 3.2). 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.

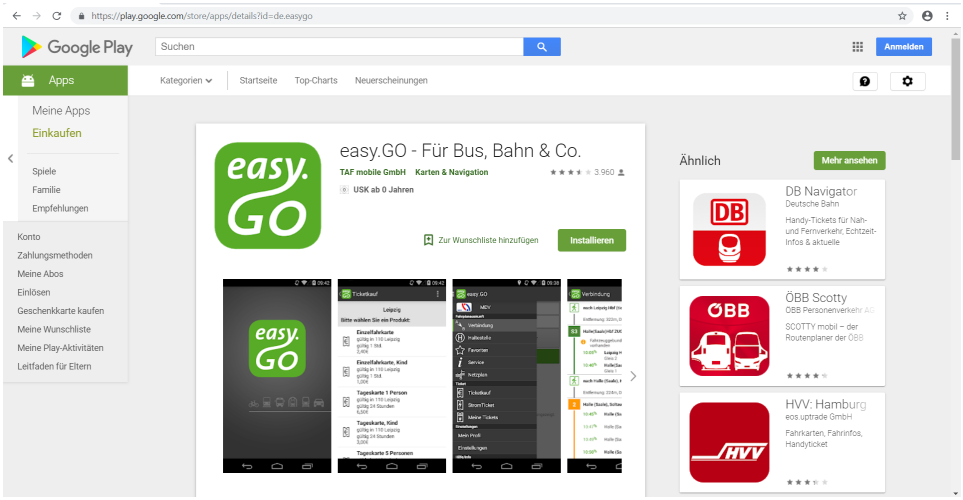

Abbildung 3.2: Detailansicht einer App

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 3.1 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.

- **Mittlere Kachel**: Als Raster unter dem Menü-Punkt „Meine Apps“ und als vertikale Reihe neben einer großen Kachel in der Detailansicht(Abb. 3.2) 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.

- **Große Kachel**: Jede App besitzt eine Detailansicht auf einer separaten Seite(Abb. 3.2). 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 Steckbrieform.

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

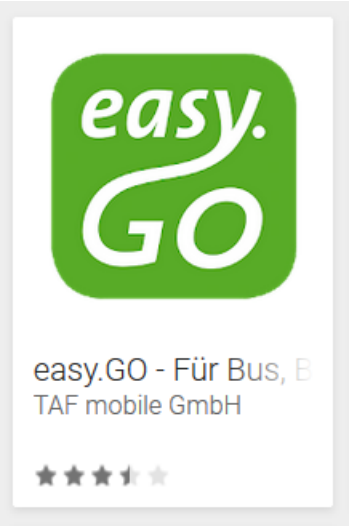

Abbildung 3.3: Kleine Kachel

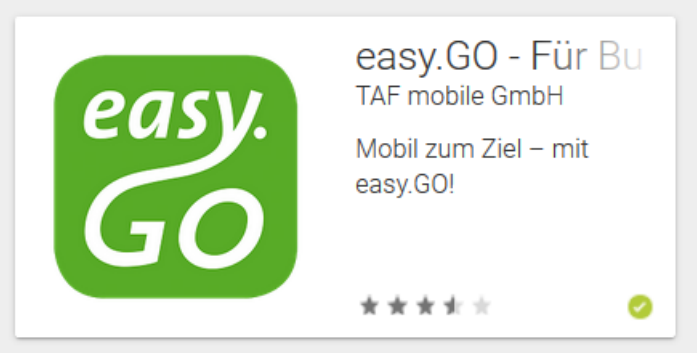

Abbildung 3.4: Mittlere Kachel

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.

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.

```
1 {
2   "name": "PGuard AppRating",
```

Abbildung 3.5: Große Kachel

```
  "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"],
      "css": ["lib/css/bootstrap.min.css", "lib/css/multiapp.css"]
    }
  ],
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  "web_accessible_resources":[
    "lib/data/*.json",
    "lib/templates/*"
```

```
24   ]
25 }
```

Listing 3.1: Aufbau der manifest.json

Das Manifest stellt die grundlegenden Zusammenhänge der Extension dar. Unter 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 **??**.
- **declarativeContent**: Bereitstellung von Events, wie Seitenaufruf oder -änderung 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.

  Der Ausdruck „*://play.google.com/store/apps*“ bedeutet, dass die Extension auf jeder Playstore-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 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.

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 der passende URL aufgerufen wurde. Dieser stimmt mit dem String aus der Manifest-Datei überein. Ist die Kondition erfüllt, aktiviert sich das Popup.

```
/**
 * Created by Alexander on 04.04.2018.
 */
chrome.runtime.onInstalled.addListener(function() {
    //removeRules, braucht declarativeContent permission, 1 = liste an regeln(undefined = keine
      Eingabe), 2= function nach loeschen
    chrome.declarativeContent.onPageChanged.removeRules(undefined, function() {
        //siehe removeRules, wird am ende von remove rules aufgerufen.
        chrome.declarativeContent.onPageChanged.addRules([{
            conditions: [new chrome.declarativeContent.PageStateMatcher({
                pageUrl: { schemes: ['https'], hostContains: 'play.google.com',
      pathContains:"/store/apps"}
            })
            ],
            actions: [new chrome.declarativeContent.ShowPageAction()]
        }]);
    });
});
```

Listing 3.2: background.js

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“, ob die aktuelle Seite eine Single-App-Page oder Mutli-App-Page ist und ermittelt sämtliche Kandidaten, welche für die Einbettung der Informationen in Frage kommen. Dabei liest der JQuery-Selektor alle Elemente mit dem entsprechenden Klassnamen aus.

Mit der Funktion „loadInfoPanels“ 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 3 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 App-Page(Single-App), 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.

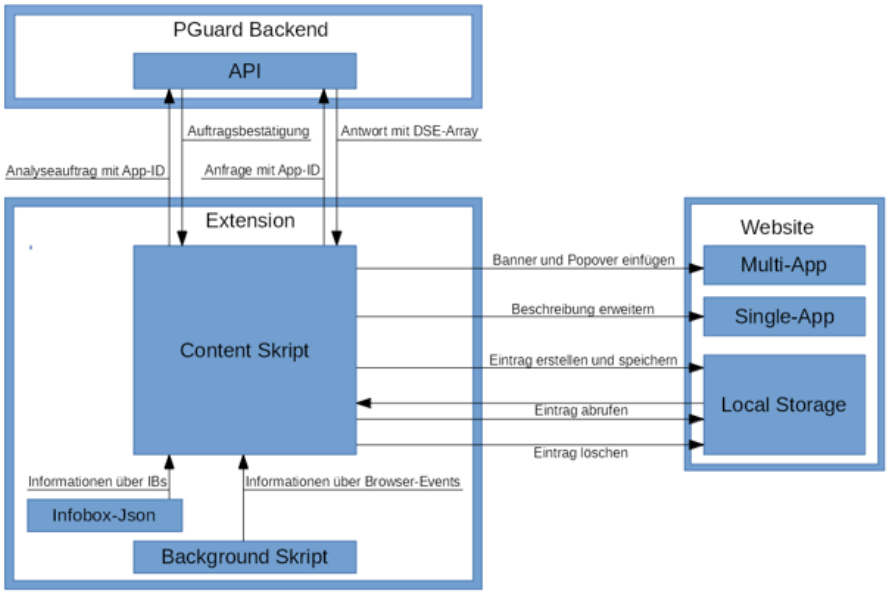

Abbildung 3.6: Aufbau und Interaktionen der Extension

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. 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.

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

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 **??** 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 großen Mehraufwand verbunden ist.

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 worden.

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.

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.

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 3.1 den Aufbau der Internetseite nach bestimmten Themen. Jedes Thema wird mit einer Reihe von Apps dargestellt. Je nach Bildschirmauflösung beinhaltet eine Reihe 7 bis 10 dieser Kacheln. Beim initialen Aufruf lädt die Seite 8 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 eine 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 Redundanz 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 und 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 3.7: Beispiel eines key-value-Datensatzes

Die gespeicherten Indizes der Infoboxen werden lokal von der Extension über die Datei „IB_texte.json “(Auszug in Abbildung 3.3) 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.


```
{
  "id": "19",
  "is_red_line": "false",
  "titel": "Die App verarbeitet Daten, die für die Funktion der App nicht erforderlich sind",
  "description": "",
  "pros": [
    {
      "first_layer": "Dies ermöglicht es, die App um Komfortfunktionen zu erweitern (z.B.
    Kalendereinträge, Kartendarstellung).",
      "second_layer": ""
    }
  ],
  "cons": [
    {
      "first_layer": "Es werden unnötige Daten erhoben.",
      "second_layer": ""
    },
    {
      "first_layer": "Ihre Daten können zu detaillierten Profilen zusammengeführt werden.",
      "second_layer": ""
    },
    {
```

```
22          "first_layer": "Ihre Daten können missbraucht werden.",
23          "second_layer": ""
24        }
25      ],
26      "recommendations": "",
27      "actions": [
28      ]
29    },
```

Listing 3.3: 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. 139Byte. 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 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[3]. Durch diese Obsoleszenz erfüllt auch WebSQL das Kriterium

[1] Gemessen durch Analyse aller App-ID-Einträge während der Evaluation
[2] https://arty.name/localstorage.html
[3] https://www.w3.org/TR/webdatabase/

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 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 aufeinanderfolgende 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 einer bestimmten Kategorie 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:

3.2.4 Ergebnisse

Storage: none Ladezeit: 1435ms Start der Extension-Funktionsaufrufe: 944ms Dauer: 491ms Anzahl der Anfragen an das Backend: 0

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 73.0.3683.75 64-Bit
Internetverbindung	Download: 100MBit/s , Upload: 6Mbit/s

Tabelle 3.1: Spezifikationen der Testumgebung

Storage: Local Storage Ladezeit: 1711ms Start der Extension-Funktionsaufrufe: 892ms Dauer: 819ms Anzahl der Anfragen an das Backend: 125

Storage: none Ladezeit: 1761ms Start der Extension-Funktionsaufrufe: 883ms Dauer: 878ms Anzahl der Anfragen an das Backend: 131

IndexedDB: API in allen modernen Browser zur Speicherung von Daten und Dateien in einer object-orientierten Datenbank. synchron und asynchron möglich. Funktioniert nach key, value prinzip Alle Datentypen von JavaScript werden unterstützt. Kann indexiert werden um Suchen effizient zu machen. Verwendet Prinzip von Transaktionen Anfragen mit Rückgabewerten als Basis aller Operationen Verfolgt den NoSQL-Ansatz

Warum nicht IndexedDB?

Vorteile von IndexedDB: Abspeicherung von großen strukturierten Datenmengen. Nachteile: hoher Aufwand bei Implementierung.Overhead lohnt nicht bei kleinen Datenmengen. Transaktionen blockieren bei Fehlern eventuell den Datenabruf bzw. die Aktualisierung

Storage API von Chrome ausreichend Speicher und geringer aufwand bei der Implementierung. Lediglich Strings benötigt. Indices bei gewählen value-Struktur nicht notwendig.

Vorteil von Session Storage: Speicherpflege nicht notwendig, da 5MB groß genug für Anzahl(?) an App-Informationen während einer Session im PlayStore. Informationen immer auf Stand der Quelle Nachteil: Bei erstmaligen Öffnen des Stores in neuer Browsersession werden viele Anfragen losgeschickt für Apps die bereits in der letzten Session schon angefragt wurden. Bei Serverausfällen fehlen die Informationen Lediglich in einer Session mehrfach aufgerufene Apps ersparen

erneute Anfragen. =¿ Speicherpflege fällt weg, dafür kaum Mehrwert bei Anfragen.

Vorteil von Lokal Storage: Apps werden einmal abgefragt und sind anschließend abgespeichert. Fällt der Server aus können die lokalen Informationen genutzt werden. Daten auch aus letzter Session bleiben vorhanden. Neue Anfragen werden nur dann geschickt wenn aktuelle Daten über 3 Tage alt sind. Nachteile: Speicherpflege notwendig. Dadurch wird die Information länger (Counter und Tag). Zusätzliche Rechenzeit für das Löschen von alten Informationen notwendig. Dadurch wird sichergestellt dass die 5MB nicht überschritten werden und somit Informationen ungewollt verloren gehen. Für Informationen mit hohem Counter muss regelmäßig überprüft werden, ob die Information noch aktuell ist, weil diese in der Regel lange im Speicher verweilt. =¿ Hohe Einsparung bei Anfragen an den Server möglich. Dafür müssen zusätzliche Operationen zur Speicherpflege und Prüfung der Informationen ausgeführt werden.

3.2.5 Diskussion

Zeitspanne verlängern

Kapitel 4

Abschließende Diskussion

4.1 Konklusion

4.2 Fortsetzung der Forschung

Kapitel 5

Appendix

5.1 Derivations

5.1.1 Example Matlab Code

Add a sourcefile directly into LaTeX

```
% simple Kalman Filter example:
% state "x" consists of position and velocity
% system model "F" is a cinematic model of constant velocity
% only the position is measured

clear     % clear all matlab variables

%%% declare matlab variables and asign default (randomly chosen) value
% simulation specifications
T = 1;              % make a measurement every T steps. also called \Delta t
                    % i.e. every 1, 2, 3, ... seconds
real_x = [0; 10];   % "real world" state, only needed in simulation context
                    % also called ground truth. start: position=0, velocity=10

% model specifications
model_F = [1, T ;   % the model we have about the real world
           0, 1];   % here: cinematic model of constant velocity
q = 9;              % controls the amount of process noise. is usually unknown
model_Q = [T^4/4, T^3/2;       % process noise
           T^3/2, T^2 ]*q;     % arises from the cinematic model

% estimations specifications
esti_x = [0; 10];   % estimated state: position and velocity
esti_P = [1, 0 ;    % estimated covariance of esti_x. refelcts
          0, 2];    % the uncertainty about the estimated state esti_x

esti_z = 0;      % estimated measured value. here: just depicting position-entry
                 % of esti_x since we are only iterested in the position. Or
                 % maybe it is only possible to measure position, but not velocity
esti_S = [0,0 ; % estimated covariance of esti_z. Will consist of process noise
          0,0]; % with added measurement noise

H = [1, 0;       % observation matrix. we only measure position values
     0, 0];      % this row could be left out, but then also modify R to 1x1
R = [1, 0 ;      % measurement noise. is usually unknown. reflects the
     0, 1];      % inaccuracy of the sensors
K = [0, 0];      % Kalman gain vector


%%% Initialization
esti_x = [0; 10];
esti_P = [1, 0 ;
          0, 2];

for step = 1:1000          % simulate for 1000 steps (simulate continuous time)
  if mod(step, T) == 0     % if it is time to take a new measurement
    % update the "real data". For simplicity: take the model F. But could be any
    % other function, possibly non-linear.
    % mvnrnd = multi variate normal random numbers
    real_x = model_F * real_x + transpose(mvnrnd([0,0], model_Q));

    %%% Step 1: Prediction Step
    esti_x = model_F * esti_x;  % estimate the new state according to the
                                % system model since we do not have any
                                % control inputs, this term is left out
    esti_P = model_F * esti_P * transpose(model_F) + model_Q;  % update the
                                % covariance of estimated state esti_x
    esti_z = H * esti_x;        % depic position value from estimated state
    esti_S = H * esti_P * transpose(H) + R;  % estimation of the covariance of
                                % the estimated measured value. inherits model
                                % noise and measurement noise

    %%% make a measurement z
    z = H * real_x + transpose(mvnrnd([0,0], R)); % make a noisy measurement

    % Step 2: Innovation Step
    K = esti_P * transpose(H) * esti_S^-1;  % calculate Kalman gain vector by
                                            % comparing model and measurement
                                            % noise
    esti_x = esti_x + K * (z - esti_z);     % update the estimated state by an
                                            % weighted sum of the measurement
                                            % and the model-estimation
    esti_P = esti_P - K * esti_S * transpose(K);  % update covariance of
                                                  % estimated state
  end
end
```

Listing 5.1: Simple example of a Kalman Filter in Matlab

Acknowledgement

First of all, I would like to express my gratitude to … for the aspiring guidance, useful comments and invaluably support throughout the whole process of this Master Thesis. Furthermore, I would like to thanks …, … and the other members of the research team for helpful discussions and constructive criticism. In addition, I would like to thank my University supervisor … for all the helpful remarks, advises and discussions.

Also, I would like to thank my parents and … who have supported me throughout the entire process by keeping me harmonious and motivated.

Last but not least, I like to thank … for funding my research and providing me with the facilities being required.

Abbildungsverzeichnis

Literaturverzeichnis

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Forname Surname, Place, 17. März 2019