Datavetenskap

Pontus Anttila

Mot effektiv identifiering och insamling av brutna länkar med hjälp av en spindel

Examensarbete, C-nivå

2018:06

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Pontus Anttila

Godkänd, Datum

Handledare: Jenni Reuben Shanthamoorthy

Examinator: Johan Garcia

Sammanfattning

I dagsläget har uppdragsgivaren, Mio AB, ingen automatiserad metod för att samla in brutna länkar på deras hemsida, utan detta sker manuellt eller inte alls.

Detta projekt har resulterat i en praktisk produkt som idag kan appliceras på uppdragsgivarens hemsida. Produktens mål är att automatisera arbetet med att hitta och samla in brutna länkar på hemsidan. Genom att på ett effektivt sätt samla in alla eventuellt brutna länkar, och placera dem i en separat lista så kan en administratör enkelt exportera listan och sedan åtgärda de brutna länkar som hittats.

Uppdragsgivaren kommer att ha nytta av denna produkt då en hemsida utan brutna länkar höjer hemsidans kvalité, samtidigt som den ger besökare en bättre upplevelse.

Towards effective identification and collection of broken links using a web crawler

Abstract

Today, the customer, Mio AB, has no automated method for finding and collecting broken links on their website. This is done manually or not at all.

This project has resulted in a practical product, that can be applied to the customer's website. The aim of the product is to ease the work when collecting and maintaining broken links on the website. This will be achieved by gathering all broken links effectively, and place them in a separate list that at will can be exported by an administrator who will then fix these broken links.

The quality of the customer's website will be higher, as all broken links will be easier to find and remove. This will ultimately give visitors a better experience.

Tack

Jag vill tacka alla inblandade på MIO:s servicekontor i Tibro, som bidragit med en trevlig arbetsplats under projektets gång. Jag vill också tacka Jonas Persson på MIO, som gjort detta projekt möjligt. Ett stort tack till Håkan Tropp som agerat handledare på plats, och som tillsammans med Jonas Ahlsson bidragit med teknisk kunskap.

Även ett stort tack till Jenni Reuben Shanthamoorthy som varit handledare för uppsatsen, och som också bidragit med rådgivning angående produktutvecklingen.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Introduktion

MIO är idag en av Sveriges större möbel- och detaljhandelskedjor. De växer snabbt, och nådde under 2017/2018 över 300 miljoner kronor i försäljning via deras hemsida, och uppemot 25 miljoner besökare [1]. Med så många besökare, och såna stora summor pengar, är det önskvärt att varje besökare kan navigera hemsidan utan bekymmer. MIO:s hemsida är stor, och består av flera tusentals sidor. Vissa sidor är autogenererade, framförallt produktsidor. En hemsida med syftet att sälja en produkt till en besökande kund vill givetvis att kunden ska kunna navigera hemsidan felfritt, utan att stöta på några sidor som ger felmeddelanden [2].

Ett exempel på ett sådant felmeddelande är ”Error 404, Page not Found” [3], vilket är bland de vanligare felmeddelandena. Då en besökare klickar på en länk som refererar till en sida som inte längre existerar, så kommer detta meddelande att visas och vi kan konstatera att den länk som klickats är bruten. En studie visade att ungefär 44% av alla besökare som haft en dålig upplevelse på en hemsida kommer att berätta för sina vänner om det [4]. Detta kan i sin tur påverka försäljningssiffror, om potentiella kunder istället väljer att besöka en konkurrent.

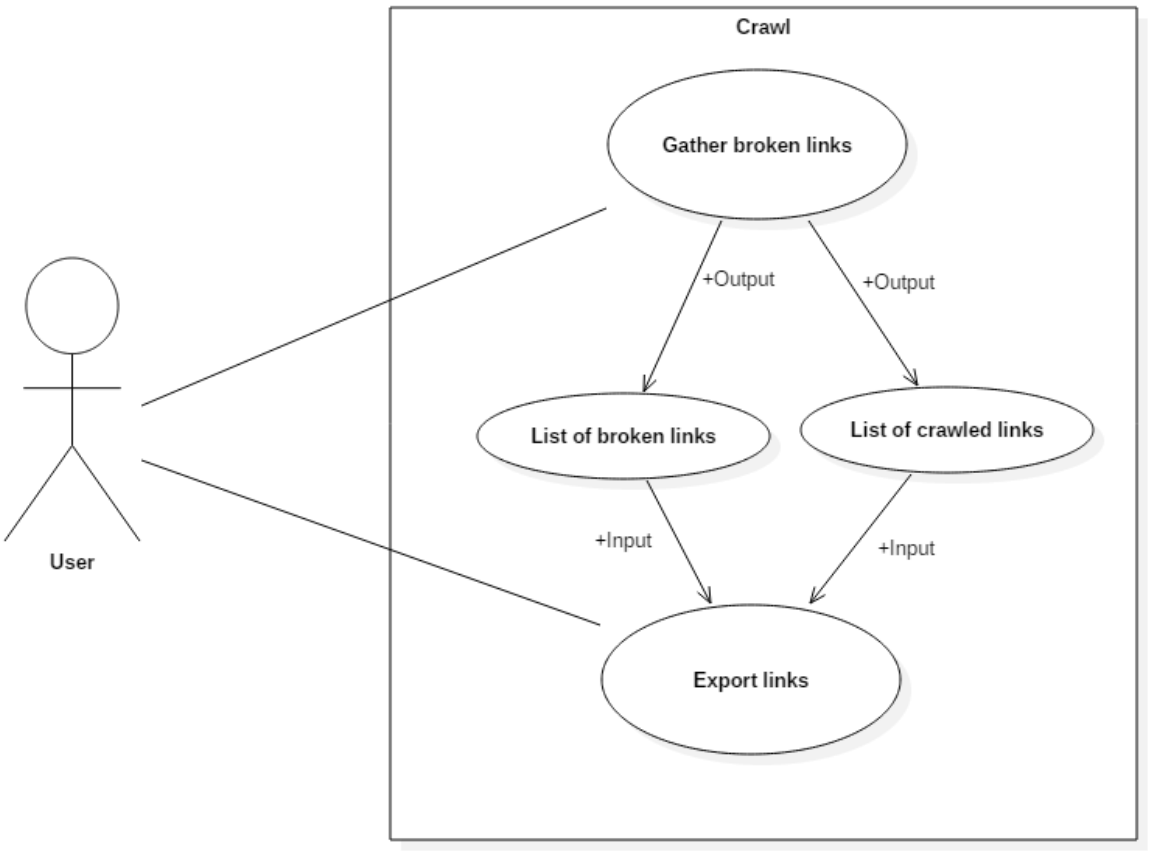
Att inte regelbundet hitta och åtgärda brutna länkar kan i längden leda till att antalet brutna länkar på hemsidan blir alltför många. Att som besökare stöta på en bruten länk efter en annan leder lätt till frustration, och kan till och med leda till att besökaren inte köper något på hemsidan. Om problemet blir allt för stort kan det även påverka företagets rykte, som på sikt kan tappa potentiella kunder.

Brutna länkar kan även leda till en lägre ranking på hos sökmotorer. Sökmotorer använder sig av spindlar för att söka igenom hemsidor och indexera dem. Då en sådan spindel träffar på en bruten länk så fortsätter spindeln vidare till nästa sida, och många länkar som kunde indexerats missas.

Målet med projektet, och utvecklingen är att underlätta och automatisera arbetet med brutna länkar. Då denna uppgift kan ta upp väldigt mycket tid, om den utförs manuellt, så är det en uppgift som på grund av detta ofta bortprioriteras. Eftersom en hemsida består av en mängd

webbsidor, som framförallt navigeras mellan via länkar, så är det viktigt att så många som möjligt av de sidor som går att nå via länkar faktiskt existerar.

1.1 Uppdragsbeskrivning



Figur 1.1: Beskrivande bild av projektet

Arbetet är tänkt att mynna ut i en praktisk tjänst, en så kallad spindel, som vid exekvering söker igenom hemsidan efter trasiga länkar. De trasiga länkar som hittas kommer att sparas undan i en textfil, som sedan kan exporteras av användaren vid behov för att på sikt åtgärdas.

I figur 1.1 visas en övergriplig bild på hur systemet är tänkt att användas. Programmets användare kan vara en systemadministratör, som har för avsikt att hitta och åtgärda brutna länkar. För att kunna exportera brutna länkar så måste de först samlas in.

I kommande kapitel beskrivs en rad regler som kan följas vid implementering av spindeln. Några av dessa har jag nyttjat, för att spindeln ska arbeta på ett så effektivt sätt som möjligt

och för att brutna länkar ska hittas och samlas in så snabbt det går. Desto tidigare en bruten länk är funnen, desto tidigare kan den åtgärdas av en administratör.

Användaren kommer att interagera med produkten genom ett grafiskt användargränssnitt (GUI), som är tänkt att öka användarvänligheten. Ett GUI underlättar för användaren som får mer insikt i vad som sker vid exekvering, och snabbare kan starta insamlingen av länkar.

Genom att använda min produkt kommer användaren att klara av uppgiften snabbare och mer effektivt än tidigare. En välskött hemsida kommer i längden att generera mer trafik åt MIO, och locka potentiella kunder. På så vis kommer de att sälja mer produkter och tjäna mer pengar, vilket också är syftet med hemsidan.

Utöver det ovan beskrivna användningsområdet, så kan produkten användas till mycket annat. Funktionalitet kommer att kunna läggas till i efterhand, för att utföra vilken uppgift användaren än vill.

För att fullfölja projektet och nå upp till de mål som satts så kommer jag att med följande:

- En färdig produkt som direkt kan appliceras på hemsidan.
- En produktbeskrivning och projektspecifikation innehållande användningsfalldiagram och aktivitetsdiagram för systemets användningsområden.
- En utvärdering av projektet, innehållande en analysering av slutprodukten där resultatet vägts mot de riktlinjer och regler som beskrivs i kommande kapitel.
- Förslag på vidareutveckling och framtida användningsområden.

1.2 Disposition

1.2.1 Kapitel 2 - Bakgrund

I kapitel 2 beskrivs olika användningsområden där en spindel kan appliceras. I kapitlet motiveras också projektet och en rad riktlinjer och implikationer vid implementeringen av en spindel introduceras.

1.2.2 Kapitel 3 – Projektdesign

I detta kapitel ges en introduktion på projektet, och projektets design presenteras. Här diskuteras även projektets syfte och mål i detalj.

1.2.3 Kapitel 4 – Implementation

I kapitel 4 avhandlas systemets alla olika komponenter och hur de arbetar tillsammans. Här presenteras också programmets grafiska användargränssnitt.

1.2.4 Kapitel 5 – Utvärdering

I detta kapitel utvärderas projektet och slutprodukten analyseras. Resultatet av utvärdering ska styrka de mål som satts för projektet.

1.2.5 Kapitel 6 – Slutsats

I kapitel 6 ges en summering av projektet. Förslag på vidareutveckling och systemets avgränsningar är punkter som diskuteras.

2 Bakgrund

Syftet med detta kapitel är att introducera vad exakt en ”spindel” är, och hur de fungerar. En rad riktlinjer som kan följas vid implementering av en spindel och hur dessa respekteras i mitt projekt, samt moderna lösningar.

2.1 Introduktion till web crawling

En spindel, även kallad web crawler, robot eller enbart crawler, besöker och laddar systematiskt ned hemsidor [5]. Den gör detta genom att följa länkstrukturen hos en hemsida.

En spindel kan användas inom många olika områden, och i många olika syften. Det vanligaste användningsområdet är insamling av länkar och indexering av hemsidor, främst nyttjat i sökmotorer vars huvuduppgift är att indexera så många hemsidor på internet som möjligt. Internet är uppbyggt och består av miljontals hemsidor som drivs av miljontals oberoende aktörer. För att nå ut till alla potentiella besökare, eller till den som söker en viss sida, så används sökmotorer. Dessa sökmotorer förlitar sig till stor del på det arbete som utförs av spindlar.

Några andra användningsområden är:

- Arkivering av hemsidor, med målet att spara en kopia av varje webbsida.
- Datainsamling för kommersiella syften.
- Datainsamling för analytiska syften.

2.1.1 Robot Exclusion Standard och User Agents

Efter att allt fler insåg att en spindel vid oetiskt bruk kan användas för att ladda ned känsliga data, eller neka vanliga användare på hemsida den bandbredden avsedd för dem, så togs *Robot Exclusion Standard* fram [6]. Denna standard, eller protokoll, introducerades i 1994, och har sedan dess setts som standarden för administratörer som vill hindra spindlar från att besöka vissa delar av deras hemsida.

Syftet med protokollet är att berätta för besökande spindlar vilka sidor och filer på hemsidan den tillåts ladda ned. Protokollet implementeras genom en textfil, skriven enligt en given

standard som spindlar enkelt ska kunna tolka. I textfilen specificeras vilka subdomäner på hemsidan som spindeln inte får ladda ned och indexera. Textfilen kan alltid kommas åt genom att lägga till "/robots.txt" i slutet av en hemsidas URL, till exempel www.mio.se/robots.txt. En spindel vars avsikt är oetiskt, kan fortfarande välja att helt bortse från hemsidans textfil. Den underlättar dock för etiskt lagda spindlar, till exempel sökmotorer som slipper ladda ned sidor och filer som inte är relevanta. Innehållet i textfilen anger vilka "User Agents" som har tillåtelse att ladda ned vilka sidor.

En User Agent representerar det program, eller den klient som kommunicerar med en server. Alla webbläsare definieras av en User Agent, precis som alla spindlar. När en besökare ansluter till en hemsida så identifierar sig webbläsaren genom att skicka med information om sig själv i HTTP-anropet till servern.

2.1.2 Site map

En "site map" är i grunden endast en lista av sidor som en hemsida består av [7]. Alla de länkar som finns på en hemsida bildar tillsammans hemsidans "site map". En "site map" kan ofta representeras grafiskt, för att se hur alla sidor är kopplade till varandra – vilka sidor som länkar till varandra.

2.2 Spindlar på internet

Varje hemsida på internet nås via en URL, en adress som skrivs in i webbläsarens adressfält [8]. När en besökare klickar på en länk hanteras denna URL automatiskt, och tar besökaren till rätt sida. En länk, eller hyperlänk [9], refererar alltså till data på en annan, eller på samma sida.

Spindeln hämtar en hemsidas HTML-kod, som den sedan tolkar och extraherar önskade data ur. Denna data kan vara vilken data (information) användaren än är intresserad av, till exempel produktinformation eller pris. Utöver insamling av data, så samlas alla länkar som finns på den besökta sidan in. Dessa insamlade länkar kommer även de att besökas så småningom, där processen upprepas.

En spindel matas kontinuerligt med insamlade URL:er, som den själv hittar genom att extrahera hyperlänkar i en nedladdad sidas HTML-kod. När spindeln startas ges en URL som

input av användaren, som indikerar var spindeln ska börja. Denna URL kan vara en hemsidas förstasida eller en subdomän. Spindeln kommer utifrån denna jobba sig allt djupare i hemsidans index.

Varje hyperlänk som spindeln finner placeras i slutet på en kö. Så fort en länk i denna kö är hanterad så kommer den att tas bort ur kön, och nästa URL på tur kommer att matas till spindeln, som upprepar processen på denna sida.

Förutom indexering av hemsidor kan spindeln hitta så kallade brutna länkar. Brutna länkar är hyperlänkar som inte leder någonstans, utan refererar till en sida som inte längre finns. Denna sida kan ha existerat vid ett tidigare tillfälle, men har tagits bort av systemadministratören som då glömt radera alla hyperlänkar som refererar till den nu borttagna sidan.

2.3 Implikationer

Vid utveckling av en spindel bör utvecklaren ta hänsyn till ett flertal riktlinjer, som avser både etiska såväl som tekniska och sociala aspekter. En spindel som är dåligt skriven eller som utvecklats utan att ta hänsyn till dessa riktlinjer, kan i slutändan orsaka problem på de hemsidor och servrar som den besöker. De två största hoten en spindel utgör mot hemsidor idag är överbelastning på deras servrar, känt som ”Denial of Service” (DoS) [10], samt potentiella integritets- och copyrightöverträdelser.

2.3.1 Denial of Service

En spindel som är för effektiv, och skickar många förfrågningar i sekund till en hemsida kan leda till överbelastning hos servern [11]. Teknikens utveckling har dock gjort att dagens servrar klarar av mer trafik än tidigare, samtidigt som moderna spindlar jobbar avsevärt mycket snabbare än deras föregångare. Så trots att moderna servrar kan hantera mer trafik, så jobbar samtidigt dagens spindlar snabbare och effektivare vilket resulterar i att problemet med överbelastning till viss del kvarstår. Mindre, självständigt drivna hemsidor drabbas hårdast av detta.

Det är idag vanligt att implementera en spindel på ett sådant sätt att den kan jobba på flera olika trådar samtidigt, parallellt. En sådan parallellt arbetande spindel utgör ett ännu större hot mot hemsidans prestanda, då den kan ladda ned flera sidor från en server samtidigt. När en

server blir överbelastad och tvingas hantera för många förfrågningar från spindeln, på mycket kort tid, så kommer hemsidans vanliga besökare tilldelas en mycket lägre bandbredd, vilket medför att deras navigering på hemsidan upplevs långsammare. Teoretiskt sätt så kan en klumpigt implementerad spindel fungera exakt som en så kallad ”DoS-attack” [10], där målet är att stänga ner den attackerade hemsidan genom att medvetet skicka fler förfrågningar till servern än vad den kan hantera. Om servern är för upptagen med att hantera förfrågningar från spindeln, så kan hemsidans primära syfte och funktion undermineras.

2.3.2 Integritet

Nästintill allt innehåll som publiceras på internet är i grunden offentliga data, tillgänglig för alla. En spindel kan vid oetiskt användande utnyttja detta, och användas för insamling och lagring av känsliga persondata. Att använda spindeln för att krypa hemsidor och extrahera alla mailadresser den hittar, är ett exempel på ett oetiskt användningsområde. Dessa mailadresser kan sedan läggas till i olika spam-listor, där ingen av mottagarna själva valt att ingå.

Vid hantering av personliga data och information krävs försiktighet, då ett sådant oetiskt användande kan leda till integritetskränkningar. Integritet är ett stort diskussionsområde både på och utanför internet, och bör alltid tas hänsyn till – även vid implementering av en spindel.

2.3.3 Copyright

Kränkningar av copyrightskyddat material är ett problem för många sökmotorer då deras uppgift är att hitta hemsidor och spara en kopia av dem. En del data på dessa hemsidor kan vara upphovsrättsskyddat, och att spara kopior av denna data är, i grunden, en kränkning av upphovsrättslagen.

Många hemsidor innehåller sidor med data som de inte vill att sökmotorer eller andra spindlar ska krypa och indexera. Detta ledde till skapandet av det idag allmänt accepterade och använda protokollet ”Robot Exclusion Standard” (robots.txt.) [6].

2.3.4 Kostnad för bandbredd

Många hemsidor som drivs av självständiga, oberoende ägare använder sig ofta av hyrda servrar, som i vissa fall enbart tillåter en begränsad mängd bandbredd. Om så är fallet, betyder det att trafik som orsakas av besökare kommer att använda upp den bandbredd som hemsidan fått allokerad. Så länge det är faktiska besökare som använder denna bandbredd, kan ägaren anse att kostnaden för att driva hemsidan är värd det. Om den allokerade bandbredden istället

används av en besökande spindel kan hemsidans ägare istället tycka att denna utgift är slöseri av pengar, då trafiken inte genererades av en faktisk besökare. En genomsnittlig besökare klickar sällan igenom en hel hemsida så som en spindel gör, vilket medför att en spindel kommer att generera avsevärt mycket mer trafik på ett besök jämfört med en vanlig besökare.

2.3.5 Dåligt implementerade spindlar

En spindel bör implementeras på ett sådant sätt att den inte påverkar prestandan på den sida den besöker. Spindelns prestanda ska inte heller den påverkas av det innehåll den stöter på, till exempel en så kallad ”spider trap” [12].

En ”spider trap”, eller spindelfälla är precis som namnet antyder en sorts fälla som fångar en besökande spindel i en oändlig loop av duplicerade sidor. En sådan fälla kan skapas både avsiktligt och oavsiktligt. Vid avsiktlig användning kan intentionen vara att fånga en spindel utsänd av en sökmotor, som då indexerar samma sidor flera gånger med målet att hemsidan ska hamna högt upp bland sökresultaten.

2.4 Implementationsriktlinjer

Vid utveckling av en spindel finns det även en del tekniska aspekter som måste tas hänsyn till. Det finns idag många utmaningar som en modern spindel måste kunna hantera för att anses vara effektiv.

Internet är ständigt växande, och under konstant förändring. Data ändras och utvecklas, samtidigt som nya tekniker introduceras. Varje sekund läggs nya sidor till det redan gigantiska nätet av hemsidor, medans existerande sidor ändras eller tas bort. Eftersom internet är så pass stort, så kan en spindel endast ladda ned en liten del av allt innehåll på en gång. Detta medför att spindlar, framförallt de som används av stora sökmotorer, måste prioritera vilka sidor som laddas ned, och när. Liknelser har gjorts mellan att krypa internet och att observera stjärnor [11]. Eftersom ljuset måste färdas från stjärnan till oss på jorden, för att vi ska se den, så ser vi inte stjärnan i realtid utan snarare hur den såg ut när stjärnans ljus påbörjade sin resa mot jorden. Samma logik kan appliceras på en spindel som indexerar hemsidor på internet. Eftersom internet är under ständig förändring, så kan ingen sökmotor ge en exakt representation av hela internet vid en given tidpunkt.

Alla de tidigare nämnda implikationerna kan undvikas om en rad riktlinjer följs vid designen och implementeringen av en spindel [11]. Dessa riktlinjer är:

- Selection Policy
- Re-visitation Policy
- Politeness Policy
- Parallelization Policy

2.4.1 Selection policy

Oavsett hur snabbt en spindel jobbar med att ladda ned och indexera sidor, så kan den aldrig indexera internet i sin helhet. Det finns helt enkelt för många hemsidor och webbsidor, som dessutom är under konstant förändring. På grund av detta krävs det att en spindel endast laddar ned och indexerar sidor som anses relevanta av användaren, och inte en slumpmässig samling av sidor. Det måste därför finnas ett sätt att prioritera vilka sidor som ska laddas ned och vilka länkar som ska följas.

2.4.2 Re-visitation policy

Som tidigare nämnt så ändras hemsidor ständigt. För att alltid ha ett så uppdaterat index som möjligt så måste sökmotorer återbesöka redan indexerade hemsidor. Dessa återbesök kan göras på indexerade sidor i den ordning de senast besöktes, eller genom att räkna ut med vilken frekvens dessa ändringar sker (hur ofta de uppdateras) och återbesöka de sidorna med högre uppdateringsfrekvens oftare än de sidorna med en låg sådan.

2.4.3 Politeness policy

De sociala implikationerna tidigare diskuterade (Denial of Service, slösaktig användning utav en servers bandbredd och resurser samt dåligt implementerade spindlar) faller under denna policy. För att ta hänsyn till dessa så krävs det att spindeln implementeras med ”Robot Exclusion Standard” i åtanke. Det enda sättet en hemsida kan kommunicera med en besökande spindel, är genom att ge den besökande spindeln direktiv om vad den får ladda ned och indexera på hemsidan. Det är därför viktigt att en spindel, främst sökmotorer, tar hänsyn till hemsidans ”robots.txt” [6].

Många hemsidor som använder sig av en ”spider trap” berättar i ”robots.txt” vilken sida denna fälla finns på [12]. En spindel som tagit hänsyn till denna policy kommer då att undvika fällan, medans andra spindlar fastnar i den.

2.4.4 Parallelization

I ett tidigare stycke diskuterades en modern spindels behov av att kunna arbeta snabbt, genom att ladda ned och hantera flera sidor samtidigt - parallellt. Ett av de vanligare sätten att uppnå detta, är genom att implementera spindeln så att multipla processer jobbar samtidigt. Varje process hanterar en unik sida, som hämtas från en gemensam kö. För att undvika att samma sida hanteras av flera olika processer samtidigt måste implementation ske på ett sådant vis att den URL som hanteras av en process, är osynlig för resterande processer. Med andra ord måste den URL som givits en process omedelbart raderas från den gemensamma kön innan nästa process tilldelas en URL. På så sätt undviker spindeln att arbeta med samma sida över flera processer samtidigt, vilket hade varit slöseri på resurser och prestanda.

2.5 Existerande lösningar

År 1994 togs en sökmotor som tillhandahöll “full text search” fram. Detta var den första sökmotor att erbjuda detta, vilket idag ses som standarden i de allra flesta sökmotorer. Denna sökmotor gick under namnet *WebCrawler* [13]. Under senare år har *WebCrawler* gått från att vara en fristående sökmotor, till att bli en så kallad “metasearch engine”. En sådan sökmotor har inte någon egen databas av indexerade sidor, utan presenterar resultatet på användarens sökning genom att kombinera sökresultat tagna från andra populära sökmotorer. Spindlar som användes av tidigare sökmotorer jobbade betydligt långsammare än den spindel som *WebCrawler* använde sig utav, och denna spindel sägs vara den första kapabel att hantera multipla sidor åt gången. Att hantera flera sidor simultant ökade hastigheten med vilken sidor laddades ned och indexerades, och introducerade konceptet ”Parallel Crawlers”. Idag är parallellisering något de allra flesta moderna spindlar implementeras med, för att maximera spindelns prestanda och nedladdningsfrekvens.

Den idag världskända, och globalt mest nyttjade sökmotorn *Google* skapades några år senare, år 1998 [14]. Det initiala syftet med skapandet av *Google* var att utveckla en storskalig spindel [14], kapabel att krypa och indexera hemsidor på en mycket större skala än någon tidigare spindel klarat av. Internet växer och förändras ständigt, detta medför att spindlar måste kunna anpassa sig. Från att de allra första spindlarna släpptes lös på internet, fram till skapandet av *Google*, så växte internet snabbt. *World Wide Web Worm* [14], som var en av de första sökmotorerna att krypa och indexera hemsidor, påstod sig år 1994 ha indexerat totalt 110 000 webbsidor [14]. *WebCrawler* påstod sig år 1997 ha indexerat uppemot 100 miljoner

webbsidor [14], vilket visar en markant ökning under bara tre år. Idag är *Google* både den globalt mest besökta hemsidan, och den mest använda sökmotorn, enligt *Alexa* [15]. De har idag hundratals miljarder webbsidor indexerade, alla funna och nedladdade av deras spindel *Googlebot* [16]. Utöver att ha bidragit med att introducera det vi idag anser vara en modern, storskalig spindel, så har *Google* också hjälpt till att upptäcka de sociala aspekter användandet av en spindel medför. Då spindeln klarade av att besöka många webbsidor på så kort tid medförde detta att många administratörer, och andra som ansvarade för hemsidorna som besöktes av spindeln, hörde av sig till *Google* med frågor angående upphovsrättsskyddat material och sekretess.

2.6 Översikt av det implementerade systemet

Som tidigare diskuterat så är den här aktuella spindelns huvuduppgift att förse användaren med en lista över alla brutna länkar på hemsidan. Utöver detta så kan användaren välja att exportera listan som innehåller alla hanterade länkar, för att på så vis få en lista över alla länkar som existerar på hemsidan. Att exportera denna lista skulle ge användaren en överblick över hemsidans innehåll, en ”site map”.

Den site map som användaren får kommer endast vara i textformat där alla länkar som finns på hemisdan är listade, till skillnad mot de flesta site maps som ger en grafisk bild av hur en hemsida är uppbyggd. Då detta inte är projektets huvudmål kommer inget mer arbete att läggas på detta.

Några av de policys och riktlinjer som diskuterades i ett föregående stycke har tagits hänsyn till vid implementering av min spindel. Den policy som jag ser som viktigast för att nå de mål som satts för projektet är ”Parallelization policy” [11]. Att implementera spindeln så att den jobbar parallellt innebär att den jobbar mer effektivt, och kan hantera flera olika sidor samtidigt. Då detta återfinns i de allra flesta moderna spindlar ser jag det som en självklarhet att även min produkt ska implementeras med denna funktion

En annan policy som jag tagit hänsyn till är ”Politeness policy” [11]. Min produkt kommer att ladda ned och följa en hemsidas ”robots.txt” [6]. Genom att respektera denna så kommer spindeln inte att besöka sidor inom hemsidan som en administratör inte vill att en spindel ska besöka.

Den tredje policyn som spindeln kommer att ta hänsyn till är ”Selection policy” [11], till viss del. Min spindel kommer att implementeras som en lokal spindel, då den endast kommer att följa länkar inom samma domän. Länkar som leder till andra hemsidor kommer att bortses från, vilket betyder att spindeln väljer vilka länkar som ska följas baserat på vilken domän de refererar till.

Spindeln kommer dock inte att ta hänsyn till den fjärde och sista policyn som beskrivits, ”Re-visitation” [11]. Då spindeln är designad för att appliceras på en hemsida i taget så kan användaren välja när detta ska ske. Ingen algoritm krävs då för att bestämma när spindeln ska återbesöka hemsidan.

2.7 Summering

En spindel är en typ av bot som systematiskt besöker och laddar ned hemsidor på internet. Detta görs genom att låta spindeln följa en hemsidas länkstruktur. En spindel används framförallt av sökmotorer, med syftet att ladda ned och indexera hela hemsidor.

En spindel som besöker en hemsida börjar med att ladda ned all HTML-kod, varpå den extraherar alla länkar den finner i den nedladdade koden. Dessa länkar placeras i en kö, som alla kommer att besökas av spindeln och på vilka denna process sedan upprepas.

De allra flesta spindlar implementeras idag enligt en rad riktlinjer och policys. Dessa riktlinjer existerar för att spindeln ska klara av en rad tekniska utmaningar, och sociala implikationer.

3 Projektdesign

I detta kapitel kommer en högnivådesign av produkten, samt olika användningsfall att presenteras. Målet med utvecklingen och produktens syfte kommer att diskuteras tillsammans med det problem som produkten är tänkt åtgärda.

3.1 Motivation

En stor hemsida, så som mio.se, består av flera tusentals sidor. En besökare navigerar hemsidan med hjälp av länkar. Alla dessa länkar måste underhållas, för att de alltid ska länka samman existerande sidor. När en besökare klickar på en länk, förväntas länken leda till en fungerande sida innehållande just det besökaren är ute efter. Då en användare försöker komma åt en sida som inte längre finns i hemsidans index, så presenteras besökaren med felmeddelandet “Error 404, Not Found”. En länk som refererar till en borttagen sida kan fortfarande existera efter att sidan är borttagen. En hemsida borde sträva efter att minimera antalet brutna länkar som genererar dessa felmeddelanden, då de kan påverka en besökares upplevelse av hemsidan.

3.2 Mål och syfte

I kapitel 1 introducerades projektets mål och syfte. Både ett huvudmål och ett andrahandsmål, som kommer av att huvudmålet uppnås. I kapitel 2 introducerades i sin tur en rad koncept och riktlinjer som underlättar implementeringen av en spindel. För att uppnå dessa mål så kommer jag att dra nytta av dessa riktlinjer.

Projektets huvudmål är att underlätta för, och att ge användaren ett sätt att automatisera jobbet med att hitta brutna länkar och att samla in dem. Genom att ta hänsyn till några av de tidigare diskuterade riktlinjerna så kommer spindeln att arbeta effektivt, och bara hantera relevanta länkar. Effektiviteten uppnås genom att implementera spindeln på ett sådant sätt att den jobbar parallellt, och kan på så vis hantera flera unika sidor åt gången. En parallellt arbetande spindel kommer att hantera samma mängd hemsidor snabbare än en spindel som bara jobbar över en tråd. Detta gör att användaren snabbare kan samla in eventuellt trasiga länkar, och då

snabbare åtgärda dessa. Desto längre de finns på hemsidan, desto större chans är det att en besökare hittar dem, vilket på sikt kan leda till att hemsidans rykte försämras.

För att ytterliga underlätta arbetet för användaren så kommer ett grafiskt användargränssnitt att utvecklas. Genom detta kommer all interaktion mellan programmet och användaren att ske. Att ge användaren tillgång till ett grafiskt gränssnitt kommer att öka användarvänligheten, vilket i sin tur ökar effektiviteten då ett enkelt gränssnitt låter användaren komma igång snabbare med insamling av brutna länkar, som sedan enkelt kan exporteras.

I samband med att projektets huvudmål uppfylls, så kommer också ett andrahandsmål att uppnås. Då hemsidan inte innehåller några brutna länkar kommer en besökare att uppleva sidan som mer komplett, och chansen för att besökaren återvänder ökar. Att en gång efter en annan stöta på brutna länkar på en hemsida kan frustrera en besökare, vars mål med besöket är att genomföra ett köp. Om kunden skulle ha otur och klicka på flertalet brutna länkar, så är chansen stor att denne vänder sig till en konkurrent. Detta skulle betyda hemsidan har förlorat en kund på grund av något som i teorin inte är svårt att åtgärda.

3.3 Introduktion till lösning

3.3.1 Statuskoder

”Hyper Text Transfer Protocol” (HTTP) har länge varit en standard världen över när det kommer till att distribuera och överföra data över internet [17]. Det är ett applikationslager-protokoll. Protokollet är ett så kallat “request-response”-protokoll, som bygger på att en klient initierar en anslutning och begär dataobjekt från servern, till exempel en fil på en webbsida. All kommunikation mellan klient och server sker via HTTP-meddelanden. Klienten skickar ett ”request message”, som servern besvarar med ett ”response message”.

När en klient vill ha åtkomst till en sida så skickas en förfrågan till den server som hemsidan finns på. Förfrågan skickas som ett HTTP-meddelande, som innehåller ett flertal rader med information som krävs för att servern ska kunna tolka förfrågan. Den viktigaste raden, för att kunna identifiera brutna länkar, är den som kallas “request line”. Denna raden innehåller information om vilken metod som använts vid förfrågan samt vilken URL klienten vill ha

åtkomst till. Vid mottagandet av förfrågan från klienten tolkas meddelandet innan ett svarsmeddelande skickas tillbaka. Svarsmeddelandet innehåller också det ett flertal informationsrader. En av dessa rader innehåller information om hemsidans status, alltså den statuskod som genereras vid anrop [2]. Då målet med spindeln är att hitta länkar som leder till icke existerande sidor, så är statuskoden som returneras vid ett anrop intressant. Statuskoden indikerar ifall den sida som klienten anropat är tillgänglig eller inte. Om sidan finns indexerad på hemsidan och klienten tillåts åtkomst till den, så returneras statuskoden "200 OK" som ger klienten klartecken att ladda ned sidan och dess innehåll.

Det finns emellertid en mängd olika statuskoder, som kan returneras av servern i responsmeddelandet efter att en klient begär åtkomst till en sida. Som förklarat i tidigare stycke, så kommer spindeln att dra nytta av dessa statuskoder för att avgöra om en länk är trasig eller inte. Alla statuskoder är grupperade i olika kategorier. Dessa kategorier innehåller statuskoder som representerar lyckade förfrågningar samt förfrågningar som resulterar i diverse felmeddelanden, antingen på serversidan eller klientsidan av anslutningen. De statuskoder som är av störst intresse för spindeln är de statuskoder som indikerar att klientens anrop resulterat i ett felmeddelande. Varje statuskod består av en siffra, följt av en förklarande text. De felmeddelanden som är av intresse för spindeln börjar på 400, och uppåt.

Om en klient anropar en sida som existerar, och som klienten har alla nödvändiga rättigheter till, så kommer servern returnera statuskoden "200 OK", som indikerar ett lyckat anrop. Denna statuskod betyder att servern har tagit emot klientens förfrågan (request message), läst och förstått förfrågan och tillslut accepterat den. Beroende på vilken metod som använts vid förfrågan, så kommer innehållet på den anropade sidan att returneras till klienten i responsmeddelandet.

Figur 3.1: En lyckad förfrågning

Om en klient istället begär åtkomst till en sida som för närvarande inte finns i hemsidans index, eller om klienten helt enkelt saknar de rättigheter som krävs för att komma åt sidan, så kommer en statuskod som representerar ett felmeddelande att returneras, istället för den sedvanliga “200 OK”. Statuskoder som indikerar att något i anropet gått fel på klientsidan av anslutningen skrivs “4xx”. Dessa felmeddelanden inkluderar “400 Bad Request”, “403 Forbidden” samt “404 Not Found”. Det sistnämnda är den statuskod som returneras då den anropade sidan för närvarande inte finns tillgänglig på hemsidan.

Figur 3.2: Felkod 404 på mio.se

Figur 3.3: En misslyckad förfrågning

En bruten länk kommer att definieras som en länk, som vid ett anrop resulterar i en statuskod som överstiger 400.

Klienten som initierar anslutningen till servern, och som begär åtkomst till hemsidan, identifieras via en ”User Agent”, eller användaragent. Då en spindel besöker en hemsida är det spindeln själv som är användaragenten, och namnet kan definieras av utvecklaren.

3.3.2 Spindelns modulära design

Genom att använda en spindel kan administratören automatisera jobbet med att samla in trasiga länkar var som helst på hemsidan. Spindeln kan jobba i bakgrunden, medan

administratören utför andra uppgifter, för att sedan enkelt exportera en lista med de trasiga länkar som spindeln funnit under tiden. Om spindeln tillåts exekvera under tillräckligt lång tid så kommer alla länkar på hela hemsidan att hanteras, vilket betyder att alla sidor på hemsidan har besökts och laddats ned. Genom att exportera listan över alla besökta sidor kan administratören även få en ”site map” av hemsidan.

En spindel är ett modulärt program, som kan användas till att utföra en rad olika uppgifter. Som omnämnt i kapitel 2, så följer de flesta Spindlar en uppsättning riktlinjer, som underlättar vid implementation och design. Hastighet är en viktig egenskap hos en modern Spindel, vilket ofta leder till att de implementeras på ett sådant sätt att de exekverar parallellt över flera trådar på samma gång.

Teoretiskt sätt så fungerar en spindel precis som en vanlig besökare, som navigerar en hemsida genom en webbläsare. En spindel navigerar dock en hemsida i en mycket högre hastighet, och skickar många gånger fler förfrågningar på kortare tid jämfört med en vanlig besökare. För att utnyttja en spindel till fullo, så ska spindeln kunna hantera samma data som kan stötas på vid navigering i en vanlig webbläsare**,** såsom gif-, jpeg- och png-bilder, samt javascript med inbäddade länkar. Alla länkar kan potentiellt vara brutna.

En stor fördel med en spindel är den modularitet som ges. Efter implementation är det enkelt att lägga till ny, och ändra redan existerande funktionalitet. Detta tillåter utvecklare att alltid hålla spindeln uppdaterad, och i fas med moderniseringen och utvecklingen av internet. Om användaren av spindeln i framtiden vill utveckla spindeln, för att samla in annan data än enbart länkar, så kan det enkelt implementeras. Vad än användaren vill samla in, med hjälp av spindeln, så kan det enkelt läggas till funktionalitet för detta vid samma del av programmet där länkar extraheras från en nedladdad sida.

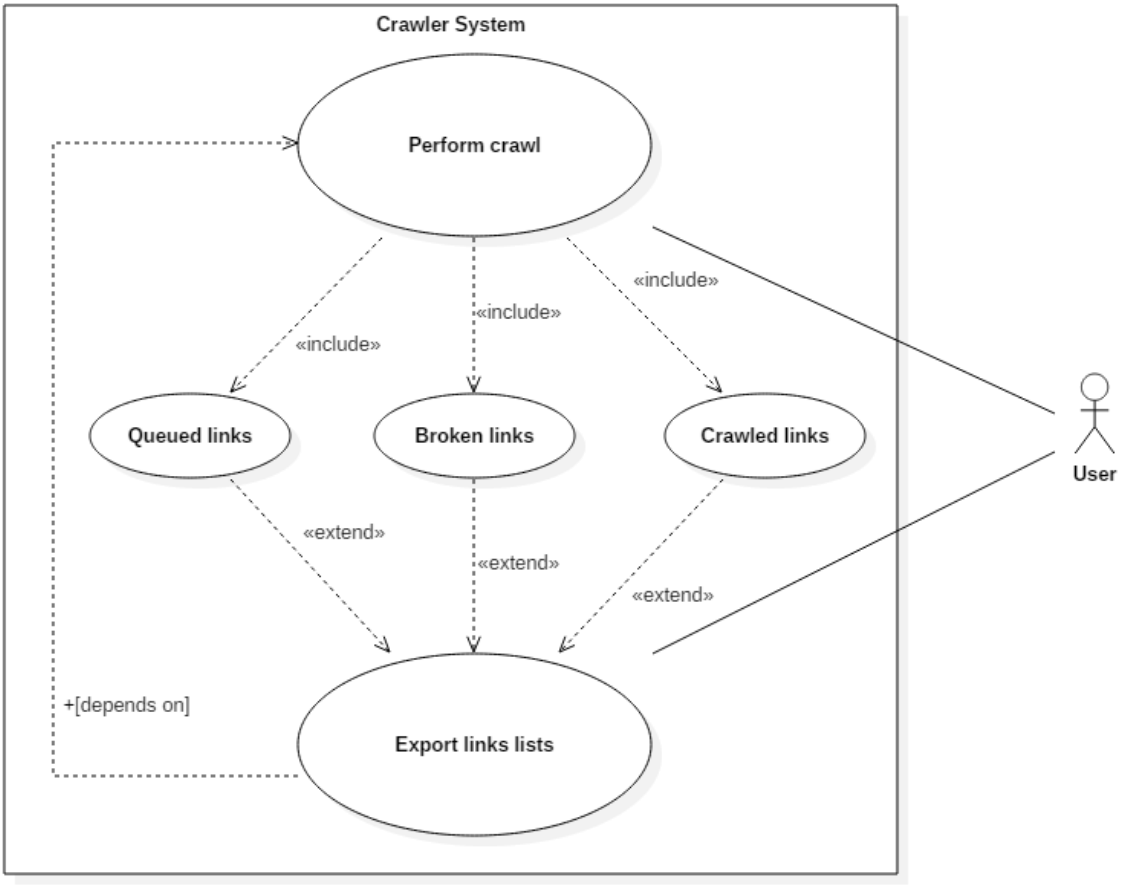
3.3.3 Min lösning

Spindeln kommer i grunden sträva efter att underlätta arbetet för systemadministratören. Denne kommer vid exekvering att hitta alla brutna länkar på hemsidan, samtidigt som en indexering (eller en site map) av alla sidor ges i form av alla funna länkar. Detta kommer i längden gynna hemsidans besökare också. Då det inte finns några brutna länkar på hemsidan, så kommer besökaren utan problem kunna navigera till de sidorna den vill, utan att stöta på några felmeddelanden.

En användare kan, så klart, uppnå samma resultat genom att manuellt leta igenom hemsidan för att hitta brutna länkar. Detta skulle ultimat ge samma resultat som spindeln, men det skulle ta betydligt längre tid och inte alls vara lika effektivt.

Bortsett från att manuellt samla in brutna länkar, så finns det en rad online-verktyg som kan användas. Dessa verktyg kan dock kosta en hel del pengar, samtidigt som verktyget inte är specifikt utvecklat för hemsidan. Denna lösning kommer kunna appliceras på vilken hemsida som helst, men kommer i första hand vara designad för att passa MIO:s behov. Spindeln kan i efterhand utvecklas vidare, och ändras efter behov.

3.4 Projektdesign



Figur 3.4: Användningsfalldiagram

En spindel kan ha ett flertal användningsfall tillgängliga för systemadministratören, som i diagrammet ovan agerar användare. Det huvudsakliga användningsområdet är att hitta, samla

in och exportera trasiga länkar. Efter att länkar samlats in, kan användaren välja att exportera en, eller flera av de genererade listorna.

Då länkar samlas in, eller en “crawl” utförs på vald hemsida, så kommer spindeln att generera tre olika listor. En lista innehåller köade länkar som väntar på att hanteras, en andra lista innehåller hanterade länkar medan en tredje innehåller alla funna brutna länkar. Dessa tre listor kommer alltid att skapas vid starten av ett nytt projekt.

Listan innehållande brutna länkar är av mest intresse för användaren, då det är dessa spindeln är designad för att hitta. Om användaren vill ha ett fullständigt index, över vilka sidor som just nu finns uppe på hemsidan, så kan listan innehållande hanterade länkar exporteras. Användaren kan välja en, eller flera, av dessa listor för exportering. Spindeln måste dock ha exekverat minst en gång för att det ska finnas någon lista att exportera.

3.5 Systemfunktioner

Vid första exekvering, vid starten av ett nytt projekt, så kommer spindeln kräva en startpunkt som input från användaren. Denna startpunkt avser en hemsida, eller en subdomän på en hemsida i form av en URL, som bestämmer var spindeln kommer att börja arbetet. Användaren kommer att vara en person inom ett företag eller privat, som har nytta av att hitta alla brutna länkar på en hemsida, en systemadministratör eller liknande. Ett nytt projekt medför nya filer, placerade i en ny mapp. Om inputen redan har hanterats, och en projektmapp med tillhörande filer redan existerar, så kommer spindeln istället att fortsätta där den senast slutade genom att hantera nästa länk i kön.

Programmet kommer alltid att skapa tre olika textfiler, där användaren själv bestämmer vilken eller vilka av dessa som ska exporteras. Den första filen innehåller en lista på alla de länkar som hittats av spindeln, och som väntar på att hanteras. Denna lista refereras till som programmets kö. I den andra filen kommer alla hanterade länkar att placeras, alltså alla de länkar som spindeln har besökt. Alla länkar som finns denna fil har vid ett tillfälle befunnit sig i kön, innan den hanterats och flyttats från kön. Vid en fullständig genomsökning av en hemsida kommer denna fil innehålla alla länkar som finns på hemsidan, och representerar en ”site map” av hemsidan. Till den tredje filen skrivs alla brutna länkar som spindeln hittar.

Efter en fullständig genomgång av en hemsida kommer alla brutna länkar som funnits vara skrivna till den tredje textfilen, som då är redo att exporteras av användaren.

3.6 Utvecklingsmiljö

Den IDE (Integrated Development Environment) som kommer att användas vid utvecklingen av detta projekt är Microsoft Visual Studio. Visual Studio är bland de mest använda utvecklingsmiljöerna. Det är snabbt och konfigurerbart, då paket och bibliotek enkelt kan laddas ned via den inbyggda NuGet Package Manager. Visual Studio innehåller också funktioner som tillåter utveckling av tekniskt sett vilket typ av program som helst, oavsett storlek. Trots att det är designat för storskaliga projekt, så är det alltså också ett bra alternativ vid utveckling av mindre. Vid kodredigering används en funktion som heter *IntelliSense*, som tillåter automatisk komplettering av den kod som skrivs, vilket markant ökar produktiviteten och utvecklingshastigheten. Visual Studio används är som tidigare nämnt ett populärt val, och används av nybörjare såväl som professionella utvecklare.

Projektet kommer att utvecklas i programmeringsspråket C#. Programmeringsspråket Python var först tänkt att användas, men då planerna på att implementera ett GUI för projektet valdes C# tillslut. Med C# kan utvecklare med enkelhet utveckla Windows-applikationer, då det finns stöd för detta inbyggt, och inga externa bibliotek behöver installeras.

3.7 Summering

I detta kapitel har projektets mål och syfte presenterats i detalj. Vilka problemområden som existerar idag, samt lösningen på dessa problem. Projektdesignen presenteras också, samt projektets komponenter. Alternativa lösningar, och varför dessa inte valts, har också diskuterats.

4 Implementation

I kapitel 4 ges en mer detaljerad och teknisk beskrivning av hur produkten har implementerats. Här förklaras tanken bakom projektets implementering, alla systemets klasser listas och förklaras precis som klassernas funktioner. Slutligen så presenteras det grafiska användargränssnittet.

4.1 Projektimplementation

Som en lösning på det problem som diskuterats i tidigare kapitel, så har jag valt att utveckla en spindel. Med hjälp av denna spindel kommer användaren utan vidare ansträngning att hitta brutna länkar, som enkel kan exporteras för att sedan kunna åtgärdas. Tanken är att processen ska vara så enkel som möjligt för användaren, samtidigt som spindeln jobbar snabbt och tar sig igenom hemsidan utan att märkbart påverka dess prestanda negativt. Genom att jobba över multipla trådar parallellt, så kan spindeln hantera flera unika sidor samtidigt och kan på så vis jobba betydligt snabbare än en spindel som bara hanterar en sida åt gången. Ett högre antal trådar betyder att spindeln jobbar snabbare, samtidigt som servern får hantera flera förfrågningar på kortare tid. Användarens maskin kommer även den påverkas av antal valda trådar, då ett högre antal trådar kräver mer prestanda för att kunna exekvera som tänkt.

Användaren kommer få möjligheten att mata in en URL till programmet, som specificerar var spindeln startar. Den inmatade URL:en kan vara hemsidans förstasida, eller en subdomän - användaren väljer fritt var den vill börja.

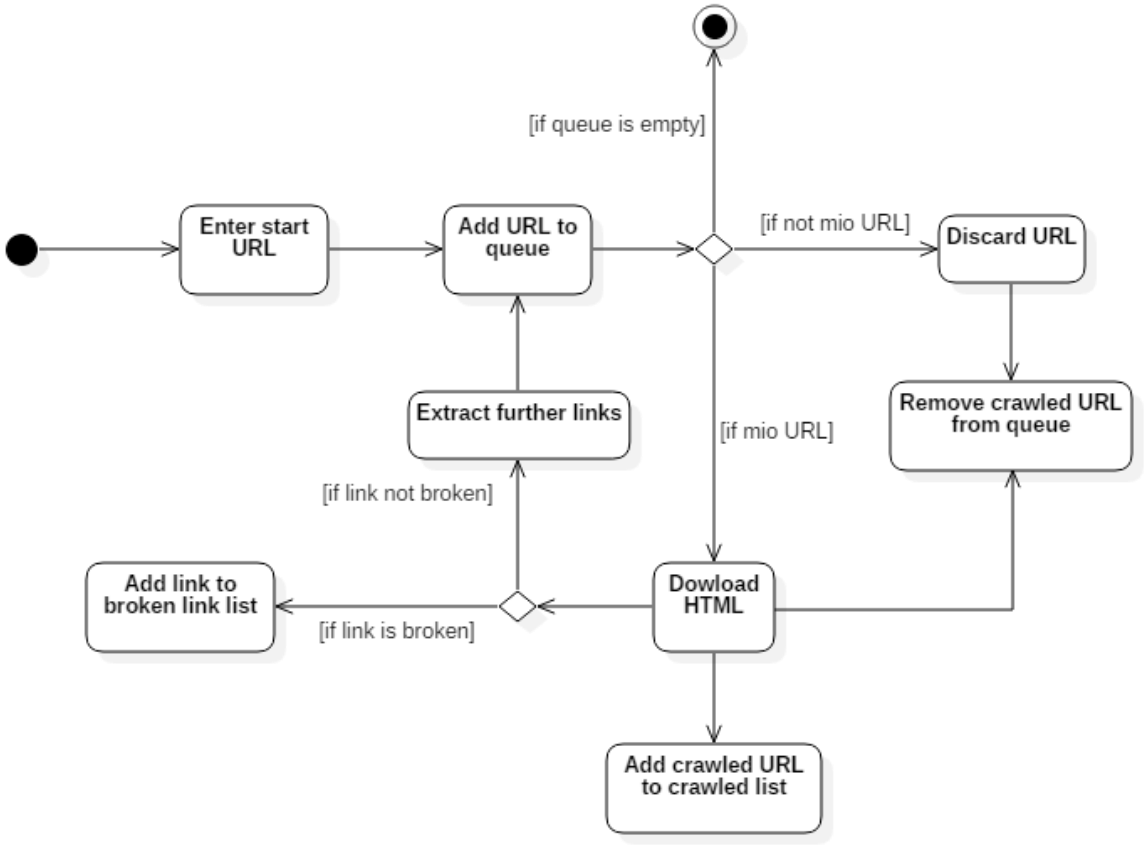
Då användarens input är inmatad, startar användaren själv spindeln genom att klicka på “Start”. Programmet kan när som helst under exekveringen stoppas, för att sedan återupptas där det avslutades.

Spindeln kommer skapa en ny mapp för varje projekt, samt tre textfiler: ”queue.txt”, ”crawled.txt” och ”broken.txt”. Projektmapp och filer kommer endast att skapas första gången spindeln körs, då de ännu inte existerar. I samband med att dessa filer skapas kommer även den URL som matades in av användaren att krypas. Alla länkar som hittas på denna sida

kommer att placeras i textfilen “queue.txt” som representerar alla länkar som ännu inte hanterats av spindeln. Den nyligen hanterade länken kommer i sin tur att placeras i filen “crawled.txt” som representerar alla länkar som har hanterats av spindeln. För varje länk som hanteras kollas statuskoden på varje HTTP Response, som spindeln tar emot från servern. Om denna statuskod överstiger 400, så tyder det på att länken som spindeln försöker komma åt är felaktig, på ett eller annat sätt. Då spindeln hittar en sådan länk placeras den i en separat fil, “broken.txt”, som representerar alla brutna länkar funna av spindeln. Användaren kan välja att exportera en eller flera av dessa textfiler.

4.2 Aktivitetsdiagram

4.2.1 Exekvering

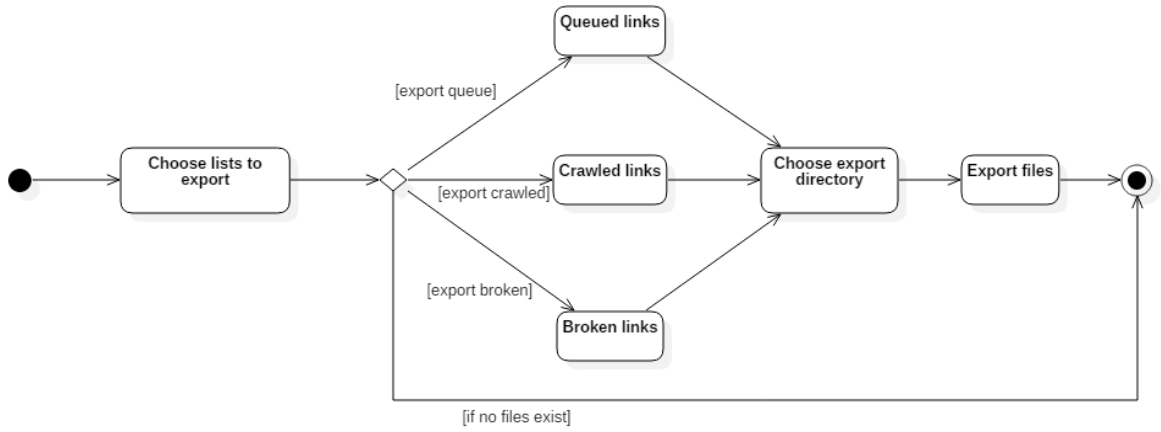


Figur 4.1: Aktivitetsdiagram över exekvering

Ovanstående aktivitetsdiagram visar en exekvering av systemet. Vid programmets start ombes användaren att mata in en URL, som input. Denna URL avser vilken hemsida, eller subdomän

på en hemsida som användaren vill applicera spindeln på. Då användaren väljer att starta programmet så kommer den URL som matats in att läggas till i programmets kö. Programmet kommer sedan att hantera denna första URL på huvudtråden, utan att skapa några nya då det än så länge bara finns en länk i kön. Programmet kollar nu om länken är en giltig URL. Om vald URL är giltig så besöks sidan och dess HTML-kod laddas ned och sparas i ett HTML-dokument. Länken kommer sedan att tas bort ur kön och läggas till i listan som innehåller redan besökta länkar, för att undvika att länken besöks igen senare. I samband med att sidan besöks, så kollar programmet om länken är bruten. Om länken är bruten så läggs den till i listan innehållande brutna länkar, annars extraheras alla länkar som finns på sidan och placeras i programmets kö. Detta kommer att fortsätta tills programmets kö inte längre innehåller några länkar att besöka.

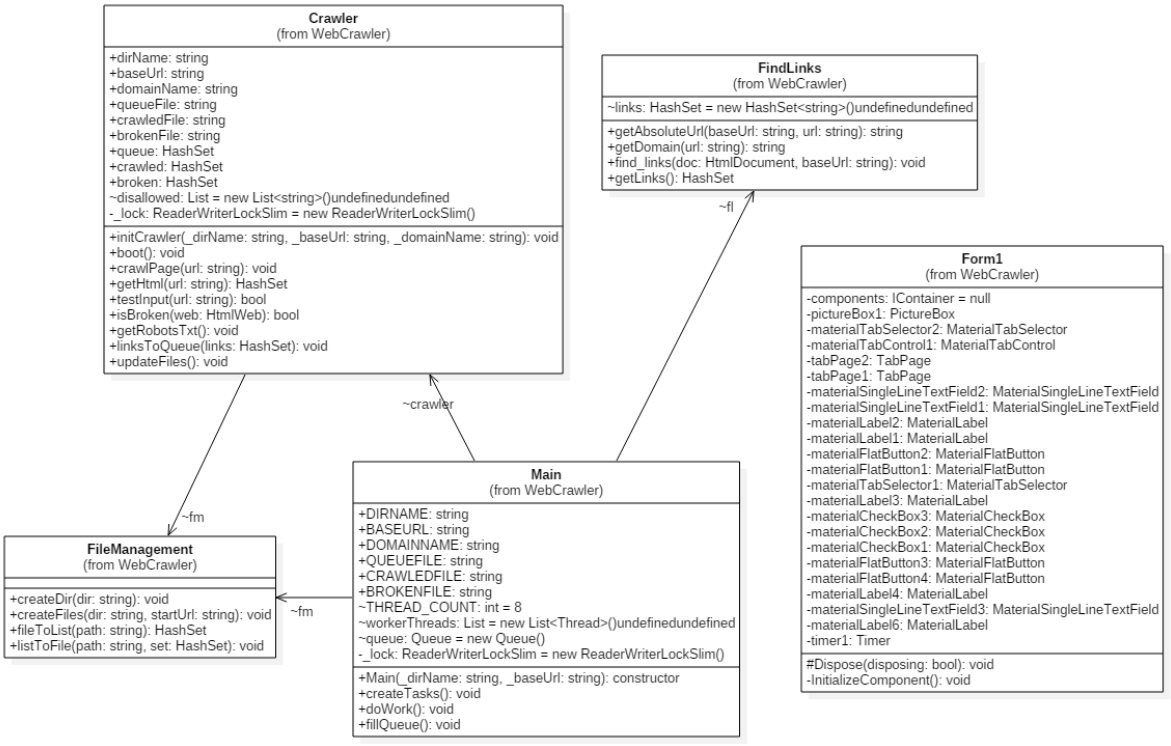
4.2.2 Exportera listor



Figur 4.2: Aktivitetsdiagram över exportering

En användare kan också välja att exportera de listor som genererats av programmet, förutsatt att det finns listor att exportera. Användaren kan välja en, eller flera listor för exportering.

4.3 Systemkomponenter



Figur 4.3: Klassdiagram över systemet

4.3.1 FileManagement

Denna klass innehåller funktioner som hjälper till vid läsning av data i filer, samt skrivning till dem. Som namnet antyder är klassens främsta funktion filhantering. Klassen används även till att initiera ett nytt projekt, då en projektmapp och filer behöver skapas. Funktioner som konverterar listor av data till textfiler, och tvärtom, finns också i denna klass.

Klassfunktioner:

- **createDir** – Funktionen används då en ny projektmapp skapas. Detta sker i klassen ”Crawler”.
- **createFiles** – Denna funktion kallas på i samband med föregående funktion, och skapar tre textfiler unika för varje projekt.
- **fileToList** – En funktion vars uppgift är att konvertera en textfil till ett HashSet. Textfilen innehåller flera strängar som alla skrivs till motsvarande HashSet i programmet. I samband med att funktionen kallas på så låses åtkomst till filen för andra trådar, för att undvika att filen ändras under tiden.

- **listToFile** – Den funktion som används då programmet måste skriva data från till en textfil. Åtkomst till filen låses även i samband med att denna funktion används, för att två olika trådar inte ska skriva till filen samtidigt.

4.3.2 FindLinks

Den mest prominenta funktionen denna klass har är att hitta alla länkar som existerar på en given sida, genom att tolka nedladdad HTML-kod. Alla länkar som hittas sparas i ett temporärt HashSet, som sedan returneras. Utöver detta så finns funktioner som skapar och returnerar en så kallad absolut URL, baserat på den URL som användaren matat in som startpunkt.

En absolut URL kan referera till en sida på vilken domän som helst, medans en relativ URL innehåller mindre information (inget domännamn) och kan bara referera till en sida på samma domän.

Klassfunktioner:

- **getAbsoluteUrl** – Denna funktion används då en länk har hittats på en sida, i funktionen find_links. Många länkar som hittas är endast relativa, och måste göras absoluta innan de sparas.
- **getDomain** – Funktionen returnerar domännamnet hos en given URL, med syftet att säkerställa att alla länkar som besöks endast finns inom samma domän.
- **find_links** – Den funktion som tolkar ett HTML-dokument och extraherar alla länkar, relativa eller absoluta. Funktionen kallas på så fort en ny sida besökts.
- **getLinks** – Funktionen returnerar det HashSet som använts för att spara de länkar som extraherats från en sida.

4.3.3 Crawler

Huvudfunktionen för denna klass är att ladda ned HTML-kod, som sedan kommer matas in i klassen ”LinkFinder”, där alla länkar extraheras. Klassen innehåller också funktioner som används vid uppstart av ett nytt projekt, där en rad klassvariabler som delas mellan alla aktiva trådar initieras. I denna klass identifieras också alla trasiga länkar.

Klassfunktioner:

- **initCrawler** – Här initieras alla klassvariabler som delas mellan de aktiva trådarna. Funktionen kallas på vid exekvering, av konstruktorn i klassen ”Main”. Funktionerna ”boot” och ”crawlPage” kallas även de på, för att skapa en projektmapp och filer.
- **boot** – Denna funktion skapar de nödvändiga projektfilerna, och projektmappen. De tre HashSet som representerar textfilerna fylls också med data från motsvarande textfil.
- **crawlPage** – Denna funktion utför alla nödvändiga steg för att en länk ska räknas som hanterad. Genom att kalla på andra funktioner, som i sin tur också kallar på andra funktioner, så laddas HTML ned och tolkas. De länkas som hittas läggs sedan till i textfilerna.
- **getHtml** – Kallas på i ovanstående funktion. Här laddas en sidas HTML-kod ned. Funktionen kontrollerar även om den länk som hanteras är bruten eller inte.
- **isBroken** – Den funktion som används för att kontrollera om en länk är bruten.
- **getRobotsTxt** – Vid varje ny projektstart så laddas en hemsidas robots.txt ned, för att tolkas. Om filen innehåller några subdomäner som inte ska besökas så respekteras det.
- **linksToQueue** – Då de länkar som finns på en sida har extraherats så lägger denna funktion till dem i programmets kö.
- **updateFiles** – Då alla länkar är hanterade och alla programmets HashSets är uppdaterade, så uppdaterar denna funktion de tre textfilerna för att matcha dem.

4.3.4 Main

Den klass som initierar systemet, med hjälp av funktioner från övriga klasser. Här skapas också trådar, som tilldelas länkar från den gemensamma kön. Den gemensamma kön fylls på allteftersom den motsvarande textfilen fylls med nyfunna länkar.

Klassfunktioner:

- **Main** – Klassens konstruktor. Denna kallas på genom användargränssnittet, då användaren väljer att starta programmet. Konstruktorn sätter igång programmet, och allt sker automatiskt härefter, utan input från användaren.
- **createTasks** – Denna funktion kallas på i klassens konstruktor. Ett förbestämt antal trådar skapas.
- **doWork** – Denna funktion kopplas till varje ny tråd som skapas. Funktionen kommer att ge varje tråd en unik länk att hantera, från den gemensamma kön. Den gemensamma

kön kommer även att konstant fyllas på här, för att hela tiden matcha den motsvarande textfilen.

4.4 Användargränssnitt

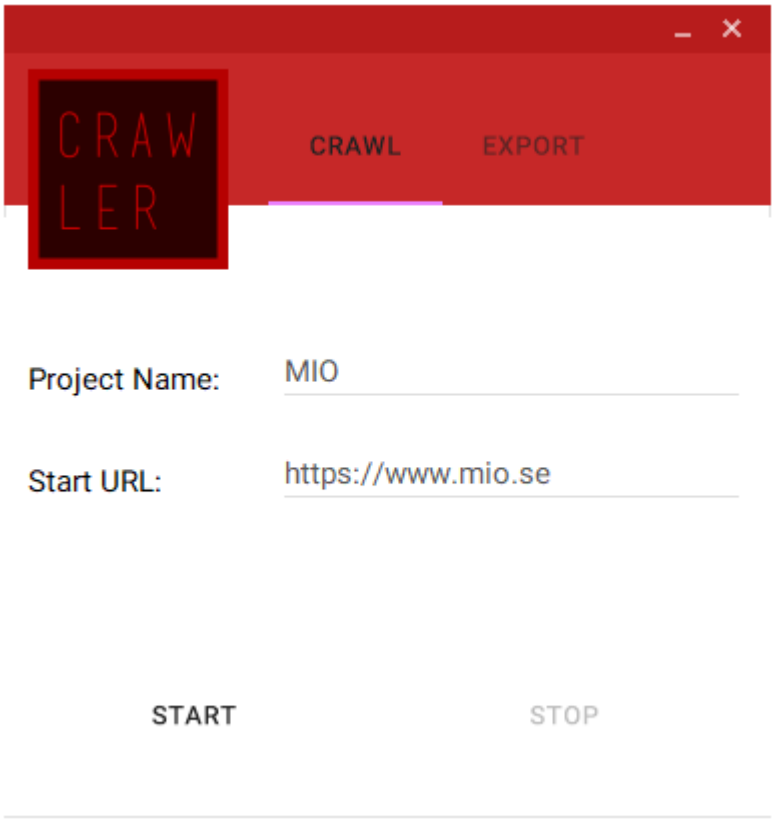
Programmets grafiska användargränssnitt är tänkt att underlätta användningen av programmet, och ge användaren en mer användarvänlig upplevelse. Alternativet är att låta användaren köra programmet genom kommandotolken, eller liknande.

4.4.1 Material Design

Designen är tänkt att följa och efterlikna Googles “Material Design” [18]. Målet är ett så användarvänligt användargränssnitt som möjligt, som i teorin vem som helst ska förstå och kunna använda utan alltför detaljerade instruktioner. Samtidigt som hög användarvänlighet står i fokus, så ska designen vara modern och snygg.

Material Design är i grunden avsedda för applikationer skapade för Google-plattformar, ex. Android-applikationer, Gmail och så vidare. Principerna och tanken bakom är emellertid detsamma gällande detta program. För att kunna efterlikna Material Design så har jag använt mig utav biblioteket ”MaterialSkin”, som finns tillgängligt i Visual Studios NuGet.

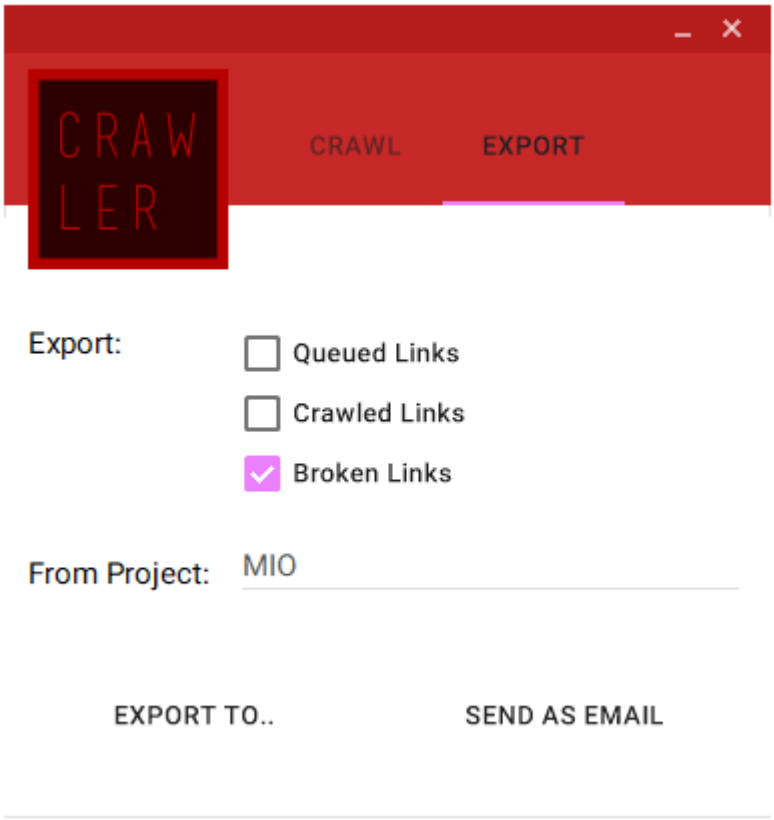
4.4.2 Crawl



Figur 4.4: Programmets startsida

I skärmklippet ovan visas startsidan för programmet. Spindeln kommer inte att börja arbeta innan användaren försett den med input och valt att starta. Vid programstart kommer textfälten innehålla platshållare med exempel på hur inputen kan se ut. Då programmet är skapat för MIO, så visas här startsidan för MIO:s hemsida. Om användaren väljer att starta programmet utan att själv ge någon input, så kommer det att starta med dessa standardvärden.

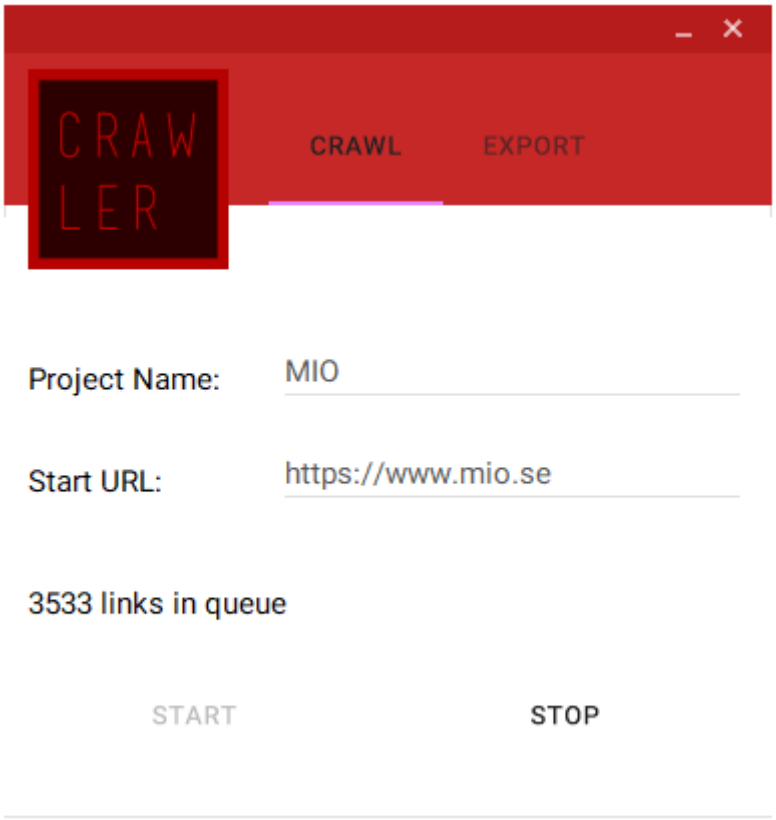
4.4.3 Export



Figur 4.5: Programmets exporteringsflik

Vid exportering av de länkar som hittats av spindeln, ges användaren tre alternativ. Användaren kan välja att exportera en eller flera av de listor som skapats under exekvering. Spindelns huvudsyfte är att hitta brutna länkar på en hemsida, därav är listan innehållande dessa brutna länkar av störst intresse för användaren. Eftersom flera olika projektmappar kan finnas samtidigt, så får användaren själv välja från vilket projekt som listorna ska exporteras.

4.4.4 Exekvering



Figur 4.6: Programmet under exekvering

Under exekvering kommer användaren i realtid att informeras om hur många länkar som finns i kön. Eftersom programmet jobbar i bakgrunden, så ges denna feedback till användaren för att denne ska vara säker på att spindeln fortfarande arbetar.

4.5 Summering

Spindeln kommer att implementeras med den ”Parallelization Policy” som diskuterades i kapitel 2 [11]. För att underlätta användarens jobb bör spindeln vara så snabb och effektiv som möjligt, därför jobbar den över flera trådar samtidigt. Den tar också hänsyn till en hemsidas ”robots.txt” [6].

Spindeln kommer också att implementeras med ett grafiskt användargränssnitt, som ökar användarvänligheten hos spindeln samtidigt som den förser användaren med feedback under exekvering.

5 Utvärdering

I tidigare kapitel har jag diskuterat de mål som satts för utvecklingen, och vilket syfte produkten har tänkt att fylla. För att styrka att dessa är uppnådda har jag utfört ett test med olika input, eller parametrar, för att sedan ställa resultatet mot de mål som satts för projektet.

Jag kommer att utföra ett test kopplat till produktens huvudmål, där fokus ligger på effektivitet. Produkten är designad på ett sådant vis att processen för att hitta och samla in länkar ska utföras på ett så effektivt sätt som möjligt. Desto snabbare en bruten länk är funnen, desto tidigare kan en administratör åtgärda den. Detta följer den riktlinje, eller policy som introducerades i kapitel 2; ”Parallelization Policy” [11]. Enligt denna policy ska en spindel implementeras parallellt, för att på ett så effektivt sätt som möjligt kunna jobba med flera länkar samtidigt. Därför har spindeln implementerats på ett sådant sätt att den jobbar över flera trådar samtidigt, där fler trådar i teorin betyder att fler länkar kan hanteras samtidigt.

Huvudfokus för denna utvärdering ligger alltså i att undersöka spindelns effektivitet.

5.1 Testet

5.1.1 Hårdvara

Testet har utförts på min personliga dator, en Dell XPS (2018) utrustad med 64 bitars Windows 10. Den har en Intel i7-8550U CPU och 8 GB RAM.

5.1.2 Testdesign

Testet kommer fokusera på att testa spindelns hastighet, eller effektivitet. Spindeln är designad att exekvera på ett flertal olika trådar, som jobbar parallellt. Tanken bakom implementationen är att öka effektiviteten, och produktiviteten med vilken spindeln laddar ned och hanterar länkar. En länk ses som hanterad då all HTML från den sidan länken representerar är nedladdad, och alla länkar funna i denna HTML-kod är extraherade och placerade i kön, efter vilket den hanterade länken placeras i listan innehållande hanterade länkar. För att styrka påståendet om att fler trådar resulterar i en effektivare spindel så har ett test, eller experiment, designats och utförts. Effektivitet mäts i hur många länkar som spindeln hanterar under en viss tid.

Programmet kommer att exekvera under 60 minuter, först på 1 tråd följt av 8 trådar. Under utvecklingen av spindeln har de flesta testerna skett på just 8 trådar, vilket hittills har fungerat oproblematiskt. Valet av att jämföra detta mot exekvering på 1 tråd, baseras på att 1 tråd är det lägsta antalet som krävs för att köra programmet.

De parametrar som kommer att påverkas under testet är antal trådar, medans outputen kommer att bli det antal länkar som hanterats under den bestämda tiden. Antalet hanterade länkar är av störst intresse under detta test, då antalet hanterade länkar indikerar spindelns hastighet (effektivitet), mätt i länkar per minut.

5.1.3 Testresultat

Under det första testet exekverade programmet över 1 tråd under 60 minuter. Under denna period har 9982 unika länkar hanterats av spindeln. Detta betyder att spindeln under en timmes konstant exekvering klarar av att i snitt hantera ca 166 länkar per minut (9982/60). Detta är det absolut minsta spindeln klarar av, då den enbart har exekverat över 1 tråd.

Då programmet istället exekverar över 8 trådar under samma tidsperiod, så ser vi att betydligt fler länkar har hanterats av spindeln. Hela 18 371 länkar har nu hanterats, vilket resulterar i att spindeln i snitt har hanterat 306 länkar per minut.

List	**1 Tråd**	**8 Trådar**
Crawled	9982	18 371
Queue	13 667	8109
Broken	9	9
Total	23 649	26 480

Tabell 5.1: Testresultat

Tabellen ovan visar även hur många länkar som finns i programmets kö under samma tidsperiod, samt det totala antalet länkar som hittats och hanterats när trådantalet är 1 jämfört med 8.

Vid exekvering över 1 tråd, jämfört med 8, så ser vi att det finns fler länkar kvar i kön. Detta betyder att spindeln klarar av att hålla kön kortare vid exekvering över flera trådar, då varje länk som placeras i kön hanteras snabbare. Alla de länkar som skrivs till kön kommer förr eller senare att hanteras av spindeln och skrivas till listan innehållande hanterade länkar.

Det totala antalet hanterade och köade länkar är nästintill detsamma. Skillnaden är dock att spindeln hunnit hantera betydligt fler länkar då den exekverat över 8 trådar. En hanterad länk är som tidigare nämnt en sida som besökts, och vars länkar har extraherats och placerats i kön. Att besöka en sida, och att samla in de länkar som finns på sidan är den del i processen som tar längst tid. För varje hanterad länk som placeras i listan över hanterade länkar, så kan flera nya länkar skrivas till kön. Därför är antalet hanterade länkar av större intresse än det totala antalet funna länkar. En funnen länk är inte detsamma som en hanterad länk och ger inte en verklig representation av spindelns effektivitet.

5.2 Summering

En spindel vars implementering följer ”Parallelization policy” [11], och arbetar parallellt kommer att hantera länkar snabbare, och är således mer effektiv än en spindel som inte är implementerad att jobba i parallell. En mer effektiv spindel samlar in länkar snabbare, och på så sätt får användare snabbare koll på vilka länkar på hemsidan som är brutna vilket betyder att de snabbare kan åtgärdas.

6 Slutsats

I detta kapitel summeras både slutprodukten och utvecklingsprocessen. Projektets avgränsningar redogörs, och förslag på framtida funktionalitet ges. Ett alternativt framtida användningsområde presenteras också.

6.1 Projektsummering

6.1.1 Slutprodukten

Det färdiga programmet fungerar precis som väntat. Flera lyckade tester har utförts, främst på MIO:s hemsida. Slutproduktens användare kommer troligtvis att vara en administratör, eller annan personal som har möjligheten att ändra hemsidans innehåll (ta bort länkar). Produkten kommer inte att användas dagligen, utan endast vid behov. Främst inför större reor och kampanjer då många länkar ändras och flyttas, samtidigt som nya produktsidor genereras.

Valet att implementera spindeln som parallell har ökat effektiviteten med vilken sidor laddas ned och hanteras. Som testresultatet i föregående kapitel visar så jobbar en parallell spindel betydligt snabbare än en som endast verkar över en tråd. I samband med att hårdvaran utvecklas så kan ytterligare trådar användas vid exekvering, vilket kan öka effektiviteten ytterligare.

Vid projektets start var jag osäker på om jag skulle ta hänsyn till en hemsidas ”robots.txt” eller inte. Då produkten främst utvecklas för MIO:s hemsida, som jag vet endast har en subdomän som den inte tillåter spindlar att besöka, så var tanken först att hårdkoda spindeln att undvika denna subdomän. Tillslut valde jag ändå att implementera en funktion som vid varje uppstart laddar ned en hemsidas ”robots.txt” och tolkar filen, ifall MIO i framtiden utökar filen med fler subdomäner, eller om spindeln appliceras på en annan hemsida.

Uppdragsgivaren är nöjd med slutproduktens funktion, och ser det grafiska användargränssnittet som en bonus. Att implementera ett GUI var på förhand inte ett krav från uppdragsgivarens sida, men under processens gång kom vi gemensamt fram till att om tiden fanns att utveckla ett så skulle det underlätta användningen av slutprodukten.

6.1.2 Utvecklingsprocessen

Utvecklingen var tänkt att delas in i olika sprintar, där en SCRUM board används för att välja vilken funktion som skulle implementeras härnäst. Jag upplevde att detta kändes mer som en börda än en fördel då jag arbetat själv. Användningen av en SCRUM board upplever jag passar bättre in vid produktutveckling i större grupper, för att alltid ha koll på hur långt kommen utvecklingen av varje funktion är. Jag trodde på förhand att detta skulle underlätta även mitt arbete, men märkte snabbt att så länge jag var noga med dokumentation så hade jag väldigt bra koll på vad som var färdigt och vad som stod på tur att utvecklas.

Jag skapade väldigt tidigt i processen en klar bild över vad jag ville uppnå, tillsammans med min handledare på MIO. Utifrån detta tog jag fram en enklare projektdesign, som har legat som grund för hela utvecklingen. Till denna design tog jag även fram ett aktivitetsdiagram, där en exekvering av programmet illustrerades. Att skapa ett aktivitetsdiagram som tydligt visar hur jag på förhand ville att slutprodukten skulle fungera har hjälpt under hela utvecklingsfasen. Aktivitetsdiagrammet har underlättat i arbetet med att utforma de klasser och funktioner som krävts, och jag känner att jag fått en bättre överblick på projektet i helhet.

Under hela projektets gång har jag haft möjligheten att diskutera projektet med min handledare på plats på MIO. Under projektets planeringsfas var detta till stor hjälp, då vi tillsammans kunde ta fram krav på vad slutprodukten. Tanken var att hålla i en kortare demo varje vecka för att involvera kunden så mycket som möjligt, men då jag till en början inte hade så mycket mer än designen att visa så glömdes det bort. Mot slutet av projektet hölls istället några demonstrationer av produkten som vid varje demo var allt mer komplett.

De olika riktlinjerna som kan följas vid implementering av en spindel är något jag kommer att ta med mig. Framförallt de som jag tog hänsyn till vid utvecklingen av min egna spindel. Vid implementeringen har jag tagit hänsyn till ”Parallelization policy” [11], då spindeln exekverar över flera trådar. Jag har tagit hänsyn till ”Politeness policy” [11] då spindeln laddar ned och tolkar innehållet i en hemsidas ”robots.txt”. Jag har även till viss del tagit hänsyn till ”Selection policy” [11], då spindeln endast följer länkar inom samma domän.

6.2 Avgränsningar

Spindeln som jag utvecklat kommer inte att lösa alla de problem och komplikationer som diskuteras i kapitel 2. Ett sånt här arbete kan i grunden bli så stort man vill, då en spindel kan implementeras på så många olika sätt och har så många olika användningsområden.

De riktlinjer som inte följts har enligt mig inte tillfört något till att uppnå målet med projektet. Däribland ”Re-visitation policy”, som bör följas om systemets användare har för avsikt att med jämna mellanrum återbesöka en hemsida automatiskt. Jag motiverar valet av att bortse från denna policy med att användaren av min slutprodukt endast har för avsikt att besöka en hemsida i taget, och kan då återbesöka samma sida när den behagar för att på så vis få en så uppdaterad indexering av hemsidan som möjligt.

6.2.1 Framtida arbete

De riktlinjer och komplikationer jag valt att ta hänsyn till är de som jag ansett vara nödvändiga att följa för att uppnå målet med detta projekt, som var att underlätta insamlingen av brutna länkar för en systemadministratör. De riktlinjer och komplikationer som ännu inte tagits hänsyn till, eller implementerats, kan självklart implementeras i framtiden då behovet av dessa finns.

Utöver att lösa dessa kvarstående problem, och implementera de riktlinjer som inte tagits hänsyn till, så kan spindeln utvecklas och funktionalitet för att täcka ytterligare användningsområden kan adderas till den redan existerande produkten.

Ett ytterligare konkret, framtida användningsområde har diskuterats mot slutet av projektet. I dagsläget använder sig MIO av olika cache-minnen, där varje nytt anrop till sidor sparas för att de snabbt ska kunna presenteras för en besökare. Produktdata som finns i databasen läses in och presenteras på sidan. Om en sida finns i cache-minnet så kommer inte produktens data att hämtas på nytt från databasen, utan denna produktdata i databasen kan ändras utan att påverka sidor som redan finns i cache-minnet. Cache-minnet töms varje natt, vilket betyder att de första personerna som besöker hemsidan på morgonen kommer att uppleva hemsidan som långsammare än de besökare som kommer efter. Vid starten på större reor och kampanjer kan trafiken vara så hög till en början att hemsidan slutar svara helt då nya anrop konstant måste göras till produktdatabasen.

Förslaget var att då cache-minnet töms innan en större reastart så används spindeln för att treversera hemsidan och då besöka så många länkar den hinner innan verkliga besökare kommer in på hemsidan. På så sätt kommer de flesta sidorna att redan finnas i cache-minnet, och mängden samtida anrop till produktdatabasen kommer att sjunka och hemsidans prestanda stannar inte av.

6.3 Slutord

Detta projekt har resulterat i en färdig produkt, som kan appliceras på uppdragsgivarens hemsida vid behov. Användandet av produkten ger användaren en lista innehållande alla brutna länkar som finns på hemsidan, varefter användaren själv kan åtgärda dessa. Vid behov kan även en ”site map” (en lista över alla länkar som finns på hemsidan) fås.

Utöver utvecklingen så har en utvärdering av projektet gjorts, där produkten har analyserats och resultatet har vägts mot både de riktlinjer och policys som beskrivits i tidigare kapitel, samt de mål som presenterades i kapitel 1.

Vidare har förslag på framtida användningsområden och vidareutveckling givits.

Referenser

[1] MIO, ”Delårsrapport 2017,” [Online]. Tillgänglig: https://www.mio.se/upload/om-oss/Dela%CC%8Arsrapport%20maj-okt%202017.pdf. [Hämtad: 2018-05-12].

[2] Oracle, ”Explanation of Failure Codes,” [Online]. Tillgänglig: https://docs.oracle.com/cd/E14269_01/doc.451/e14266/result_codes.htm. [Hämtad: 2018-06-14].

[3] R. Fielding, J. Reschke, ”Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”, 2014.

[4] Kissmetrics, ”How Loading Time Affects Your Bottom Line,” [Online]. Tillgänglig: https://blog.kissmetrics.com/loading-time/. [Hämtad: 2018-05-12].

[5] C. Olston and M. Najork, “Web crawling,” Foundations and Trends in Information Retrieval, vol. 4, no. 3, 2010.

[6] Wikipedia, ”Robots Exclusion Standard,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Robots_exclusion_standard. [Hämtad: 2018-05-12].

[7] Wikipedia, ”Site map,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Site_map [Hämtad: 2018-05-12].

[8] T. Berners-Lee, L. Masinter, M. McCahill, ”Uniform Resource Locators (URL)”, 1994.

[9] Wikipedia, ”Hyperlink,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Hyperlink. [Hämtad: 2018-05-12].

[10] Wikipedia, ”Denial-of-service attack,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Denial-of-service_attack. [Hämtad: 2018-05-12].

[11] C. Castillo, “Effective Web Crawling,” University of Chile, 2004.

[12] Wikipedia, ”Spider Trap,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Spider_trap. [Hämtad: 2018-05-12].

[13] B. Pinkerton, “Finding what people want: Experiences with the WebCrawler,” in Proceedings of the 2nd International World Wide Web Conference, 1994.

[14] S.Brin, L. Page, “The Anatomy of a large-scale hypertextual Web search engine,”Computer Science Department, Stanford University, 1998

[15] Alexa, ”google.com Traffic Statistics,” [Online]. Tillgänglig: https://www.alexa.com/siteinfo/google.com. [Hämtad: 2018-05-12].

[16] Wikipedia, ”Googlebot,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Googlebot. [Hämtad: 2018-05-12].

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P.Leach, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” 1999.

[18] Material Design, ”Material System Introduction,” [Online]. Tillgänglig: https://material.io/design/introduction/#principles. [Hämtad: 2018-05-12].