jsoup

***Skrevet af**: Daniel Winther Jensen*

***Vejleder**: Anders Børjesson*

***Skole**: Erhvervsakademi Sjælland - Campus Roskilde*

***Uddannelse**: Datamatiker - 4. semester*

***Afleveret d.**: 27. maj 2016*

***Anslag**: 23959*

Daniel Winther Jensen

Erhvervsakademi Sjælland - Campus Roskilde

Datamatiker – 4. semester

Indholdsfortegnelse

Daniel Winther Jensen

Erhvervsakademi Sjælland - Campus Roskilde

Datamatiker – 4. semester

Indledning

Denne synopsis danner grundlag for min mundtlige 4. semesterseksamen på datamatiker-uddannelsen i valgfaget Android-udvikling. Det valgte specialiseringsområde omfatter hvorledes scraping-biblioteket JSOUP[1] anvendes rent praktisk og hvilke muligheder det har at byde på.

Motivation

Min motivation for dette emnevalg udspringer sig i en en personlig interesse indenfor data scraping, da jeg i adskillige andre programmeringssprog har arbejdet med nævnte emne. Jeg kunne derfor godt tænke mig, at udforske de muligheder der måtte være indenfor dette område til Android-platformen. Særligt er jeg interesseret i, at undersøge om JSOUP API'et er tilstrækkeligt nok til, at kunne udtrække og fremvise data i en applikation, frem for at være bundet af f.eks. et RESTful API.

Problemformulering

Følgende problemformulering danner fundament for denne synopsis. Jeg har valgt at stille 1 grundlæggende hovedspørgsmål (markeret med fed) og herunder 5 underliggende spørgsmål til dette (markeret med kursiv).

Hvordan anvendes kodebiblioteket JSOUP til data scraping?

- *Hvad er JSOUP?*
- *Hvorfor anvende JSOUP til at hente data fra en given kilde?*
- *Hvilke begrænsninger forekommer der ved benyttelse af JSOUP?*
- *Hvilke muligheder forekommes der ved benyttelse af JSOUP?*
- *Hvilke alternativer til JSOUP eksisterer der?*

Metode

Måden hvorpå jeg ønsker at finde svar på ovennævnte problemformulering, vil primært foregå igennem researcharbejde. Denne research afgrænser til bl.a. læsning af relevant dokumentation, at se videovejledninger samt selv at udvikle egne eksperimenter. For at sikre sig kildernes korrekthed og relevans, vil jeg - så vidt det er muligt - arbejde primært med den respektive teknologis officielle dokumentation.

Eksperimenterne skal alle tjene det formål, at danne overblik samt blive klogere på hvorledes det valgte emne anvendes i egentlig praksis. Der vil derfor i sidst ende ikke være tale om et konkret færdigudviklet produkt, men derimod små prototyper, der med stor tydelighed demonstrerer teknologiens forskellige anvendelsesmuligheder i forskellige sammenhænge.

Planlægning

Planlægning- og struktureringsmæssigt har jeg bevidst valgt ikke at anvende nogen bestemt anerkendt systemudviklingsmetodik (herunder SCRUM, UP, XP osv.), da dette forløb er hverken team- eller produktorienteret, men derimod mynter sig mere på arbejde af researchmæssig karakter.

I løbet af denne proces vil jeg skiftevis arbejde med hhv. 2 forskellige aktiviteter; Den første aktivitet går på at udvikle eksperimenter og researche hvordan det valgte emne bruges rent praktisk, og den anden aktivitet udmønter sig i, at nedfælde dette i synopsens research-afsnit herunder. Slutteligt vil jeg som det sidste led i

[1] https://jsoup.org/download

forløbet, besvare problemformuleringen i konklusionen - kortfattet. Samt vil jeg reflektere over projektets proces.

Jeg har afsat alle ugens alm. arbejdsdage på at afvikle første aktivitet og efterfølgende bruge hhv. fredag, lørdag og søndag til synopsisskrivning (anden aktivitet). Jeg vil skiftende bevæge mig mellem hver fase. Som forløbet skrider frem, vil researchen derfor være mere og mere omfattende samt vil der være fyld i synopsen.

Rent prioriteringsmæssigt tilstræber jeg mig først at afdække området indenfor JSOUPs GET-metoder for efterfølgende at gå i gang med POST. Disse underemner forklares i dyber detaljegrad længere nede i synopsen.

Endvidere vil jeg under min proces anvende et åbent GitHub-repository[2] til at uploade mine eksperimenter, da det giver mig mere mobilitet og fleksibilitet ift. hvor jeg ønsker at arbejde (via stationær eller laptop) - ligeledes giver dette en bedre gennemsigtighed, såfremt min vejleder, eller en helt 3. part, skulle ønske en let og gennemskuelig indsigt i min proces. Samt giver dette en ganske udmærket backup-løsning for mig selv, såfremt uheldet skulle være ude (GitHub tilbyder indbygget rollback-funktion og versionsstyring).

Ugentlig tidsplan

Herunder fremgår den ugentlige tidsplan for hele projektets forløb.

	Mandag	**Tirsdag**	**Onsdag**	**Torsdag**	**Fredag**	**Lørdag**	**Søndag**
Uge 18	Udvikle eksperimenter/researche JSOUP. Herunder hvordan data udtrækkes med GET.				Indskrive research i synopsis.		
Uge 19	Udvikle eksperimenter/researche JSOUP. Herunder hvordan data udtrækkes med POST.				Indskrive research i synopsis.		
Uge 20	Udvikle eksperimenter/researche JSOUP. Herunder hvordan data udtrækkes med POST.				Indskrive research i synopsis.		
Uge 21	Indskrive research i synopsen.			Lægge sidste hånd på synopsen.	Deadline kl. 11:00 - aflevere synopsis.		

Research

Indledning

I følgende afsnit vil jeg klarlægge den research som jeg har gjort mig i løbet af projektets forløb. Heraf vil jeg nedfælde den proces og viden som værende det bærende element i denne synopsis. Jeg vil i løbet af afsnittet foruden, at dokumentere processen, ligeledes besvare mine underspørgsmål i problemformuleringen. Selve hovedspørgsmålet lader jeg gemme til den mundtlige præsentation. Min research er baseret på JSOUP version 1.9.2.

JSOUP grundlæggende[3]

Rent indledningsvis kan jeg kort berette at JSOUP helt grundlæggende set fungerer som en HTML-parser. I sin egentlige forstand betyder dette, at JSOUP tilbyder et bekvemt Java API, der muliggør det, at kunne indhente

[2] https://github.com/danielwinther/4.-semester
[3] https://jsoup.org/

og manipulere data i HTML-format. For at sortere i denne datamængde (af HTML) som måtte komme fra en given kilde (heraf bl.a. hjemmesider), anvender JSOUP CSS-selectors til dette formål.

Den begrænsning som dog ligger i JSOUP er, at biblioteket i alt sin enkelthed blot er en parser og dermed ikke består af en indlejret browser-engine (en browser-engine er et program der blot renderer HTML og omsætter dette til noget visuelt i browseren). Dette har i samtlige af mine eksperimenter vist sig at være et gentagende problem, da mange hjemmesider oftest afvikler kompleks JavaScript-kode, der er altafgørende for sidens virkning og præsentation af data. JSOUP er komplet uvidende om det indhold der tilføjes til DOM'en af JavaScript efter den første sideindlæsning. Al den data der tilføjes af JavaScript er derfor utilgængelig for JSOUP og kan derfor ikke scrapes.

Hvorom alting er, har det ligeledes sine fordele. JSOUP er bl.a. relativ hurtig i sine udtræk, hvorimod et større framework/kodebibliotek (med dertilhørende indbygget browser-engine) nok ville tage længere tid om, at indhente selvsamme data. I mange tilfælde kan denne JavaScript/AJAX-kode dog spores og omgås med lethed ved at analysere de HTTP-pakker der sendes afsted, når hjemmesiden besøges - eksempelvis ved at anvende HTTP-debuggeren Fiddler. Fiddler giver indsigt i, hvilke URL'er der måtte blive kaldt, når en side besøges. Er der eksempelvis implementeret en autocomplete-funktion i et søgefelt, sker dataudtrækket som regel via AJAX. JavaScript-koden modtager data fra en ellers skjult URL (oftest i JSON-format).

Indlæsning af HTML fra en webadresse via JSOUP gøres ved hjælp af Document-klassen. Først initialiseres et nyt objekt af ovenstående klasse, hvorefter metoden Jsoup.connect kaldes. Connect-metoden tager et argument; URL'en til den pågældende webadresse. Det er ligeledes væsentligt at tage stilling til, hvilken slags HTTP forespørgsel der er tale om. JSOUP skelner kun skelner mellem GET og POST. PUT, DELETE mv. kan sagtens implementeres, men kræver mere benarbejde. Herunder ses et kodeeksempel der foretager en GET-forespørgsel og gemmer efterfølgende indholdet (HTML'en) i en variabel.

```
Document dokument = Jsoup.connect("http://www.anbo-easj.dk/").get();
```

Ydermere er parseren i stand til at indlæse selv ulovlig HTML - HTML der ikke følger den regelmæssige konvention og standard. Dvs. parseren trods tags, der ikke er lukkede korrekt eller sågar opdigtede, stadig kan indlæses, forstås og sorteres i. Dette kan helt bestemt have sine praktiske egenskaber på større enterprise hjemmesider, der ikke altid følger de standarder, der er sat særligt nøje, og som ofte er udviklet med et større JavaScript-framework for øje. Disse frameworks implementerer ofte sine egne HTML-attributter og lign. Dette skaber tilsyneladende ikke voldsomme problemer.

Som det er med så meget andet netværksprogrammering i Android, skal JSOUPs API ligeledes afvikles asynkront ved at arve fra AsyncTask-klassen[4]. Dette vælger jeg bevidst ikke at gå i nærmere detaljer med, da dette må antages at være et relativ basalt emne, som allerede er omfattet af dette fags pensum.

Scraping versus webservices

Normalt når der indhentes data, udfører klienten som oftest et eller flere kald til en webservice. Webservicen foretager derefter de nødvendige bagvedliggende databasekald og returnerer dernæst resultateti enten JSON- eller XML-format, som klienten da kan præsentere i sin applikation. Denne metode at udtrække data på er ligeledes at foretrække, da dataene leveres i en konsistent og logisk struktur. På samme måde er man som udvikler også sikret, at de data man ønsker at udtrække rent faktisk er tilgængeliggjort fra virksomhedens side

[4] https://developer.android.com/reference/android/os/AsyncTask.html

af, hvorimod scraping, rent juridisk, befinder sig i en gråzone på det punkt, da firmaet ikke selv har udviklet et API til dataudtræk og derfor ikke givet deres samtykke til at dele deres data. Dette kan skabe stridigheder og uoverensstemmelser mellem virksomhed og scraper og har i tidens løb også ledt til et utal af retssager - alle med vidt forskellige udfald.

Den primære fordel, der derimod ligger til grund for at scrape er, at man har muligheden for, at udtrække data, der almindeligvis ikke er tilgængelige gennem f.eks. et RESTful API. Den væsentligste ulempe ved at scrape er, at dataudtrækkene er skrøbelige i den forstand, at foretages der bare de mindste ændringer i HTML-koden på den scrapde hjemmeside, vil den dertil programmerede scraper kaste en exception, da strukturen scraperen oprindeligt er programmeret op imod pludselig har ændret sig. Herunder vil jeg beskrive en mindre case, som jeg selv har observeret.

Som led i mit eksperimentelle arbejde, har jeg erfaret at Facebooks API har fjernet muligheden for at indhente sin egen venneliste fra API versionsnummer 2.0 og opefter. Dette problem løste jeg ved at logge ind på min konto rent programmatisk og netop scrape min egen venneliste og få dataene præsenteret pænt, som jeg selv ønsker. Det er netop den styrke der ligger i scraping, og det er der mulighederne næsten er uanede.

Anvendelsesmuligheder[5] [6] [7]

Selektion af data

JSOUPs mange anvendelsesmuligheder gør, at data kan udtrækkes på en relativ nem og enkel måde. Herunder vil jeg gennemgå nogle af de mest essentielle metoder og teknikker.

Når den pågældende hjemmesides HTML er gemt i en variabel, kan der ved forskellige fremgangsmåder sorteres i denne data. Primært gøres dette ved, at udvælge de ønskede data via enten HTML'ens tag, id, klasse eller attribut.

```
Element tag = dokument.getElementsByTag("strong").first();
Element id = dokument.getElementById("header");
Element klasse = dokument.getElementsByClass("center").first();
Element attribut = dokument.getElementsByAttribute("href").first();
```

```
String tagString = tag.text();
String idString = id.html();
String klasseString = klasse.tagName();
String attributString = attribut.attr("abs:href");
```

Efterfølgende skal det udvalgte element behandles ved at specificere, i hvilken form dataene skal præsenteres. Nogle af de mest fundamentale måder at præsentere disse på, er ved at indhente selve elementets tekststreng, HTML-koden i sig selv eller en given attributs værdi (f.eks. et a-tags href-attribut eller src-attributten i et img-tag).

En alternativ måde at udvælge sin data på er ved at anvende JSOUPs indbyggede select-metode. Denne metode targer et argument i form af en CSS-selector. Denne metode er om noget nok den mest præcise, da man med stor nøjagtighed kan præcisere, hvilke data man ønsker at trække ud. Herunder ses et eksempel på dette. Eksemplet kalder first-metoden, da jeg ikke ønsker mere end blot et enkelt element udtrukket.

[5] https://jsoup.org/cookbook/extracting-data/dom-navigation
[6] https://jsoup.org/cookbook/extracting-data/selector-syntax
[7] https://jsoup.org/cookbook/extracting-data/attributes-text-html

```
Element select = dokument.select("body > div.center > strong > span").first();
```

Forekommer der mere end ét element, der skal præsenteres i applikationen, kan dette løses ved at bruge Elements-klassen for derefter at løbe hver enkelte element igennem med en foreach-løkke. Elements, i modsætning til Element, returnerer en liste af elementer, frem for for kun et enkelt.

Dette er praktisk såfremt der er tale om gentagne data. F.eks. et nyhedssite hvor der forekommer mere end blot én nyhedsartikel. Nedenstående eksempel demonstrerer et udtræk af af alle tr-tags. Dataene gemmes i elementer-objektet og løbes derefter igennem med en løkke. Løkken udskriver hver elements respektive tekststreng ved brug af text().

```
Elements elementer = dokument.select("body > table > tbody > tr");
for (Element element : elementer) {
    Log.d("debug", element.text());
}
```

Indlæsning af billeder [8]

Hvad angår visning af billeder gennem et ImageView, understøtter native Android ikke muligheden for, at vise billeder fra en absolut webadresse. Billedet skal som udgangspunkt gemmes lokalt i applikationens mappestruktur - oftest under drawable-mappen. Derfra kan der foretages referencer, når et billede skal vises. Når det kommer til scraping af data, vil billedstien altid være absolut og derfor pege på en bestemt webadresse, hvor billedet er lagret.

Den åbenlyse måde at løse dette problem på vil sædvanligvis være at downloade billedet lokalt, rent programmatisk, og derefter tilgå det og få det vist i et ImageView.

Lejlighedsvist har jeg dog fundet et velfungerende kodebibliotek, der kan vise billederne - uden at være nødsaget til at gemme dem lokalt i selve applikationen. Dette bibliotek hedder Picasso og er letanvendeligt. Herunder ses et eksempel. Load-metoden tager et argument; en URL til et givent billede. Billedet indlæses efterfølgende til et ImageView, som vises ved applikationens opstart. Jeg vælger ikke, at komme Picasso-biblioteket nærmere, da denne synopsis kun har JSOUPs API for øje.

```
Picasso.with(getApplicationContext()).load("http://www.anbo-easj.dk/cv/andersBorjesson.jpg").into(imageView);
```

POST-data[9]

JSOUP tilbyder desuden et glimrende API til, at afvikle POST-data, som oftest udføres gennem en HTML-form. Dette kan da udføres programmatisk og har en væsentlig betydning for at tilgå data, som normalvis er session-beskyttet af f.eks. login-funktionalitet. Som angivet i screenshottet til højre, tillader API'et ydermere at kunne tilføre forespørgslen forskellige POST-værdier, som sendes videre til den angivne URL, når POST-metoden kaldes slutteligt. Værdierne tilføjes med data-metoden via et Map/Dictionary-

```
Document login = Jsoup.connect("URL")
        .data("user", brugernavn)
        .data("pass", password)
        .post();
```

[8] http://square.github.io/picasso/

[9] https://jsoup.org/cookbook/input/load-document-from-url

lignende API; Først indtastes navnet på POST-værdien (denne værdi kan spores i hjemmesidens HTML-kode) og efterfølgende selve værdien.

Med udgangspunkt i screenshottet ovenover; Såfremt der er angivet et filnavn i HTML-formens action-attribut, vil denne fils indhold nu blive returneret, hvis brugernavn samt password vel og mærke er korrekt indtastet og dermed matcher den data, der er lagret i databasen. Dette er i nogle tilfælde udmærket til formålet, men i mange andre situationer, vil et serverside-sprog ofte omdirigere brugeren til en helt ny underside. Problemet opstår, da denne underside ikke er i stand til at huske de POST-værdier, der netop er blevet angivet i programmets kode. Samt er den session som akkurat er blevet oprettet nu glemt og ubrugelig. Programmet bliver nu automatisk omdirigeret til login-siden igen med besked om, at indtaste brugernavn og password, og de session-beskyttede data kan derfor ikke scrapes - sessionen er som sagt glemt.

Lagring af cookies/SESSIONID

For at komme dette problem til livs, tillader JSOUPs API, at kunne lagre de cookies, der måtte medfølge ved et evt. korrekt login-forsøg. Dette betyder ligeledes, at det SESSIONID, som er nødvendigt for at tilgå de beskyttede data, nu kan gemmes - og kaldes alle de steder, der er behov for at omgå en beskyttets side data.

```
Connection.Response response = Jsoup.connect(URL + "/login.php")
        .method(Connection.Method.POST)
        .data("email", email)
        .data("pass", password)
        .data("_fb_noscript", "true")
        .execute();
```

Dette kan gøres ved først at etablere en POST-forbindelse ved at indtaste de korrekte login-oplysninger. Er loginnet succesfuldt, vil forbindelsen blive gemt i en variabel som vist på screenshottet ovenover.

Forbindelsens cookies kan senere genkaldes ved lejlighed blot ved at kalde cookies-metoden. Denne metode tager ét argument i form af POST-forbindelsens cookie-data. Eksemplet ovenover og nedenfor demonstrerer netop dette med stor tydelighed: Screenshottet foroven opretter først POST-forbindelsen succesfuldt til min egen Facebook-profil ved at logge mig ind med dertilhørende e-mail og password. Senere anvendes selvsamme forbindelse til at omgå den ellers session-beskyttede side. Denne side viser en liste af alle mine nuværende venner, som der nu er rig mulighed for at scrape.

```
Document document = Jsoup.connect(URL + "/danieldk1992/friends?startindex=0")
        .cookies(response.cookies())
        .get();
```

Det er væsentligt at pointere, at visse sider implementerer en noget mere kompleks form for sikkerhed - bl.a. med beskyttelse af HTTPS-protokollen samt andre sikkerhedsforanstaltninger såsom forskellige tokens, der sendes frem og tilbage. Fronter er blot ét eksempel af mange. Det er bestemt ikke en umulighed at scrape disse sider, men det kræver dog et større detektivarbejde, for at opspore de rette værdier og få dem sendt videre med JSOUP, så sessionen kan opretholdes. De fleste sider er dog (heldigvis) ret lemfældige hvad angår sikkerheden og derfor er det sjældent et nævneværdigt problem der opstår særlig ofte. Som nævnt tidligere, er den eneste forhindring JSOUP ikke kan løse kun meget JavaScript-tunge sider.

Modificere elementer[10] [11]

Ønskes der at modificere i den udtrukne HTML, kan dette gøres ved, at angive et argument i f.eks. text-metoden. Herunder vises et eksempel med dette - blot ændres her value-attributten på et input-felt til den tekststreng variablen indeholder. Denne handling bliver udført på det input-felt der har id'et "s" i kildekoden. Dette kan have sine praktiske egenskaber ved eksempelvis generering af HTML-filer.

```
document.select("#s").first().val(soegord);
```

Alternativer til JSOUP

Herunder vil jeg uddybe hvilke alternativer, der er til JSOUP. Såfremt JSOUPs udvikler i fremtiden opgiver at videreudvikle kodebibliotektet, kan disse løsningsforslag implementeres i stedet.

Pattern og Matcher[12] [13]

Det mest umiddelbare alternativ til at scrape data, er ved at anvende regular expression til at udvælge dataene. Java implementerer dette igennem Pattern og Matcher-klassen. Regular expressions er ligeledes også scraping i sin mest primitive form, og er også sværere at arbejde med - og tager derfor også længere tid at implementere som en helstøbt løsning. Ydermere kan regular expressions ikke håndtere komplet HTML parsing, eftersom det afhænger af, at kunne matche åben- og lukke-tagget i HTML-koden - hvilket ikke er muligt. En parser vil være at foretrække til dette, da parseren selv aflæser hvor det HTML-tagget stopper og starter.

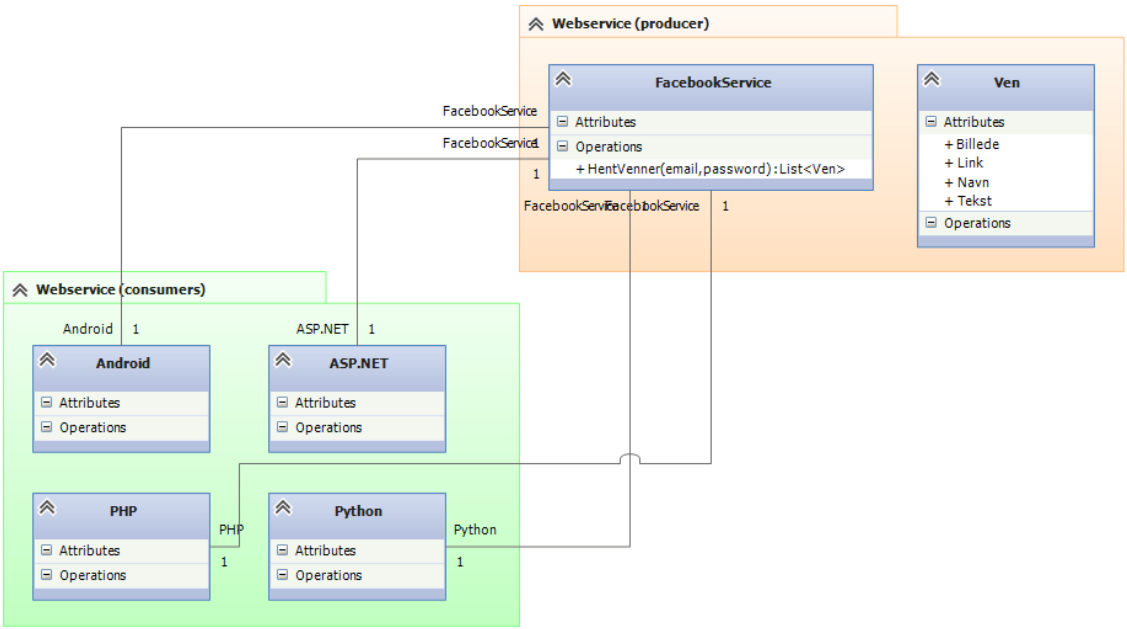
WebView[14]

En helt anden mulighed er, at udnytte Androids WebView. WebView'et kan bruges til, at fremvise hjemmesiden og efterfølgende kan HTML-koden indhentes programmatisk. Denne løsning er udmærket til præsentere data, da viewet sågar er i stand til at rendere elementer, der indlæses efter DOM'en. Ulempen er dog, at der ikke medfølger en HTML-parser, da WebView oprindeligt kun er ment til visning af hjemmesider. Ydermere er WebView væsentligt langsommere end JSOUP.

Webservice

En tredje løsning kunne være, at udvikle en webservice som kontaktpunkt der scraper dataene. Dataene udbydes senere til de forskellige systemer (bl.a. Android), der skal præsentere dem. På denne måde opnås der konsistens i hvilke data der bliver vist. Ligeledes undgår man, at skulle reprogrammere scraperen for hvert system der udvikles. Diagrammet herunder viser hvordan – med udgangspunkt i at scrape og udbyde Facebooks data.

[10] https://jsoup.org/cookbook/modifying-data/set-attributes
[11] https://jsoup.org/cookbook/modifying-data/set-text
[12] https://developer.android.com/reference/java/util/regex/Pattern.html
[13] https://developer.android.com/reference/java/util/regex/Matcher.html
[14] https://developer.android.com/reference/android/webkit/WebView.html



Andre kodebibliotekter

Der findes et utal af forskellige kodebiblioteker skrevet i Java, der alle kan parse HTML. Min research viser desværre, at de fleste af dem ikke er fungerende til Android. Det eneste bibliotek, der fortsat opdateres og samtidig er kompatibel med Android er JSOUP.

Konklusion

Indledningsvis vil jeg her besvare problemformuleringen kortfattet. Som nævnt tidligere besvarer jeg selve hovedspørgsmålet til den mundtlige eksamen.

- *Hvad er JSOUP?*
  JSOUP er et Java-baseret kodebibliotek, der er i stand til at parse HTML fra en given kilde. Parseren kan selektere data på forskellig vis.

  JSOUP bruger Document-klassen til at indlæse HTML'en, hvorefter enten POST eller GET-metoden kan kaldes derved. JSOUP anvender CSS-selectors til at udvælge dataene. Endvidere afvikles alle forbindelser asynkront. Parseren er i øvrigt i stand til at indlæse selv ulovligt HTML.

- *Hvorfor anvende JSOUP til at hente data fra en given kilde?*
  JSOUP tillader dataudtræk i de tilfælde, der ikke eksisterer en webservice. JSOUP fejler dog, hvis HTML-koden ændrer sig på hjemmesiden – hvad den sandsynligvis gør før eller siden.

- *Hvilke begrænsninger forekommer der ved benyttelse af JSOUP?*
  JSOUP kan ikke parse elementer, der tilføjes til DOM'en efter indlæsning af kilden. Dette omfatter hovedsageligt JavaScript-kode. Udover det er begrænsningerne meget få med JSOUP.

- *Hvilke muligheder forekommes der ved benyttelse af JSOUP?*
  JSOUP giver mulighed for at selektere data ud fra id, klasse, tag og attribut i HTML-koden. Alternativt kan data udvælges med CSS-selectors, som giver mere præcision ved selektering af data. Efterfølgende kan der præciseres i hvilket format dataene skal vises – herom som alm. tekststreng, HTML-kode eller noget helt tredje.

  Endvidere tillader JSOUP også POST actions, således at man omgår ellers session-beskyttede sider. JSOUP kan desuden opretholde den sessionm, der bliver oprettet ved et login og genkalde den ved senere brug. Dette gøres ved at lagre de cookies, der sendes afsted.

  JSOUP er desuden i stand til, at foretage ændringer i HTML-koden. Dette kan evt. bruges til at generere nye HTML-filer, hvor indholdet er blevet tilpasset programmatisk.

- *Hvilke alternativer til JSOUP eksisterer der?*
  De alternativer der findes er meget få. Der kan anvendes regular expressions (ved brug af Pattern og Matcher-klassen) til at udvælge dataene. Dette har sin ulempe i og med, at regular expression ikke kan aflæse, hvornår HTML-tagget åbner og lukker. Endvidere er regular expressions et sværere API at arbejde med, og kræver ligeledes flere linjers kode at få til virke, sammenlignet med f.eks. JSOUPs select-metode, der blot er 1 linje.

  Sekundært er det muligt at indlæse HTML-koden ved hjælp af et WebView. HTML'en kan efter indlæsning hentes rent programmatisk. Fordelen ligger i, at WebView også afvikler JavaScript, da viewet fungerer som en almen browser. Dog tager scrapingen længere tid, da WebView generelt set er langsommere end JSOUP.

  En tredje løsning er at anvende en webservice som knudepunkt for scrapingen. Webservicen scraper dataene og udbyder disse til de forskellige systemer der konsumere servicen og præsenterer dens data.

Refleksion

I retrospekt har min research helt bestemt være en meget lærerig proces. Jeg har igennem mit specialiseringsforløb fået et rigtig godt indblik i, hvordan JSOUP fungerer på et praktisk plan. Et af mine personlige store gennembrud har været, hvordan jeg kan scrape session-beskyttet data - noget som jeg har haft stor besvær med tidligere i sprog som C#. At lære at arbejde med dette API kan jeg uden tvivl bruge i andre programmeringssprog i et lign. kodebibliotek.

Planlægningsmæssigt har jeg fulgt min tidsplan ret nøje. Jeg har har på intet tidspunkt været betydeligt bagefter eller foran planen, og har samtidig afsat en tilpas mængde tid til, at skrive om researchen i synopsen i weekenderne. Mit fravalg af en bestemt systemudviklingsmetode har i løbet af processen også givet god mening. Min påstand vil være, at havde jeg valgt at arbejde med f.eks. SCRUM som metode, ville det have givet mig en masse unødvendigt ekstraarbejde, som ikke giver pote i et research-projekt. Afslutningsvis har det været fornuftigt nok, at afholde sig fra en produktorienteret metode.

GitHub viste sig også at være et fornuftigt tilvalg, da jeg af flere omgange har skiftet min arbejdsplads fra stationær til bærbar computer. At kunne indhente gårsdagens arbejde fra et repository, har derfor været absolut nødvendigt – og meget mere anvendeligt ift. at placere filerne i en Dropbox-mappe.

Daniel Winther Jensen

Erhvervsakademi Sjælland - Campus Roskilde

Datamatiker – 4. semester

Litteraturliste

jsoup: Java HTML Parser. *Udgivet af Jonathan Hedley. Internetadresse: https://jsoup.org/ - Besøgt d. 27.05.2016 (Internet)*

- *Officiel hjemmeside samt dokumentation af JSOUP.*

Jsoup Tutorial. *Udgivet af javapoint. Internetadresse: http://www.javatpoint.com/jsoup-tutorial - Besøgt d. 27.05.2016 (Internet)*

- *Relevant læsevejledning til brugen af JSOUP.*

Jsoup Tutorial and Examples. *Udgivet af Lokesh Gupta. Internetadresse: http://howtodoinjava.com/jsoup/complete-jsoup-tutorial/ - Besøgt d. 27.05.2016 (Internet)*

- *Relevant læsevejledning til brugen af JSOUP.*

Android Basic JSOUP Tutorial. *Udgivet af AndroidBegin. Internetadresse: http://www.androidbegin.com/tutorial/android-basic-jsoup-tutorial/ - Besøgt d. 27.05.2016 (Internet)*

- *Relevant læse- og videovejledning til brugen af JSOUP.*

Picasso. *Udgivet af Square. Internetadresse: http://square.github.io/picasso/ - Besøgt d. 27.05.2016 (Internet)*

- *Officiel dokumentation til Picasso - indlæsning af billeder.*