Wearep

En app som kan twittra och visa väderprognosen för en plats i html och i en tweet

SYSTEMINTEGRATÖR YHSIPI17/DATAKOMMUNIKATION OCH NÄTVERK

den 8 oktober 2018
Billy Andersson, Malin Albinsson, Sten Karlsson

WEAREP

En app som kan twittra och visa väderprognosen för en plats i html och i en tweet

1 INNEHÅLL

2 INLEDNING

2.1 SAMMANFATTNING

Wearep är en app för att rapportera väderdata för en utvald plats i html och om användaren så önskar även på twitter. Appen går också att använda för att twittra om annat. Vi valde en GNU GPL-licens därför att vi gjort projektet för att lära oss och genom att vi delar med oss av vår kod kan även andra lära sig av vad vi gjort.

2.2 INLEDNING/SYFTE

Vår uppgift var att leverera ett projekt som skulle skrivas i Java och ha någon form av koppling till webb, exempelvis interaktion med Twitters eller Facebooks APIer. JSON eller XML skulle ingå i flödet för programmet, alternativt att hämta en hemsida och med hjälp av en DOM-parser hämta ut information från den, det vill säga, projektet skulle visa att vi kan hämta information från JSON, XML eller DOM-struktur i HTML.

Programmet behövde inte vara supersnyggt. Grafiskt eller kommandoradsbaserat spelade ingen roll. Det var tillåtet att använda andra komponenter som databaser eller javabibliotek för att lösa uppgiften.

Huvuduppgiften var att visa att vi kan hantera nätverksanslutningar från Java, med fokus på webben. Vi skulle också beskriva hur vi analyserat säkerhetsaspekten i det program vi skrivit, vad vi tänkt på och hur vi löst det. Källkoden skulle versionshanteras med git.

Vi valde att bygga en app som kan som kan logga in på twitter, hämta och posta tweets, samt hämta och publicera väderdata för en utvald plats både som html och i en tweet.

2.3 DEFINITIONER OCH FÖRKORTNINGAR

Förklaringar till några av de begrepp och förkortningar som förekommer i rapporten ges i Tabell 1.

Tabell 1 Definitioner och förkortningar

Definition/Förkortning	**Förklaring**
API	Application Programming Interface (applikationsprogrammeringsgränssnitt)
fxml (JavaFx)	Layouten för en sida
GUI	Graphical User Interface (grafiskt användargränssnitt)
HTML	HyperText Markup Language, ett märkspråk för hypertext som beskriver och definierar innehållet på en websida
HTTPS	Hypertext Transfer Protocol Secure, ett protokoll för krypterad transport av data för HTTP-protokollet
JSON	JavaScript Object Notation, ett textbaserat, kompakt format för datautbyte utbyta data
nod (Java Fx)	Ett element på en sida
OAuth	Open Authentication, se sektion 4.7.1
regex	Regular expression (reguljärt uttryck)
scene (JavaFX)	En sida
TDD	Test Driven Development (testdriven utveckling)
UTC	Coordinated Universal Time
XML	Extensible Markup Language, ett universellt och utbyggbart textbaserat märkspråk

2.4 Referenser

Länk till git-repository:
https://github.com/anderssonbilly/uppgBMS

Länk till Trello-tavla:
https://trello.com/b/dN3ze08J/wearep#

gson:
https://github.com/google/gson

JavaFX-dokumentation:
https://www.oracle.com/technetwork/java/javase/documentation/javafx-docs-2159875.html

Google Geocoding API-dokumentation:
https://developers.google.com/maps/documentation/geocoding/intro

http-protokollet:

https://tools.ietf.org/html/rfc7230

OAuth rtc protokoll:
https://tools.ietf.org/html/rfc8252

Twitters API-dokumentation:
https://developer.twitter.com/en/docs.html

Twitter4J biblioteket:
http://twitter4j.org/en/index.html

SMHIs API-dokumentation:
https://opendata.smhi.se/apidocs/metfcst/index.html

3 Genomförande

3.1 Övergripande mål

Det övergripande målet var att skapa ett program som tar emot inmatning av en plats av användaren, slår upp koordinaterna för platsen, hämtar väderleken för den och sedan låter användaren välja om han/hon även vill twittra väderprognosen. Programmet skulle även kunna twittra annat.

3.2 Avgränsningar

Då vi hade väldigt begränsad tid kvar på kursen fick vi göra större avgränsningar än vi kanske annars hade valt. Funktionsmässigt kan man exempelvis inte välja vilka väderdata man vill twittra ut, utan variablerna är förvalda. Tiderna i väderdelen presenteras i UTC istället för i lokal tid som kanske varit det mer logiska. I Twitterdelen kan man inte se andras flöden eller följa ett enskilt tweets historik. TDD testnings-coverage är inte alls så god som man skulle önska. Exceptionshanteringen är inte heller ordentligt implementerad, åtminstone inte i Väder-delen. Valideringen av indata är otillräcklig då vi inte hann implementera den delen ordentligt.

Arbetsmetodik

Vi använde ett repository på github, https://github.com/anderssonbilly/uppgBMS, där vi alla hade skrivrättigheter. Vi förde det mesta av diskussionerna via Slack. Trello fick fungera som visualiseringsverktyg över

projektets status och vi använde det både som en KANBAN-tavla med kolumner för TODO, In progress, Ready for review och Done, och för att samla userstories och följa upp frågor och bolla över uppgifter till varandra.

Initialt fördelade vi arbetet så att vi arbetade med varsitt API; Billy med Twitter, Sten med Google Geocoding (Google Maps) och Malin med SMHI. Därutöver arbetade vi med review av varandras kod och hjälpte varandra när det behövdes. Varje API fick varsin branch (även om de fick namn efter oss), JavaFX-skalet fick en egen och därutöver hade vi ett par olika test-brancher för att kunna testa att de olika delarna fungerade tillsammans innan vi släppte dem till master.

4 WEAREP-APPEN

4.1 ÖVERSIKT

I appen Wearep kan användaren logga in på Twitter, hämta och posta tweets. Appen kan också, för en av användaren angiven plats, hämta och publicera väderdata från SMHI både som html och i en tweet. Platsens koordinater hämtas från Google Geocoding. Vi har således interagerat med tre olika API:er.

Appen är uppbyggd av olika moduler:

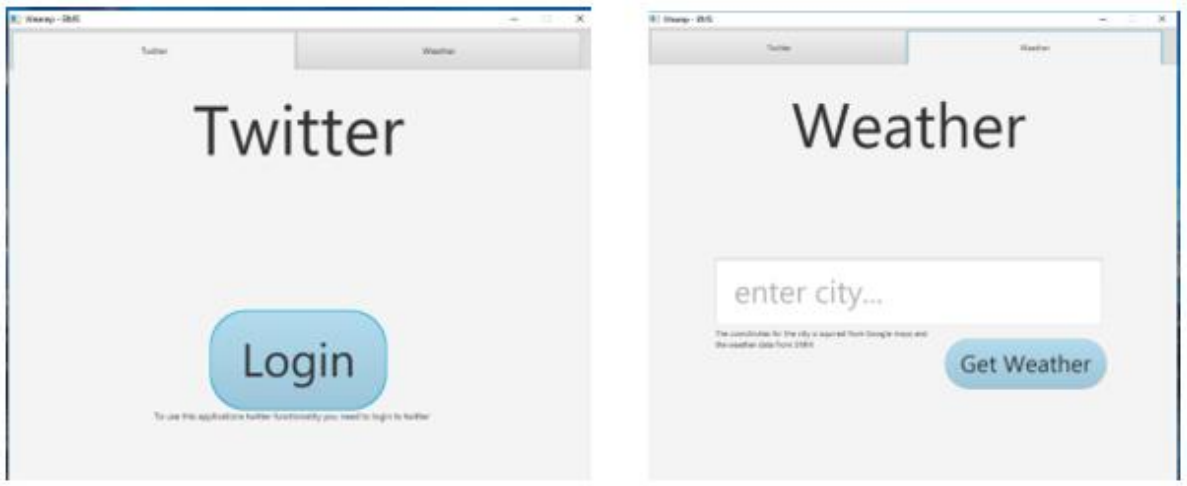
- Geokodning - Hämtning av koordinater för en plats
- Väder - Hämtning av väderdata och ihoppackning av data till JSON-objekt
- Twitter - Inloggning, hämta och posta tweets
- Dataout - Publicerar väderdata som html, producerar en textsträng som kan twittras ut

4.1.1 Användarinterface

När man startar Weareap-appen får man upp två flikar, se Figur 1.

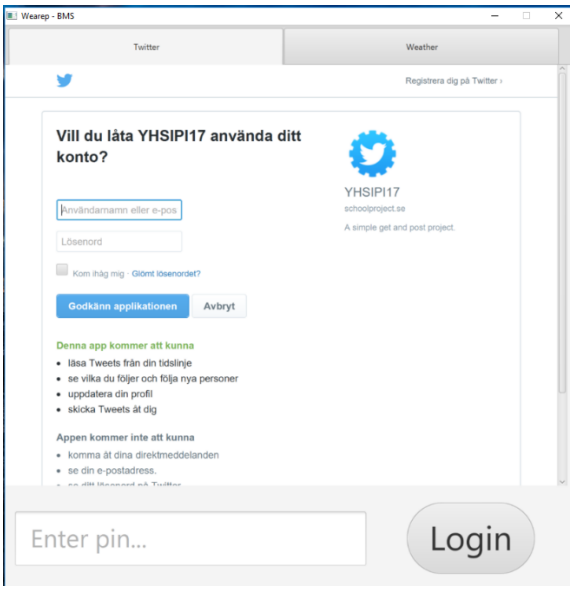
- en Twitterflik (inloggning krävs)
- en Weatherflik

Appen har också en inbyggd webbrowser som websidan öppnas i när väderprognosen är färdig.



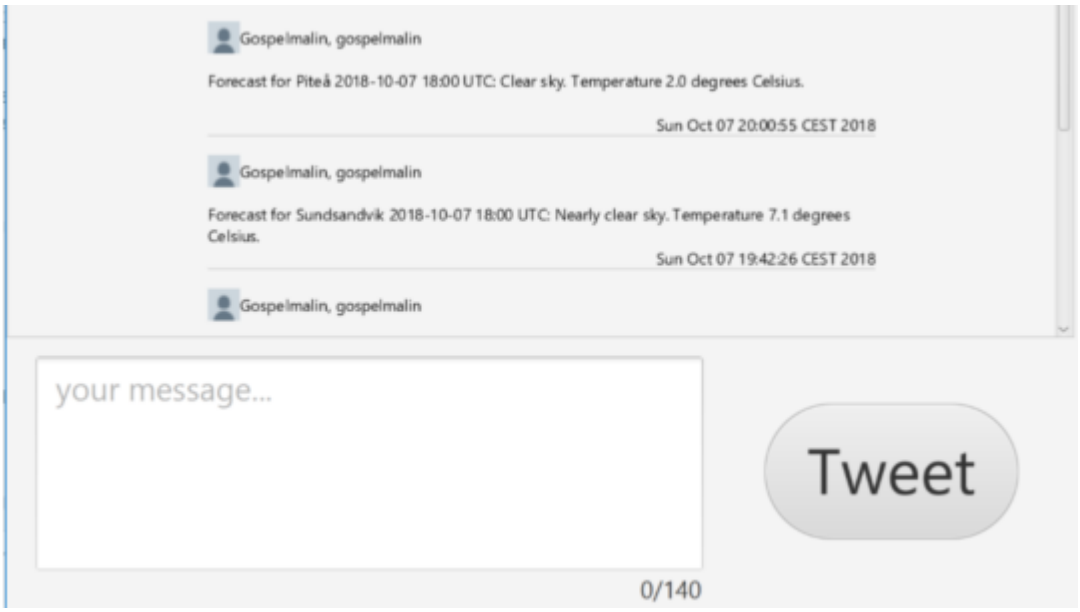
Figur 1 Twitter-fliken (inloggning) respektive Weather-fliken

Om man bara är intresserad av att få en väderprognos kan man gå direkt till Weather-fliken.Vill man även ha möjlighet att twittra vädret måste man först logga in till sitt Twitter-konto genom att följa instruktionerna i appen, se Figur 2.



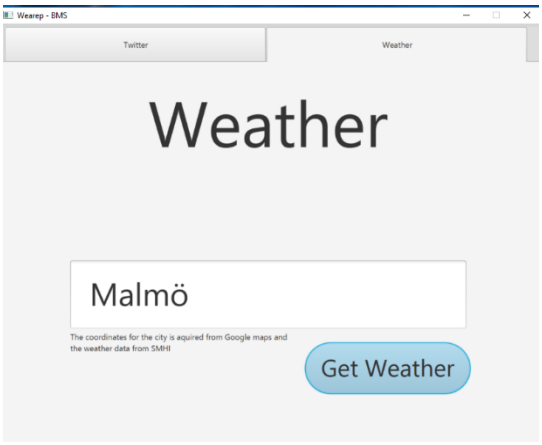
Figur 2 Twitter-fliken, inloggning

Väl inloggad kan man läsa tweetsen i sitt flöde eller posta en ny tweet, se Figur 3.



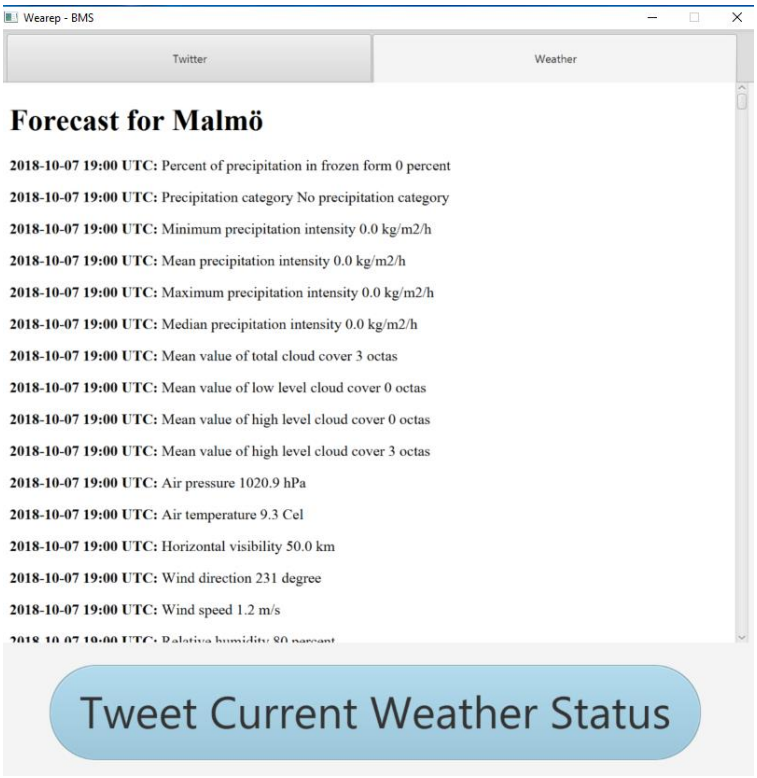
Figur 3 Twitter-fliken, twitterflödet

Väljer man istället Weather-fliken har man möjlighet att skriva in den plats man önskar en väderprognos för, se Figur 4.



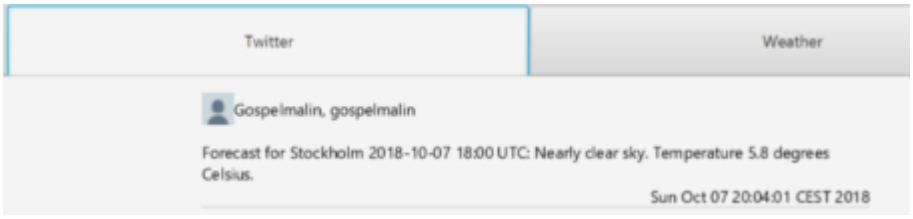
Figur 4 Weather-fliken, välj plats för prognosen

Ett klick på Get Weather, och väderprognosen för de närmaste 10 dagarna skrivs ut i webbläsaren, Figur 5.



Figur 5 Väderprognosen i html

Är man inloggad på twitter kan man även twittra väderprognosen för nästkommande timme för sin utvalda plats genom att klicka på Tweet Current Weather Status. Och Figur 6 visar exempel på en väder-tweet.



Figur 6 Väder-tweet för Stockholm

4.2 Platsdata från Google geocoding API

Platsdata hämtas från Google geocoding API via klassen GetCoords. Klassen har en run(String city) metod som först bygger upp en URL med staden och en API-nyckel.

Med en färdig URL så upprättas en HttpURLConnection som hämtar JSON datan till en sträng, som sedan omvandlas till JSON objekt och arrayer för att komma åt datan man vill ha, och sen rundas koordinaterna av till 6 decimaler.

Efter det kan man hämta koordinaterna med getLatitude() och getLongitude(). Man kan även hämta ut stadens namn med getCity().

4.2.1 Google geocoding API

Googles geocoding API tar emot några argument i URLen, som exempelvis vilken stad man är intresserad av, vad man har för API-nyckel, vilket format man vill ha datan i (JSON / XML) och sensor (true/false).

4.3 Väderdata från SMHIs API

Väderprognosdata hämtas från SMHIs API i JSON-format. JSON-objektet bryts ner i JSON-arrayer och JSON-Objekt i Weather-klassen så att slutligen enskilda variabler kan avläsas. Klassen levererar en JSON-array som endera utnyttjas av klassen WeatherSelector där förhandsdefinierade variabler för den närmaste timmens prognos plockas ut och packas ihop i ett nytt JSON-objekt, eller används direkt i klassen HtmlPrinter.

4.3.1 SMHIs API

SMHI Open Data Meteorological Forecasts, PMP, omfattar prognosdata för de kommande 10 dagarna och baseras på ett antal prognosmodeller och manuella justeringar. SMHIs API är ett öppet API, tillgängligt för applikationer som vill använda tjänsten för att hämta prognosdata.
Data levereras som JSON och alla tidpunkter är angivna i UTC (Coordinated Universal Time).

Entry point är: https://opendata-download-metfcst.smhi.se och licensvillkor finns att läsa här: http://www.smhi.se/klimatdata/oppna-data/information-om-oppna-data/villkor-for-anvandning-1.30622

Longitud och latitud för den plats man vill ha prognosen för ingår i URL:n som används vid requesten. En GeoJSON-punkt i JSON-resultatet visar longitud och latitud för den grid punkt som är närmast den efterfrågade. Prognosens starttid återfinns under referenceTime och den tid meterologen godkände prognosen anges i approvedTime. Blocket TimeSeries innehåller själva prognosdata, med validTime, parametrar och deras mätvärden och enheter. ValidTime anger när prognosen gäller. Ögonblickstider gäller för validTime, utom när det gäller nederbörsparametrarna, då intervallet börjar vid tidssteget innan. I början av prognosen är intervallen korta, bara 1 timme, men ökar senare alltmer, upp till 12 timmar.

4.4 Twitter-möjlighet från Twitters API

4.4.1 Twitters API

Twitters API tillåter oss att hämta tweets och att posta tweets. Man måste först logga in via API med hjälp av OAuth för att få tillgång till dessa funktioner.

All tweet data som hämtas av API är i json format.

Vi har använt oss av ett externt bibliotek som heter twitter4j för att underlätta användandet av API. Biblioteket har färdiga metoder för twitters alla API funktioner. Detta gör det mycket smidigt att bygga en applikation runt Twitter.

För att hantera twitter så har vi skapat en huvudklass Twitter4J.java. Det är via denna klassen som applikationen kommunicerar med twitter4j-biblioteket och tillhandhåller applikationen twitter-informationen. Huvudklassen

har även två hjälpklasser Authorization.java och Tweet.java. Vi använder oss även av observerstrategidesign mönster för att kunna uppdatera delar av applikationen om vi är autoriserade eller ej.

getAuthURL hämtar en request token, med hjälp av **getRequestToken**-metoden, och skapar webbaddressen för inloggning till twitter. **authorize**-metoden skickar in pinkoden som vi fått när vi loggat in till twitter. Här hämtar vi en accesstoken och ser om autenticationen fungerade. Detta sker i Authorization.java.

tweet är den metoden som med hjälp av Tweet.java skickar ditt meddelande upp till twitter.
getTweets hämtar ditt twitterflöde och ger det i en lista.
isAuthorized kan kallas på för att se om man är autoriserad eller ej.
addObserver, **updateTweetList** och **updateObserver** är en del av observer strategy pattern.

Vi har en abstrakt TwitterObserver klass med metoderna **updateTwitter** och **update** som vi kallar på när vi kör metoden **updateObservers**.

Så fort vi vill ha en klass som skall ha information om vad som sker i Twitter4J klassen så extendar vi dem med TwitterObserverklassen och lägger till dem i en observer-lista i Twitter4J med **addObserver**-metoden.

När någon av **updateTweetList** eller **updateObserver** körs så meddelas alla klasser i vår observer-lista genom de abstrakta metoderna.

4.5 Output

4.5.1 Publicera HTML

Klassen HtmlPrinter omvandlar JSON-objektet från WeatherSelector till en sträng som kan hämtas upp och twittras ut. Klassen bryter även ner JSON-arrayen från Weather till en sträng och skriver den till html.

4.5.2 Meddelanden

Klassen WeatherMessages producerar följande textsträngar:

- Vädermeddelande för Twitter
- Vädervariabelinformation för utskrift till html
- Html-sidans inledning
- Html-sidans avslutning

4.6 Teknisk lösning

4.6.1 JavaFx

Vi har använt oss av JavaFX som GUI till vårt projekt. För att bygga upp våra scenes i applikationen har vi använt oss av fxml-filer och FXMLLoader.

Fxml-filen är upplagd som en kombination av xml och html. Varje fil innehåller element som representerar noder på sidan, dessa går att sätta css-style på precis som i html. I Fxml-dokumentet sätter vi även en controller som binder denna filen till en java-klass. Noderna når vi sedan i vår controller-klass genom de unika id som vi satt på elementet i fxml-dokumentet. Man når dem genom att använda @FXML-annoteringen följt av nodens id.

Applikationen är byggd med en container-scene som innehåller två tabbar för att vi skall kunna byta mellan två aktiva scenes. I dessa tabbar öppnar vi sedan två scenes. Man navigerar därefter inom taggarna till nya scenes.

Den scene som vi visar webb-innehåll i använder sig av JavaFX WebView och WebEngine. Denna scene är även delad i två mindre delar varav den översta är webbläsaren och den nedre en annan scene.

4.6.2 Utvecklingsmiljö och utvecklingsverktyg

Följande miljöer och verktyg användes för realisering, byggande och tester:

Produkt	Ansvar
Eclipse	Programmeringsverktyg (IDE)
JUnit	Enhetstestning av kod
JavaFX Scene Builder 2.0	Byggande av fxml filer

4.6.3 Felhantering

Javas Exceptions används för felhantering i applikationen.

4.6.3.1 Regler

När ett fel inträffar ska problemet i första hand åtgärdas. Om felhanteringsrutinen inte kan åtgärda problemet ska det kastas vidare.

De fel som kastas ska i första hand vara egendefinierade, för att dölja implentationsdetaljer som inte bör framgå i det publika användargränssnittet. På så vis förbättras också systemets underhållsbarhet.

4.6.3.2 Dataformatering

Datum skall formateras till ÅÅÅÅ-MM-DD HH24MM format, t ex 2018-07-12 15:00.

4.7 SÄKERHET

4.7.1 Autenticiering

OAuth är ett protokoll som beskriver hur man säkert tillåter en applikation att ta del av information ifrån en annan tjänst utan att dela med sig av lösenord. För att kunna använda twitter funktionerna i vår applikation så behöver användaren autentisera sig på twitters webbsida. Detta sker via twitters API med hjälp av OAuth.

Vår applikation frågar först twitters API efter request token, i denna token finns information om vem som kallade på den, när den kallades och unika signaturer. Sedan använder vi oauth signaturen och går till twitters webbsida som hanterar inloggning via API, där når vi en login prompt. När man väl loggat in så genereras en unik pinkod. Denna pinkod skriver vi sedan in i vår applikation.

Vi frågar sedan twitters API med vår unika oauth token vi fick tidigare och pinkoden som vi fick ifrån twitter efter en ny access token. Om allt stämmer så är vi inloggad, twitter har autentiserat vår applikation och ger oss tillåtelse att använda deras API.

För att enbart titta på väderprognosen krävs ingen autenticiering, lika lite som det krävs för att titta på väderprognosen på SMHIs egna sidor.

4.7.2 Dataexponering

Anslutningarna görs via HTTPS, men det finns alltid risk att någon tittar på din skärm när du skriver och på så sätt kan snappa upp uppgifterna till ditt Twitter-konto. I och med att vi använder HTTPS-anslutningar så bör det inte gå så bra att "sniffa" sig till uppgifterna.

API-nycklar bör man inte lägga ut publikt, så den till Twitter lämnas in separat, och kan inte hittas på GitHub. Nyckeln till Google Maps var en "try for free" variant där man fick en mängd requests gratis, och om dom tar slut så måste man logga in och godkänna att dom börjar dra pengar från kortet innan nyckeln funkar igen. Den här borde kanske skyddats också, men det kändes som om det inte behövdes i det här fallet.

4.7.3 Validering

Validering av indata är viktigt ur säkerhetssynpunkt då det är en möjlig angreppsväg (dessutom minskar man antalet applikationskrascher, vilket är positivt både för användare och utvecklare). Ett sätt att minska risken att få in skadlig kod är genom att bara tillåta att den förväntade typen och formatet av data matas in. Det kan exempelvis

göras genom användning av regular expressions. Ett annat sätt är att ”escapa” inmatad data för att på så sätt oskadliggöra eventuell skadlig kod.

Då det kan vara klurigt att få till rätt regex-uttryck och tiden var knapp valde vi en enklare lösning för inmatning av platsnamn. Mindre säker, men ändå något som åtminstone gör det lite svårare för en potentiell angripare. Det är helt enkelt en if-sats som, om strängen för stad innehåller "/" eller "&", sätter stad till ””, vilket i sin tur resulterar i att getLatitude/Longitude returnerar 0.000.

4.7.4 Sårbarheter

Det är viktigt att välja en lämplig nivå på visibilitet för klasser och metoder. Det är onödigt att exponera mer kod än nödvändigt. Vi har försökt att hålla så mycket som möjligt private, annars protected och endast låtit de metoder som verkligen behöver vara publika vara det.

Att källkoden är öppen kan vara en sårbarhet. Vem som helst kommer åt den och då är det lättare att hitta svagheterna. Man bör vara noggrann och försiktig med vad man skickar upp till github då alla kan se det. Exempelvis är det inte lämpligt att lägga sina API-nycklar där, se Sektion 4.7.2 för hur vi hanterat det.

Html-filen sparas lokalt. Det kan ses som en sårbarhet, då den som har html-kunskap kan ändra i den, men den är å andra sidan inte publikt publicerad. Att vår applikation raderar filen om den redan existerar när metoden för att hämta väderprognosen inleds kan vara en säkerhetsrisk om någon skulle ändra på vilken fil som tas bort så att det istället försvinner någon väsentlig systemfil.

Via nätverket skulle man kunna tänka sig att någon ändrar urlen för att hämta väderdata till en annan adress som matar ut gigantiska JSON, så att datorn fylls.

En annan sårbarhet är om valideringen av inputs till applikationen inte är god nog, se även Sektion 4.7.3.

Enligt https://maven.apache.org/security.html finns det inte några kända säkerhetsrisker med Maven (4.0.0) som vi använder.

5 Slutsatser/Diskussion

5.1 Appens implementering

All grundläggande funktionalitet som vi tänkt oss är uppfylld:

- Användaren ska kunna mata in en plats i appen.
- Appen ska hämta koordinater för platsen från Google Maps API.
- Appen ska använda de hämtade koordinaterna för att hämta väderdata för platsen från SMHIs API.
- Appen ska publicera väderdata för den utvalda platsen i en html-fil.
- Appen ska också lägga viss väderdata i en textsträng som användaren kan välja att twittra.
- Användaren ska kunna logga in på twitter, läsa och publicera tweets.
- Appen ska hantera data i JSON-format.

Vissa aspekter och funktioner är inte implementerade eller inte implementerade fullt. Det gäller bland annat felhanteringen och valideringen av inkommande data som kunde vara mer heltäckande för att göra applikationen både säkrare och mer stabil. Vi fick också välja bort vissa funktioner, exempelvis kan man i Twitterdelen inte se andras flöden eller följa ett enskilt tweets historik, och i väderdelen kan man inte välja vilka väderdata man vill twittra ut, utan variablerna är förvalda. Säkerhetsaspekterna är analyserade, se Sektion 5.2, men på grund av tidsbristen måhända inte tillräckligt implementerade i den här versionen av appen.

Appen är byggd i moduler och integrationen mellan dem hanteras av några få klasser. Detta underlättar om vi skulle vilja lägga till, byta ut eller ta bort moduler. När vi skulle koppla ihop de olika modulerna visade det sig att en del av metoderna behövde modifieras, för att få integrationen samlad snarare än utspridd på olika ställen.

5.2 Säkerhetsaspekter

Då vi använder oss av Twitters autheniticiering och man av en sådan stor aktör kan förvänta sig en god säkerhet på den fronten (även om det givetvis inte är någon garanti) anser vi inte att authenticiering utgör någon större risk eller sårbarhet i vår applikation. Dataexponering, både i form av den öppna källkoden och risken att man råkar inkludera något man inte borde lägga publikt, och att någon faktiskt ser vad man matar in känns som en större risk. Likaså svårigheten att fullt ut validera de inputs som sker till applikationen, inte minst som vi inte hann implementera den nivå på inmatningskontroller vi hade önskat. Exempelvis skulle vi vid inmatningen av stad kanske helst velat använda ett regex-uttryck, men fick nöja oss med den lösning som presenteras i Sektion 4.7.3

Säkerheten är en viktig fråga, inte minst när det gäller applikationer som verkar över nätverk/webb (och det är få som inte gör det idag). Det är väldigt lätt att glömma av att validera alla inputs som sker till applikationen, och det är väldigt svårt att göra applikationen hundraprocentigt säker. Hur mycket tid man än lägger på att analysera och diskutera olika säkerhetsaspekter, som vi gjort i Sektion 4.7, och hur mycket man än försöker att göra den så felsäker som möjligt känns det ändå alltid som om man missat något.

Med andra ord: säkerhet är mycket viktigt men väldigt komplext och svårt.

5.3 Resultat

Vi har lyckats väl med att uppfylla både vår och projektets målsättning. Vi har med versionshanterad källkod skapat en relativt säker JavaFx-baserad app som interagerar med tre APIer och använder JSON för att hämta och skicka information över nätverk/webb. Vi har också gjort en analys av projektets säkerhetsaspekt, se Sektion 5.2.

All grundläggande funktionalitet som vi tänkt oss är uppfylld, samtidigt som det på grund av den begränsade tiden finns flera aspekter, felhantering exempelvis, som vi inte hunnit implementera eller inte hunnit implementera fullt ut. Säkerhetsaspekterna är relativt grundligt diskuterade och analyserade men måhända inte tillräckligt implementerade i den här versionen av appen.