|  |
| --- |
|  |
| Wearep |
|  |
| En app som kan twittra och rapportera väder för en plats i html och i en tweet |

**SYSTEMINTEGRATÖR YHSIPI17/Datakommunikation och nätverk**

den 7 oktober 2018

Billy Andersson, Malin Albinsson, Sten Karlsson

Wearep

En app som kan twittra och rapportera väder för en plats i html och i en tweet

Innehåll

[2 Inledning 2](#_Toc526375050)

[2.1 Sammanfattning TODO 2](#_Toc526375051)

[2.2 Inledning/Syfte 2](#_Toc526375052)

[2.3 Definitioner och förkortningar TODO 2](#_Toc526375053)

[2.4 Referenser TODO 2](#_Toc526375054)

[3 Genomförande 2](#_Toc526375055)

[3.1 Övergripande mål TODO 2](#_Toc526375056)

[3.2 avgränsningar TODO 2](#_Toc526375057)

[3.3 Arbetsmetodik TODO 3](#_Toc526375058)

[4 Wearep-appen 3](#_Toc526375059)

[4.1 Översikt TODO 3](#_Toc526375060)

[4.2 Platsdata från Google Maps API TODO 3](#_Toc526375061)

[4.2.1 Google Maps API TODO 3](#_Toc526375062)

[4.3 Väderdata från SMHIs API TODO 3](#_Toc526375063)

[4.3.1 SMHIs API TODO 3](#_Toc526375064)

[4.4 Twitter-möjlighet från Twitters API TODO 3](#_Toc526375065)

[4.4.1 Twitters API TODO 3](#_Toc526375066)

[4.5 Publicera HTML 3](#_Toc526375067)

[4.6 Teknisk lösning 3](#_Toc526375068)

[4.6.1 JavaFx TODO 3](#_Toc526375069)

[4.6.2 Utvecklingsmiljö och utvecklingsverktyg TODO 3](#_Toc526375070)

[4.6.3 Felhantering 4](#_Toc526375071)

[4.6.3.1 Regler 4](#_Toc526375072)

[4.6.3.2 Dataformatering 4](#_Toc526375073)

[4.7 Säkerhet TODO 4](#_Toc526375074)

[4.7.1 Autenticiering TODO 4](#_Toc526375075)

[4.7.2 Dataexponering TODO 4](#_Toc526375076)

[4.7.3 Sessionshantering TODO 4](#_Toc526375077)

[4.7.4 Sårbarheter TODO 4](#_Toc526375078)

[5 Slutsatser/Diskussion 4](#_Toc526375079)

[5.1 Appens implementering TODO 4](#_Toc526375080)

[5.2 Säkerhetsaspekter TODO 4](#_Toc526375081)

[5.3 Resultat TODO 4](#_Toc526375082)

# Inledning

## Sammanfattning TODO

Wearep är en app för att rapportera väderdata för en utvald plats på twitter eller i html.TODO Vi valde en MIT-licens därför att detta är en enkel applikation där källkoden kommer vara åtkomlig för alla via github. Det är inte heller något nytt, komplicerat eller innovativt så det finns anledning till större begränsningar av den anledningen.

## 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.

## Definitioner och förkortningar TODO

## Referenser TODO

Länk till git-repo

Länk till Trello-tavla

Twitters API-dokumentation

SMHIs API-dokumentation

Google Maps API-dokumentation

# Genomförande

## Övergripande mål TODO

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.

## avgränsningar TODO

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. 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.

//TODO Håller ni med? Något mer?

## Arbetsmetodik TODO

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 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.

# Wearep-appen

## Översikt TODO

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 Maps. 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

### Användarinterface TODO

Användarinterfacet ser ut så här: (//TODO bild)

en Twitterflik (inloggning krävs) /

en Weatherflik

Appen har också en inbyggd webvisare //TODO vad heter det?? som websidan öppnas i när väderprognosen är färdig.

//TODO beskriv flödet när man använder appen

//TODO bild från någon som har ett gäng tweets, gärna med vädertweets inkluderade

## Platsdata från Google geocoding API TODO

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.

Beväpnad med en 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().

### Google geocoding API TODO

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).

## Väderdata från SMHIs API TODO

Väderdata hämtas från SMHIs API (TODO länk) 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.

//TODO parametrar som sparas i JSON-objekten?

### SMHIs API TODO

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

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.

## Twitter-möjlighet från Twitters API TODO

### Twitters API TODO

## Output

### 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.

### Meddelanden //TODO

Klassen WeatherMessages producerar följande textsträngar:

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

## Teknisk lösning

### JavaFx TODO

### Utvecklingsmiljö och utvecklingsverktyg TODO

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

|  |  |
| --- | --- |
| Produkt | Ansvar |
| Eclipse | Programmeringsverktyg (IDE) |
| JUnit | Enhetstestning av kod |
| xxx | Byggstöd och rapporteringsverktyg |

### Felhantering

Javas Exceptions används för felhantering i applikationen.

#### 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.

#### Dataformatering

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

## Säkerhet TODO

/\*

Begrepp vi kanske bör ta upp här i dessa undersektioner:

Escapande

Validate data - klokt att använda reguljära uttryck och pattern matching

Kontrollera alltid att databasen går att nå, filen man ska skriva till finns etc.

SLS/TLS

Autentication

Basic

- base64

Form Based

- figurer, bilder där något ska identifieras

Oauth2

- Avancerat. Kan vara klokt val. Stora aktörer större möjlighet att säkra inloggningen. Samtidigt mer lockande för hackers.

Cross Site Scripting

- en cookie som hänvisar till en sida du är inloggad på stjäls och läggs på en annan webplats och min inloggning utnyttjas

- skydda oss genom att

* -escape all data.
* fånga upp t ex html-kod i vår applikation

Kända sårbarheter i olika ramverk

* hur är det med Maven?
* Sårbarhet Github?
* -remote exploit
* -local exploit

SQL Injections - borde inte vara så aktuellt i vårt fall, vi använder väl ingen databas? Eller hur är det med inloggningen på twitter?

- Skydda sig med Prepared Statements.

Sessionscockies

Wifi - då är kommunikationen sällan krypterad. Risk för sniffing.

T ex en webtjänst som inte använder TLS, då kan man fånga upp din sessions-id, eller få fram ditt användarnamn och lösenord i klartext.

Det som finns på min skärm - vad har vi exponerat i webgränssnittet? Känsliga uppgifter bör undvikas

Data exposure innebär att du ska tänka på vad som skickas fram och tillbaka över nätverk, och vad som är viktigt för användaren att se. Användaren kan istället aktivt få välja att visa personnummer, företagsnamn etc.

WAF - eller next generation firewall - en brandvägg står mellan internet och din applikation. Tittar på vanliga sårbarheter/sätt att ta sig in och har en lista och uppdaterar den. Den går upp i application layer och tittar, vad kommer egentligen, aha, den försöker göra en cross site scripting, det gilllar vi inte, då blockerar brandväggen det,

\*/

### Autenticiering TODO

TODO

### Dataexponering TODO

﻿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.

### Sessionshantering TODO

TODO

### Sårbarheter TODO

Vissa metoder är public fast man kanske kunde gjort dom private. //TODO sant?

Vi ska även bygga till kontroller av inmatad data, utifall nån skriver in "4" som stad t.ex. //TODO utveckla

Vem som helst kan använda väderdelen av appen för att hämta väderdata. där krävs ingen autentisering, lika lite som det krävs för att titta på väderprognosen på SMHIs egna sidor. För att använda twitter däremot krävs inloggning på ett twitterkonto. //TODO hör det hemma här eller under autentisiering?

//TODO Kan det ses som en sårbarhet att vi sparar html-filen lokalt? Den som har html-kunskap kan ändra i den, men den är å andra sidan inte publikt publicerad.

//TODO Borde vi fånga upp inkommande data (inmatning av stad?) och escapa den för att skydda oss serverside?

//TODO Finns det sårbarheter i Maven som skulle kunnat påverka vår app?

# Slutsatser/Diskussion

## Appens implementering TODO

//TODO Flytta det mesta här till 5.3 Resultat istället?

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. 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 en klass. Detta underlättar om vi skulle vilja lägga till, byta ut eller ta bort moduler. // Detta är i varje fall implementationsdiskussion När vi skulle koppla ihop de olika modulerna visade det sig att en del av metoderna behövde skrivas om. Det kändes vettigare att ha integrationen samlad på ett ställe än utspridd i olika klasser.

## Säkerhetsaspekter TODO

TODO

## Resultat TODO

Vi har lyckats väl med att uppfylla både vår och projektets målsättning. Vi har med versionshanterad källkod skapat en 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. //TODO slutsatser/resultat säkerhetsaspekter  
  
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 analyserade men måhända inte tillräckligt implementerade i den här versionen av appen.