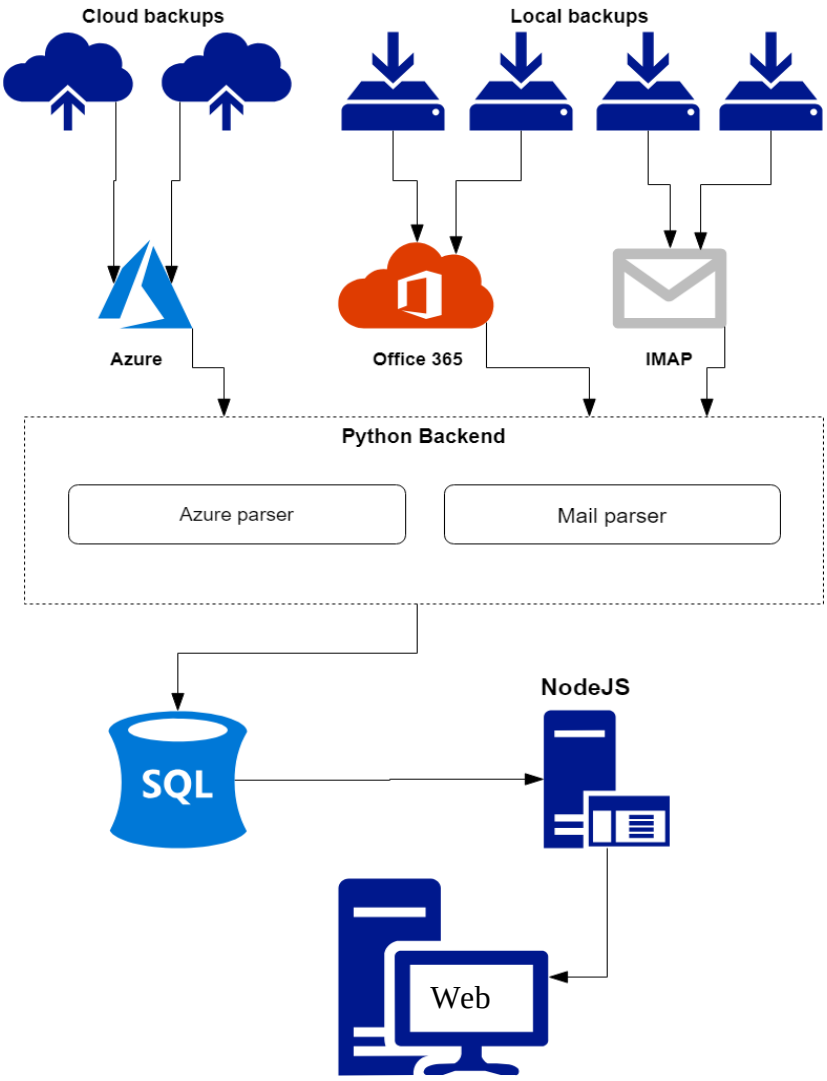
CHALMERS



Centralt Backupsystem

System för monitorering av backups

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

Nikolai Dahlberg

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA

GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2019

EXAMENSARBETE

Central Backup System

Metoder och val som gjorts för skapandet av ett omfattande system för monitorering av backups.

Nikolai Dahlberg

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2019

Central Backup System

Metoder och val som gjorts för skapandet av ett omfattande system för monitorering av backups.

Nikolai Dahlberg

Examinator: Peter Lundin, Data- och Informationsteknik

Handledare: Sakib Sistek, Data och Informationsteknik

Institutionen för Data- och Informationsteknik

Chalmers Tekniska Högskola / Göteborgs Universitet

412 96 Göteborg

Telefon: 031-772 1000

Central Backup System

Metoder och val som gjorts för skapandet av ett omfattande system för monitorering av backups.
Nikolai Dahlberg
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborg 2019

Sammanfattning

Consat Data AB hanterar IT för många av sina kunder. För varje kund finns det backupservrar med olika mjukvaror. En backupserver är en central punkt för alla backups som skapas hos en kund. När en backup har skapats behöver administratören antingen läsa ett mail eller logga in i backupsystemet för att se hur det gått med backupen. Detta kan resultera i att en administratör behöver läsa upp till 300 mail om dagen vilket är ineffektivt.

Projektet innebar att effektivisera arbetet med backups genom att skapa ett system som läser alla mail och hämtar data om backups för att sedan visa detta på en hemsida. Systemet delades upp i två applikationer, en webapplikation som styr hemsidan och en pythonapplikation som läser mail och hämtar data.

Projektets syfte var att undersöka olika metoder för att hantera data från backupservrarna och visa dessa i ett webinterface. Metoderna undersöktes och valdes beroende på funktionalitet, popularitet och existerande dokumentation.

Arbetet resulterade i två olika prototypapplikationer. Den första applikationen skrevs i Python. Applikationen hämtar mail samt backupinfo och utvinner backupdata från dessa. Den andra applikationen är en webbapplikation med Node.JS som grund. Webbapplikationen visar upp den utvunna backupdata i ett webbinterface.

Nyckelord: Node.JS, Python, Rest, Microsoft Azure, Backup, Microsoft Office 365, texthantering, Webapplikation, IMAP.

Abstract

Consat Data AB manages other companies' IT environment. For every customer there are backup servers with different softwares. These are responsible for creating backups of the customers' servers and send reports to the administrators when done. Due to the number of customers and servers the administrators can receive up to 300 reports a day through mail and web interfaces. One administrator must read through all these reports every day which makes the work inefficient. This is what this project tries to solve.

The goal of the project was to make Consats work with backups more efficient. This was done by creating a system that fetched backup data and presented this in a web interface. The system was split into two applications, one to handle the web interface and one python application to handle, fetch and read the backup data.

While working on the goal the purpose of the project was to discover and examine different methods to fetch, handle and view backup data. The different methods were chosen depending on functionality, popularity and existing documentation.

The work resulted in two working prototype applications. The first application is written in Python. The purpose of the application is to fetch mails and backup information to extract backup data. The second application is a web application with Node.JS as a base. It's purpose is to show the extracted backup data in a webinterface.

Keywords: Node.JS, Python, REST, Microsoft Azure, Backup, Microsoft Office 365 texthandling, Webapplication, IMAP

Förord

Jag vill tacka till de anställda på Consat som gav mig möjligheten att arbeta med detta projektet och behandlande mig som anställd på företaget.

Jag vill även tacka min handledare Sakib Sistek för att han alltid ger mig stöd och tar mig på allvar.

Till sist vill jag tacka Cornelia Johansson för motivationen hon gett mig till att slutföra rapporten!

Nikolai Dahlberg, februari 2019.

Innehållsförteckning

Terminologi

API	Application Programming Interface, verktyg för att skapa kommunikation mellan applikationer
Azure	Molntjänster utvecklade av Microsoft, rapporten syftar specifikt på backuptjänster i molnet
Backend	Den delen av applikationen som arbetar ”bakom kulisserna”
Backup	Kopierad data som sparas på annan plats utifall originaldata skulle försvinna
CSS	Cascade Style Sheet, språk som används för design av hemsidor
CURL	Client URL, mjukvara för att hämta resurser på nätet
Frontend	Detsamma som UI.
Git	Versionshantering för kod.
HTML	Hypertext Markup Language, programmeringsspråk för att skapa strukturer för hemsidor
IMAP	Internet Message Access Protocol, ett internetprotokoll för att hämta mail från mailservrar.
Javascript	Programmeringsspråk som ofta används för att skapa interaktiva hemsidor men även i webapplikationer.
JSON	JavaScript Object Notation, ett textbaserat sätt att spara data.
Mailhuvud	Del av ett mail som innehåller bland annat avsändare, mailidentifierare och ämne
Mailidentifierare	En unik textrad som identifierar motsvarar ett specifikt mail
Mailkropp	Delen av ett mail som innehåller meddelandet
MVC	Model View Controller, ett sätt att strukturera applikationer
NLTK	Natural Language Toolkit, mjukvarubibliotek för hantering av text i naturligt språk
Oauth2	Open Authorization standard, används för autentisering mot bland annat REST API
Office 365	Molnbaserade tjänster till lokala applikationer utvecklade av Microsoft

Open source	Öppen källkod. Mjukvara där koden är fritt tillgänglig för alla att använda, modifiera och distribuera.
POP3	Post Office Protocol 3, används för att hämta mail från en mailserver
Python	Mångsidigt programmeringsspråk som är tillgängligt för de flesta operativsystem
Regex	Regular expression, ett sätt att uttrycka mönster för att söka i text
REST	Representational State Transfer, applikationsstruktur för storskaliga applikationer.
RFC	Request For Comments, standarder som definierar olika internetstandarder.
Route	Backend för en websida i Node.JS med express
Runtime	En samling av mjukvaror som gör det möjligt att köra andra program. T ex Javascript Runtime gör det möjligt att köra kod skriven i javascript.
SNMP	Simple Network Management Protocol, nätverksprotokoll för hantering av enheter på nätverk
Template engine	En mjukvara som använder mallar (så kallade templates) för att skapa data, i detta fallet hemsidor
Text Parsing	Översättning av text till användbara data
UI	User Interface, delen av en applikation som användaren interagerar med
Web interface	Hemsidan som användaren ser.
Websocket	En mjukvarudel som kopplar en applikation till ett nätverksprotokoll

1

Inledning

1.1 Bakgrund

Consat Data AB:s (Consat) företagsidé innebär att erbjuda företag helhetslösningar i drift av IT. Dessa lösningar involverar även backup för kundföretagens data. Varje företag äger en eller multipla backupservrar som administreras med olika mjukvaror. När en backup av data har skapats skickas ett mail från backupserver till administratörerna på Consat. Detta resulterar i 200–300 mail om dagen som administratörerna behöver läsa igenom för att ta reda på status hos samtliga backups.

1.2 Mål

Målet med projektet är att effektivisera administratörernas arbete med backups genom att skapa ett system som visar status hos backups i ett webinterface.

1.3 Syfte

Syftet med projektet är att identifiera olika metoder att hantera data från backupservrarna samt metoder för att visa detta i ett UI som fyller Consats behov.

1.4 Frågeställningar

Projektet delas upp i två deluppgifter:

1. Skapa en service för att hämta data från mail, REST API och eventuellt SNMP och lägga till denna i en databas.
2. Skapa ett webinterface för att visa informationen om backups.

1.5 Avgränsningar

- Data från backupsystem skall endast inhämtas via mail samt via REST API från Microsoft Azure.
- Inga mailimplementationer förutom Office 365 och grundläggande IMAP ska implementeras.
- Säkerhet behandlas inte i detta projektet.
- Ingen hårdvara behandlas i detta projektet.
- Följande avgränsningar gjordes på grund av brist av tid:
    - Bootstrap används som designbibliotek, inga andra alternativ utforskas.
    - Tabeller i databasen skall endast användas för att lagra data. Inga relationer mellan data eller tabeller skapas. Inga restriktioner på data ska tillämpas.

2

Teknisk bakgrund

2.1 Mjukvaror

Här beskrivs de mjukvaror och vissa programmeringsspråk som använts i projektet.

2.1.1 Operativsystem

Systemet utvecklades till en server som körde Debian 9. Debian är ett Linuxbaserat operativsystem som är open source. Detta gör att användarna kan implementera en funktion som de efterfrågar utan att gå via ett företag. [1]

2.1.2 Python

Python är ett objektorienterat programmeringsspråk med många användningsområden. Python används inom allt från forskning [2] till webbservrar. [3] Python har en stor databas över paket med utökad funktionalitet, de flesta är skrivna av andra utvecklare. Dessa paket hämtas av pythons pakethanterare, pip. [4] Det är högst relevant att veta vilken pythonversion en utvecklare programmerar i, eftersom en del paket inte är kompatibla med de nyare versionerna. För att ha möjlighet till att använda majoriteten av paketen användes pythonversion 3.5.

2.1.3 Node.JS

Node.Js är ett javascript runtime som gör det möjligt att skriva nätverksbaserade javascript applikationer. För att ha stöd för javascript använder Node.JS Googles V8 motor. [5]

Node.JS är open source vilket gör det möjligt för alla utvecklare att påverka källkoden för Node.JS. Detta gör även att tredjepartsutvecklare kan skapa paket för att utöka Node.JS funktionalitet. Dessa paket hanteras av NPM (Node.JS Package Manager). Paket kan hittas på npmjs.com. På npmjs.com finns nedladdningsstatistik och information om hur stabilt varje paket är. Detta gör valen av paket enklare och säkrare. [6]

2.1.4 Atom

Atom är en textredigerare som i projektet användes för att programmera all kod. Atoms kod är open source. [7] Detta gör att tredjepartsutvecklare kan utveckla utökad funktionalitet till applikationen. Denna utökade funktionalitet representeras som plugins i applikationen. Användaren kan då välja vilken funktionalitet de vill ha genom att ladda ner olika plugins. Alla tillgängliga plugins finns på Atoms hemsida. [8]

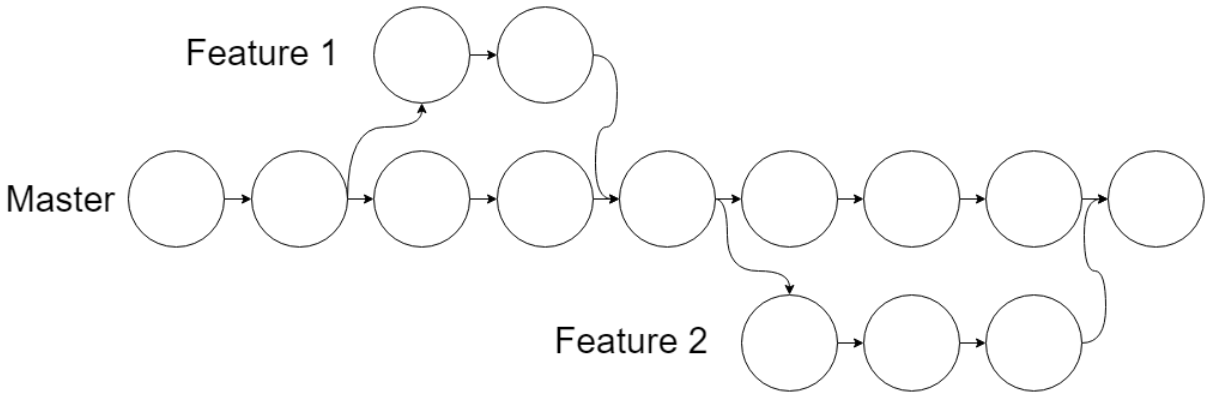
Atom stödjer alla programmeringsspråk som det finns plugins för. Därför fungerade det bra för att programmera i de programmeringsspråk som projektet innefattade. Atom har även stöd för Git som användes för att versionshantera koden.

I projektet användes olika paket för att automatiskt komplettera kod, uppladdning av filer till server och skapa stöd för fler programmeringsspråk.

2.1.5 Git

Git är ett versionshanteringsprogram som används för att hålla koll på versioner av mjukvaruprojekt. [9] Git gör det möjligt att se vem som skrev vad i olika versioner av filer. Detta gör det enklare att hitta källan när problem uppstår i projektet.

Det finns många arbetsmetodiker för Git. I projektet användes en metodik vid namn ”feature branches”. Denna metodik går ut på att olika delar (features) utvecklas i andra grenar från huvudgrenen. När en del är färdigutvecklad slås grenen ihop med huvudgrenen.



Figur 2-1-Exempel på Feature branches

2.2 Processer

Här beskrivs olika processer som är involverade i projektet.

2.2.1 Autentisering med OAUTH2 och användande av REST

REST är en mjukvarudesign skapad för att förenkla kommunikation mellan server och klient över webben. Designen använder sig av http-metoder för att hämta, uppdatera, publicera och radera data på servern. REST implementerar stateless communication, detta betyder att servern inte behöver veta något om klienten för att kunna slutföra en förfrågan. [10]

Oauth2, open authorization, är ett protokoll för autentisering av klienter gentemot servar. För att identifiera en autentiserad klient använder Oauth2 en textsträng, så kallad token. För att få en token behöver klienten skicka en förfrågan till Oauth2. Förfrågan måste innehålla ett lösenord (secret), för att servern ska veta att det är en tillåten klient. [11]

Oauth2 och REST används i projektet för att autentisera och hämta data från Azure och Microsoft 365. Processen för att autentisera och hämta data från servrarna sker i flera steg, illustrerat i figuren nedan.

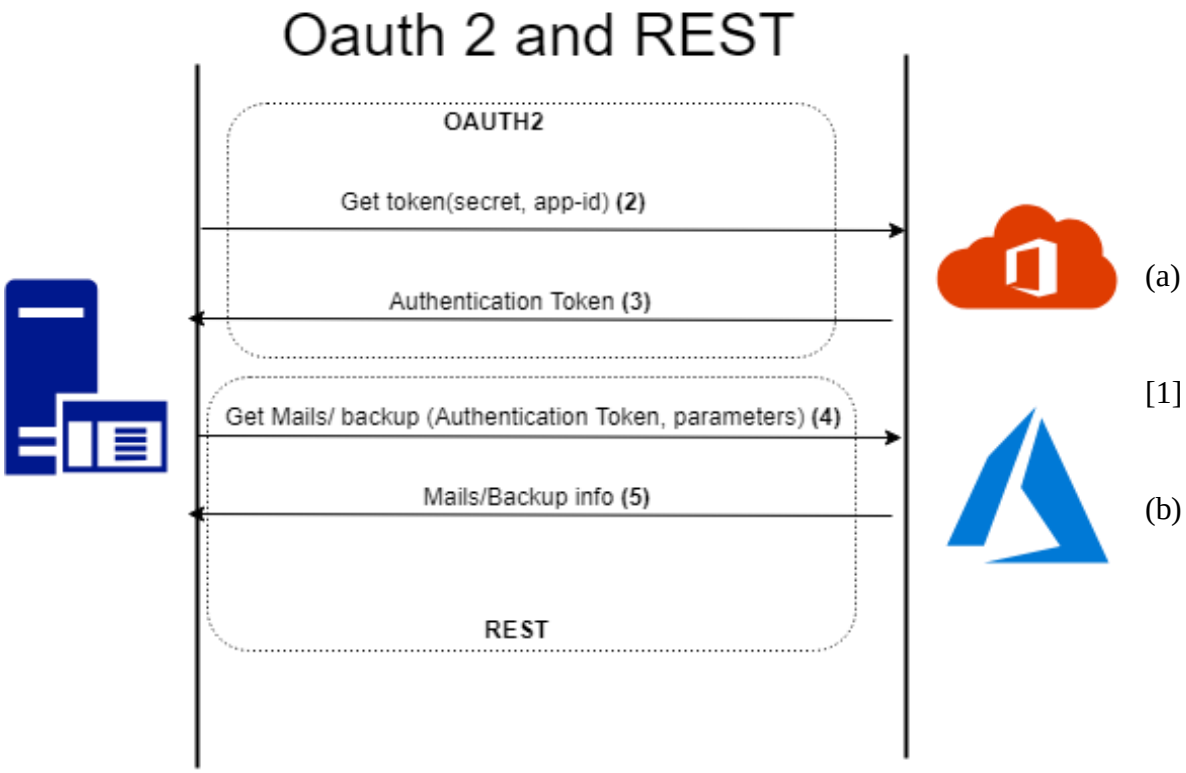


Figure 2-2 Oauth and Rest Communication

1. En applikation skapas i Microsoft 365 (a) eller Azure (b). Applikationen har ett hemligt lösenord som visas en gång och ett app-id.

2. En förfrågan om en token skickas till servern. Det hemliga lösenordet samt app-id skickas med förfrågan för att autentisera klienten.

3. En authentication token skickas till klienten.

4. En förfrågan om att skapa/hämta/uppdatera/radera data skickas till serverns REST endpoint. För att servern ska veta att förfrågan är korrekt måste en authentication token skickas med. Oftast behövs även information om den data förfrågan gäller.

5. Servern skickar informationen som efterfrågas eller en status över hur operationen gick.

2.2.2 Backup till Mail

En backupserver hanterar backups för flera servrar. När en status för en backup har uppdaterats notifieras administratören av backupservern. Notifikationerna ser olika ut beroende på system. Azure skickar ut ett mail endast om den misslyckas göra en backup. Resterande av backupservrarna mailar ut status. Mailen har olika typer av information i sig och ser olika ut beroende på backupmjukvara.

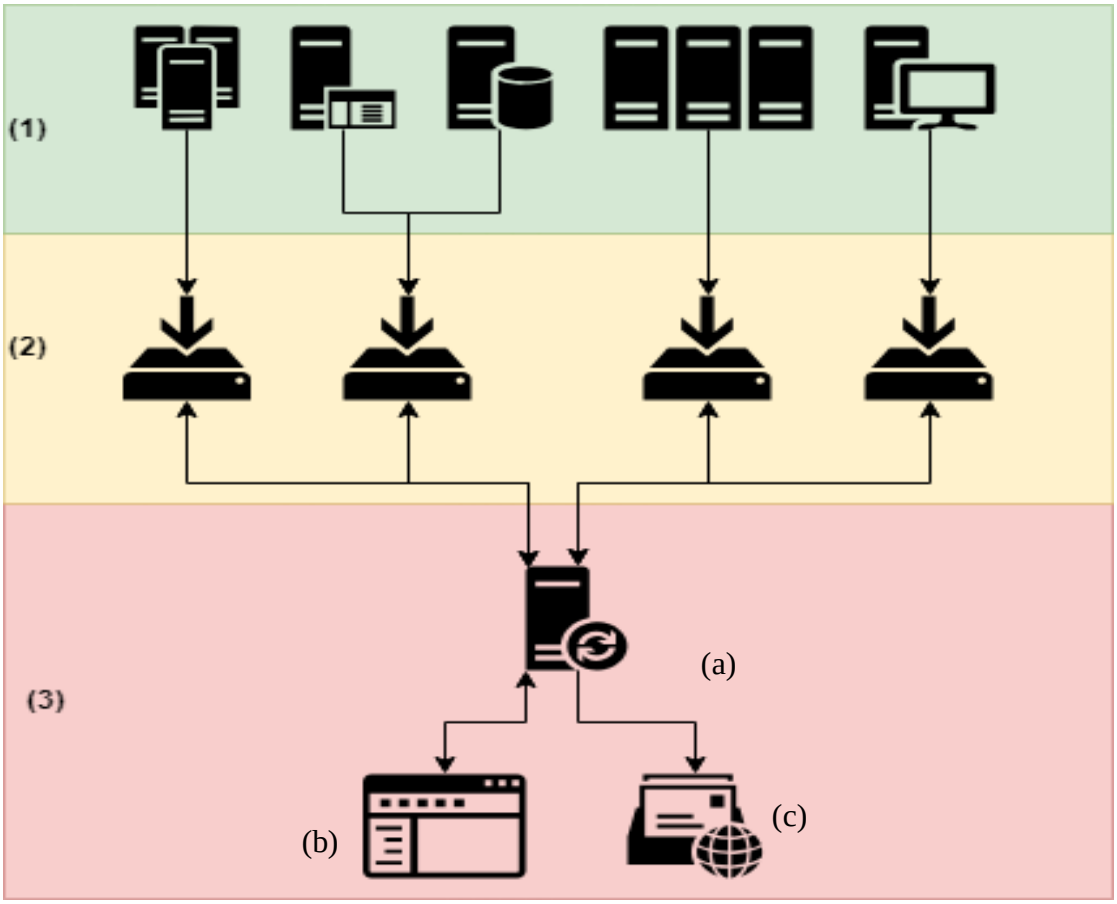


Figure 2-3-Information flow from backup to mail

1. Servrar som backas upp, backupadministratörerna behöver inte administrera dessa för att hantera backups.
2. Backups. Dessa kan skapas och hanteras av både användare och backupadministratören.
3. Backupserver för att backupadministratören ska hålla koll på backups. (a)
Applikation/hemsida där administratören kan läsa om status.(b)
Mail om status som skickas till administratören.(c)

3

Metodbeskrivning

3.1 Planering

Här beskrivs arbetsmetoder i projektet samt vad som bestämdes i planeringsfasen.

3.2 Arbetsmetoder

I projektet användes en implementation av den agila arbetsprocessen scrum. Scrum gick ut på att arbetet delades upp i mindre delar, så kallade. tasks. En task var en del av arbetet som uppfyllde en viss funktion. Till exempel om arbetet var att skapa ett spel där en spelare skulle kunna hoppa så kunde hoppfunktionen vara en task. [12]

Arbetet lades sedan upp i intervaller, så kallade. sprints. Inför varje sprint utfördes en sprint planning, där valdes tasks som skulle utföras under varje sprint. När en sprint var färdig utfördes en review för att reflektera över det genomförda arbetet. Efter en review utfördes en så kallad retrospective där diskussion kring problem under sprinten samt eventuella lösningar diskuterades. [12] Då projektgruppen endast bestod av en person hölls review och retrospective väldigt korta. För att hålla reda på arbetets framsteg, tidsåtgång och framtida arbete behövdes en visuell representation av tasks och sprints. I projektet användes en whiteboard i Consats kontor med en post-it för varje task och sprint.

Till skillnad från den traditionella vattenfallsmetoden tillåter scrum ändringar från kunden i alla steg av utvecklingen. Vattenfallsmetoden är en arbetsmetod som utgår från definierade faser. Faserna för programutveckling kan till exempel vara ”Design” ->”Konstruktion”-> ”Testning”. I scrum delas arbetet upp så varje liten task hade sin egen korta process som kan innebära samma process som i vattenfallsmetoden men i mycket mindre skala. När en task är klar levereras denna till kunden, på så sätt kan kunden se produkten växa fram del för del inom små intervaller till skillnad från vattenfallsmetoden där stora delar av produkten levereras med stora intervaller. Detta gör att utvecklarna kan förlita sig mer på produkten då alla saker testas individuellt samt att leveranserna av produkten blir tätare. [13]

3.3 Förstudier

Då arbetet utfördes enligt scrum utfördes det en förstudie inför varje task. En förstudie kunde innebära att ny kunskap behövde hämtas och att tester behövde utföras. Nedan beskrivs olika källor till kunskap om huvuddelar av projektet samt tester som utfördes innan implementation.

3.3.1 Kunskapsinhämtning

I detta avsnittet beskrivs de primära källorna till kunskapen som inhämtats i förstudien.

3.3.2 Backupmjukvaror

Consats kunder använde sig av flera olika backupmjukvaror. Informationen om varje backup som skapades skickades till Consat genom varierande metoder. För att samla kunskap om mjukvarorna fördes diskussioner med Consats anställda. När en ny mjukvara upptäcktes frågades Consats anställda om information sen söktes mjukvaran upp på nätet för vidare information.

3.3.3 Microsofts tjänster

Kunskapen kring Microsofts tjänster inhämtades från Microsofts dokumentation för varje individuell tjänst. Flera av kundernas backups samt Consats mail låg i molnet hos Microsoft vilket gjorde det nödvändigt att samla kunskap om hur dessa fungerade.

3.3.4 Textparsing

För att hitta den bästa lösningen till att extrahera data från text undersöktes flera olika metoder. Inspirationen till textklassifikation kom från bibliotek som NLTK[14] och TextBlob. [15] Genom att studera dessa var det möjligt att skapa en egen metod för textklassifikation.

3.3.5 Node.JS

Node.JS är uppdelat i moduler där varje modul representerar en viss funktionalitet.[32] För att samla kunskap om vilka moduler som passade för projektet söktes information upp på olika forum och på modulernas egna sidor på npmjs.com.

3.3.6 Python

Kunskap om Python hämtades från Pythons egen dokumentation. [16] Enskilda problem söktes upp på forum för utvecklare. Information om tredjepartskod hämtades från dokumentationen hos tredje part.

3.3.7 IMAP

Kunskapen kring IMAP inhämtades från RFC-standarderna [17] samt från python biblioteket imaplib. [18]

3.4 Testning av lösningar

När kunskapen inhämtats fortsatte förstudien med testning av de funna metoderna. Här följer resultatet av förstudien.

3.4.1 Testning av olika metoder för extrahering av relevant data

Testen skedde i två delar.

1.Testning av metoder för att extrahera rapporter från backupservrarna.

2.Testning av metoder för att extrahera data från text.

Backupservrarna hade olika mjukvaror. För att identifiera en generell metod för att hämta backuprapporter behövdes flera metoder testas på varje mjukvara. SNMP testades först för att se om det på något sätt var möjligt att extrahera data. Detta visade sig dock snabbt vara omöjligt då SNMP inte implementerades i backupmjukvarorna. Alla mjukvaror visade sig dock ha en funktion gemensamt, de kunde skicka sina rapporter via mail.

De flesta rapporter skickades via mail och var därför skrivna i rent textformat eller HTML-format. Alla mailen var också strukturerade olika. Extrahering av data var därför mer komplicerat än väntat. Flera lösningar testades för att extrahera data.

Det första som testades var att extrahera data från mailen genom att ta fram skillnaden mellan tidigare mail och nya mail och på så sätt extrahera den data som skiljer sig mellan mailen. Detta visade sig inte fungera då mailen skiljde sig för mycket från varandra.

Efter detta söktes inspiration på nätet där NLTK [14] och TextBlob [15] hittades. NLTK och TextBlob använder sig av något som kallas text classification där ord och sammanhang kategoriseras som olika datapunkter. Då både NLTK och TextBlob anses vara för generella testades en egen implementation av text classification och den blev sedan den officiella implementationen.

3.4.2 Testning av olika metoder för hämtning av mail

Då Consat använder Microsoft Office 365s mailtjänster lades mycket fokus på att använda dessa för att hämta mail. Office 365 implementerar inte de standardiserade mail-protokollen. Om då Consat skulle byta mailtjänst skulle applikationen inte kunna användas. Därför undersöktes mer generella mail-protokoll. Undersökningen visade att IMAP fyller syftet. [17]

Office 365 använder ett REST API för att överföra data till webbklienter. Genom att läsa dokumentationen för REST API:t och testa att skicka förfrågningar med CURL hittades en metod för att hämta mail. [19]

IMAP testades genom pythons inbyggda bibliotek ”imaplib” med information från dokumentation och RFC-standarderna för IMAP. [17,18]

3.4.3 Testning av metoder för hämtning av data ej extraherat från mail

Rapporterna kom inte bara via mail, backups gjordes också av Microsoft Azure som inte mailar ut rapporter. Azure lagrar och hanterar sina backups i vad Microsoft kallar ”Recovery services”. [20] Azure har ett REST API för hämtning av data från deras servrar. För att testa vilka REST-noder som skulle kontaktas användes Azures dokumentation och CURL. Dokumentationen innehöll information om många noder och var otydlig. Varje nod lästes därför igenom och testades med CURL om denna ansågs vara relevant. Detta resulterade i att backupdata kunde extraheras från Azures servrar.

3.4.4 Val av databas för extraherade data

Kravet för databasen var att den skulle ha bra dokumentation och ha stort stöd för implementation i flera programmeringsspråk. Den databas som valdes ut var PostgreSQL då den har stort stöd i flera språk. Då tidigare kunskaper finns inom PostgreSQL var den enklare att implementera än andra alternativ som till exempel MySQL.

3.4.5 Testning av olika moduler för Node.JS applikationen

För att kunna använda Node.JS i projektet behövdes ytterligare moduler för att utöka funktionaliteten. Eftersom projektet använde sig utav PostgreSQL som databas behövdes en anslutning mellan databasen och Node.JS skapas. Node.JS stöder inte kommunikation med PostgreSQL utan en extern modul. Modulen ”Node-Postgres” (pg) undersöktes och visade sig uppfylla behovet. Pg testades genom att manuellt skicka och hämta data från databasen. [21]

Dynamiska data skulle visas på webinterfacet. Node.JS stöder inte dynamiska data vilket gör en extern modul för detta nödvändigt. De som testades i projektet var Handlebars och Mustache. Båda fungerade mycket väl för att visa dynamiska data men då Handlebars var populärare och hade bättre nedladdningsstatistik på npmjs.com valdes denna. [22]

3.4.6 Testning av realtidskommunikation med Node.JS

För att användaren skulle kunna interagera med webinterfacet på ett lämpligt sätt behövdes någon typ av realtidskommunikation. Utbudet av moduler för realtidskommunikation i Node.JS var smalt men en av dessa var Socket.IO. Socket.IO ör ett populärt bibliotek som använder webbsockets för realtidskommunikation mellan klient och server. Innan Socket.IO implementerades testades det genom exempel, dokumentation och guider på tillhörande hemsida. Socket.IO valdes då det var enkelt att komma igång med och var det populäraste alternativet på npmjs.com. [23] Ett exempel på ett annat bibliotek är Websocket, detta är mindre intuitivt att använda och har mycket mindre funktionalitet. [33]

3.4.7 Testning av designmetoder för hemsida

En stor del av projektet handlade om att skapa ett webinterface som visade data på ett övergripande sätt med möjlighet till fördjupning. För att kunna skapa ett webinterface utan designkunskaper behövdes ett bibliotek som hjälper till att göra sidan enhetlig och ger efterfrågad funktionalitet. Det bestämdes tidigt att inte mycket tid skulle läggas ner på att hitta ett sådant bibliotek. Biblioteket Bootstrap valdes då det hittades snabbt och uppfyllde kraven. [24]

4
Systemkonstruktion

I detta kapitlet beskrivs konstruktionen av hela systemet, från backups till webinterface, se översikt i figur 4-1.

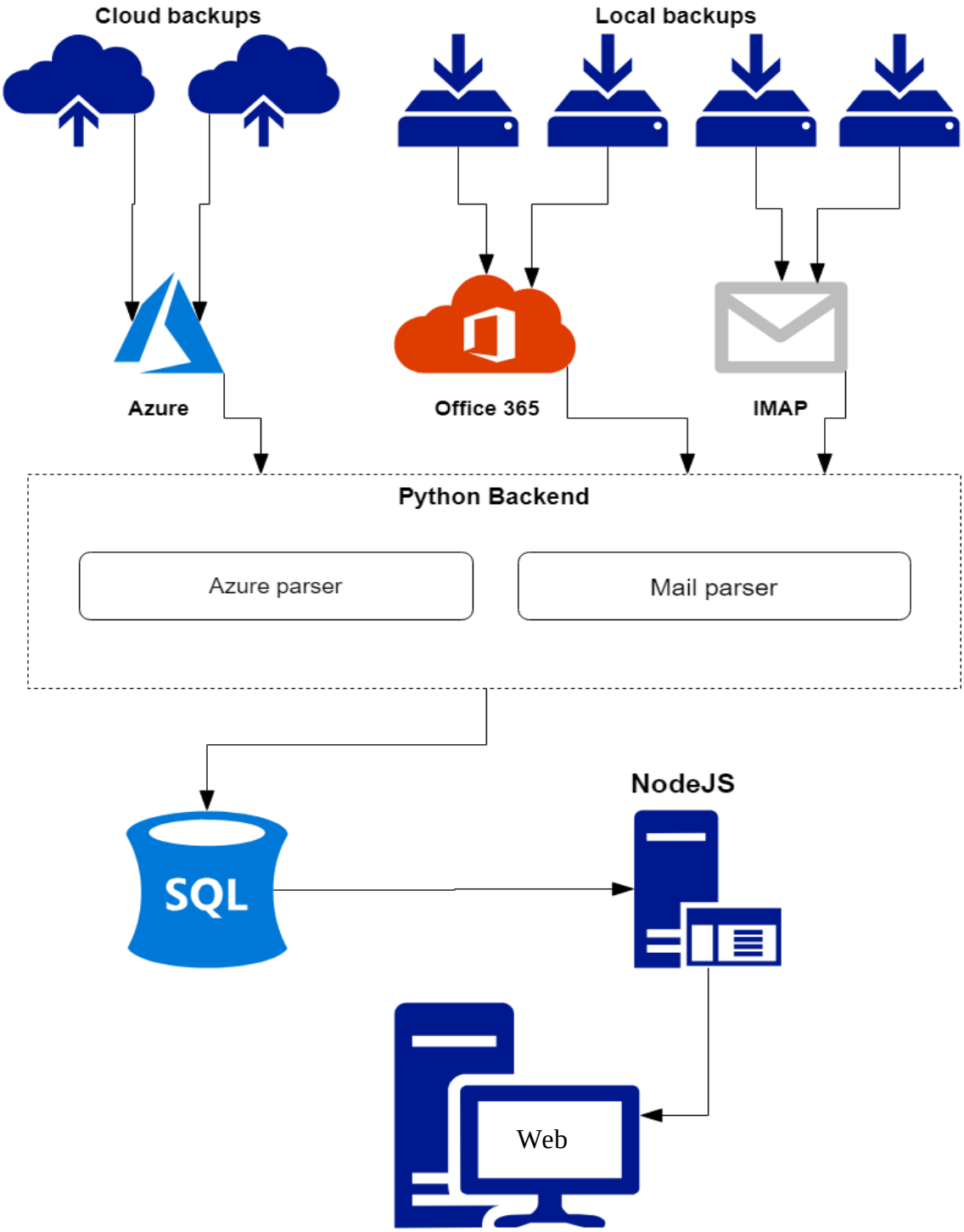


Figure 4-1- Hela systemet från backups till webinterface

4.1 Pythonapplikation

Här följer information om hur Pythonapplikationen är konstruerad.

4.1.1 Office 365

För att hämta mail från Office 365 via tillhörande REST API krävs en mail-adress att hämta från samt administratörernas tillåtelse att hämta mailen. Administratören behöver ge tillåtelse i ett webinterface på Office 365. [19] När administratören gett tillåtelse och användaren angivit mailadress att hämta från läggs mail-adressen in i databasen. Applikationen hämtar därefter mail-adressen var femte minut för att skicka förfrågningar efter nya mail till Office 365. Om det finns nya mail hämtas dessa och skickass till parsern för utvinning av data.

4.1.2 IMAP

Consat har inte en mailserver som stödjer IMAP som projektet kunde utnyttja. För att implementera IMAP sattes ett mailkonto upp på ww.fsdata.se som stödjer IMAP.

IMAP-implementationen slogs ihop med implementationen för Office 365. Uppgifter för autentisering hämtas från databasen och används för att hämta mail var femte minut. Mailen skickas sedan vidare till mailparsern för utvinning av data.

4.1.3 Azure

Azure använder sig av REST API för kommunikation samt Oauth2 för autentisering. Då Office 365 använder samma autentiseringsmetod implementerades en generisk hanterare för Oauth2. Även i Azure krävs det tillåtelse av en administratör för att kunna hämta data från Azure. Varje kunds Azure har en egen unik identifierande text i Azure. Denna användes för att specifiera vilken kund data skulle hämtas från.

Vid implementationen av Azure lades mycket fokus på automatisering. Backups skulle därför upptäckas automatiskt av systemet när administratören gett tillåtelse och den unika identifieraren lagts till av användaren. Detta implementerades genom att söka i alla möjliga platser på det kopplade Azure-kontot och skicka förfrågningar (requests) om att hämta data från dem. [20] I implementationen görs detta var sjätte minut för att eventuella förändringar i kundens Azure ska upptäckas och appliceras i systemet.

Azures REST API svarar med data i JSON-format. Pythons inbyggda JSON-parser används för att formatera data till ett användbart format. [25] Utvunnen data sparas sedan i databasen för framtida användning.

4.2 Mailhantering

Denna delen handlar om utvinning av data från mail.

Mailen som kommer från Office 365 eller IMAP kans vara skrivna i två olika format, HTML eller klartext. Dessa hanteras olika då mail i HTML format upptäcktes ha tydligare struktur än mail i klartext.

4.2.1 HTML

Utvinnande av data ur text där HTML-taggar är inblandade visade sig vara svårt. Därför implementerades en HTML-parser för att behålla strukturen i mailen men utan HTML-taggar. Denna utvecklades med hjälp av den inbyggda HTML-parsern i Python. [26] Texten skickas sedan vidare för att klassificeras.

4.2.2 Klartext

Mail i klartext hanteras genom att läsa av specialtecken och sammanhang för att extrahera data. Till exempel extraheras text innanför parenteser och ordföljder innehållande specifika ord. (se exempel 1 nedan) Den extraherade texten skickas sedan vidare för att klassificeras.

4.2.3 Extrahering av data

Olika metoder testades för att identifiera relevant data i text. Den första metoden gick ut på att hitta skillnader i mail för att kunna identifiera data som ändrar på sig. Den data som ändrade på sig skulle då vara relevant. För att testa detta användes Pythons inbyggda bibliotek ”difflib” [27]. Denna metoden visade sig vara ineffektiv då de olika mailen skiljde sig för mycket från varandra.

Eftersom difflib inte fungerade som väntat letades en annan metod upp. Den funna metoden kallas klassficering. Klassificering finns i flera olika texthanteringsbibliotek, till exempel NLTK [14] och TextBlob [15]. Då dessa bibliotek implementerar en generell klassificeringsmetod och inte var anpassad för backupmail var dessa inte applicerbara på applikationen. Därför utgjorde dessa bibliotek endast en källa till inspiration för att skriva en egen klassificeringsmetod mer anpassad för backupmail. Metoden skrevs i Python då språket har god funktionalitet för texthantering. [34]

Metoden börjar med att identifiera strukturer i texten. Radbrytningar, specialtecken och ordföljder delades upp. Sedan väljs olika strukturer ut beroende på vad för ord de innehåller. Till exempel ordet ”size” innebär att det kommer en storlek i närheten av ordet. Orden får sedan en typ, till exempel ordet ”size” får typen ”storlek”.

När strukturerna valts ut identifierades datatyperna. Om datatyperna stämde överens med strukturens datatyp extraherades texten till databasen. (se Exempel 1)

Exempel 1

Identifiering av data från ett mail.

Den extraherade texten från mailet:

Name:Server 1
Size:1GB
Date:8
The backup job was a success.

I första steget identifieras och typsätts strukturen:

---

namn:data
Storlek:data
Datum:8
Status:fritext

Orden ”data” och ”fritext” symboliserar text som inte fått en typ ännu.

I andra steget identifieras och verifieras data:

Namn:namn
Storlek:storlek
Date:Error
Status:status

Eftersom ”8” inte är ett korrekt datum verifieras inte detta.
Följande data extraheras därför och läggs till i databasen

Name: Server 1
Size: 1 GB
Status: Success

De olika typerna identifieras genom mönster. Mönster identifierades genom användning av regex. Till exempel så identifierades ”1 GB” som en storlek för att det är en siffra följt av texten ”GB”.

I databasen associeras all inhämtade data med ett mail. För att identifiera ett mail hämtas därför mailidentifierare, avsändare samt tid vid avsändning från mailhuvudet på varje mail. [28]

När data klassificerats validerades den. Valideringsprocessen applicerade krav på data som var extraherad från ett mail.

- Namnet på backupjobbet ska vara kortare än 48 tecken
- Namnet består av mer än 50% bokstäver
- Data som extraherats från mailet innehåller följande typer:
    - Namn
    - Mailidentifierare
    - Avsändare
    - Status

Efter validering placeras data i databasen för att enkelt kunna hämtas vid behov.

4.2.4 Databas

För att kommunicera med databasen användes ett tredjepartsbibliotek, Psycopg2. Detta valdes då det rekommenderas av skaparna av PostgreSQL. [29]

För att spara tid hölls databasens struktur enkel utan kopplingar mellan data.

Tabell	Syfte
Mail_name	Koppla mailidentifierare till avsändare
mail_addresses_365	Mailadresserna som mail ska hämtas från i Office 365
App_info_365	Secret och app-id för applikationen i office 365
Token_info_365	Här sparas authentication tokens för office 365 REST kommunikation
Imap_auth	Kontouppgifter för imap konton

token_info_azure	Här sparas authentication tokens för Azure REST kommunikation
groups_azure	Information om subscriptions i Azure
status_jobs	Information om olika backupjobb
app_info_azure	Secret och app-id för applikationen i Azure
vaults	Information om vaults för recovery services i Azure
subscriptions	Information om subscriptions för recovery services i Azure

4.2.5 Föråldrade data

Backups görs kontinuerligt, oftast varje dag. Detta gör att data snabbt kunde bli föråldrad. För att kunna göra bedömningen om när data anses vara föråldrad diskuterades detta med Consat. Efter diskussioner sattes gräns på tre månader för sparande av data. För att implementera detta skapades en rutin i pythonapplikationen som letade och raderade föråldrade data var femte minut.

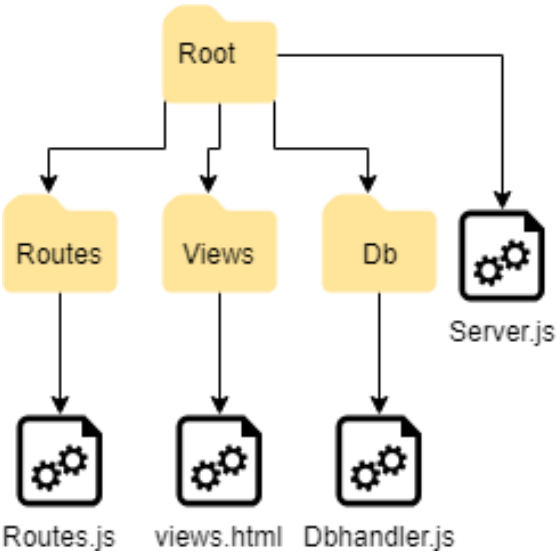
4.3 Webapplikation

Här beskrivs strukturen och funktionerna bakom webapplikationen.

4.3.1 Applikationsstruktur

Applikationen följer MVC-mönstret, vilket går ut på att applikationens källkod struktureras i tre kategorier, Model, View och Controller. [30]

- Model innefattade funktionalitet som har med datainhämtning att göra. Datainhämtning kan till exempel vara kommunikation med databas eller att hämta data från en annan server. I applikationen syftar model på källkoden som hanterar databaskommunikation.
- View innebär all kod som hanterade strukturen av det grafiska interfacet. I webbutveckling och projektet syftar detta på HTML, CSS och Javascriptfiler som visas för användaren.
- Controller binder samman funktionaliteterna från view och model för att visa hämtade data i ett grafiskt interface. I projektet motsvarade controller de olika routes som finns i webapplikationen.



Figur 4-1 I applikationsstrukturen motsvarar "Routes" Controller, "Views" motsvarar "Views" och "Db" motsvarar "model" i MVC strukturen

4.3.2 Node.JS

Node.JS används som bas för webapplikationen då det är gjort för nätverksbaserade applikationer. [32] Node.JS kommer med grundfunktionalitet för att skapa en applikation. För att utöka funktionaliteten ytterligare finns det tredjepartspaket att hämta från npmjs.com. I applikationen används paketen express, express-handlebars, pg och socket.io.

4.3.3 Paket

Här beskrivs de olika paketen till Node.JS djupare.

4.3.3.1 Express

Express är ett webframework anpassat för Node.JS för att förenkla hanteringen av webförfrågningar (http requests) till en webapplikation. Express möjliggör uppdelning av routes samt smidig hantering av olika typer av förfrågningar. Express var det naturliga valet för att skapa en webserver då det är anpassat för Node.JS. [31]

4.3.3.2 Express-Handlebars

Express-Handlebars (Handlebars) är en visnings motor (view engine) som används för att skapa dynamiskt innehåll på websidor. Express-handlebars valdes då det var en minimalistisk version av HandlebarsJS som innehöll mer funktionalitet än vad som behövdes. [22]

4.3.3.3 PG

PG skapar funktionalitet för att Node.JS ska kunna kommunicera med en PostgreSQL-server. Då all data från Pythonapplikationen sparas i en PostgreSQL-databas behöver Node.JS kommunicera med denna. PG valdes då den har hög stabilitet samt att den är ett populärt val för programmering med PostgreSQL enligt npmjs.com. Alternativen till PG på npmjs.com är baserade på PG vilket gjorde att PG var ett naturligt val. [21]

4.3.3.4 Socket.io

Socket.io används för realtidskommunikation mellan server och klient med hjälp av websockets. Realtidsuppdatering används för att uppdatera filter i webapplikationen, mer om filter i 4.3.9. Socket.IO är stabilt och populärt vilket gör att det ofta används till realtidsuppdatering, se npmjs.com. [23]

4.3.4 Olika typer av http-förfrågningar

Applikationen använder sig av två olika typer av http-förfrågningar, POST och GET. POST-förfrågningar används för att uppdatera, radera eller lägga till data i databasen. GET-förfrågningar används för att hämta sidor samt data ur databasen.

4.3.5 Generering av sidor genom Node.JS med express

Node.JS behövde en visningsmotor för att generera dynamiska sidor, i projektet används Handlebars till detta. Handlebars använder sig av mallar och data från databasen för att generera sidor. I en mall används dubbla klammerparenteser för att ange platser där data skulle appliceras. Till exempel så kan ”{{stad}}” motsvara data innehållandes städer. Handlebars behöver en filtyp för sina mallar, denna filtypen kunde väljas fritt och i projektet valdes filtypen HTML då sidorna var skapade i HTML-kod. När Handlebars applicerar data till mallen skickas den genererade sidan till klientens webbläsare. Nedan visas ett exempel på detta.

Exempel 2

Generering av sidor med Handlebars.

template.html:

Mallen för Handlebars

```
<p>{{Namn}}</p>
```

Variabel skickas med i routes.js:

Mall och data anges för att skapa sidan.

```
res.render(”template”,{”Namn”:”Nikolai”});
```

Sidan som skickas till klienten blir då:

```
<p>Nikolai</p>
```

4.3.6 Sidornas syfte

Fem sidor skapades för att visa och uppdatera data.

4.3.6.1 Reports

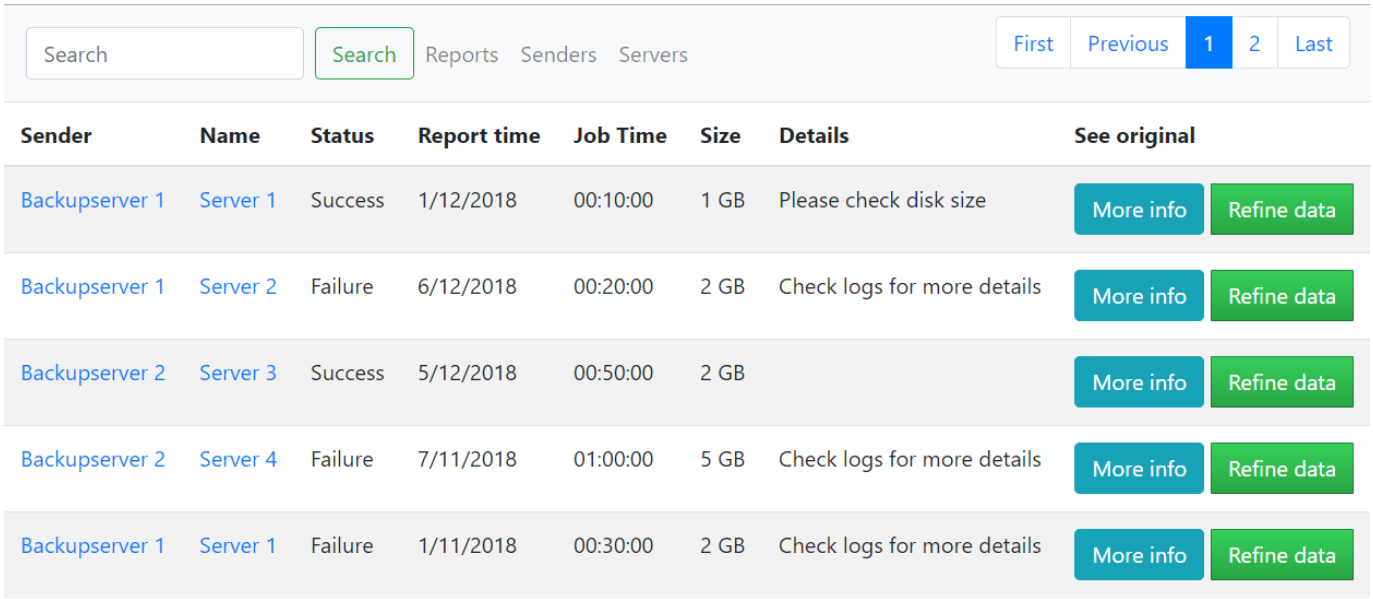


Figure 4-3-Vy över alla rapporter

I figuren visas en överblick över alla rapporter med tillhörande data. Det går även att söka och sortera rapporterna för att användaren ska kunna hitta rapporten de letade efter.

4.3.6.2 Backupservrar

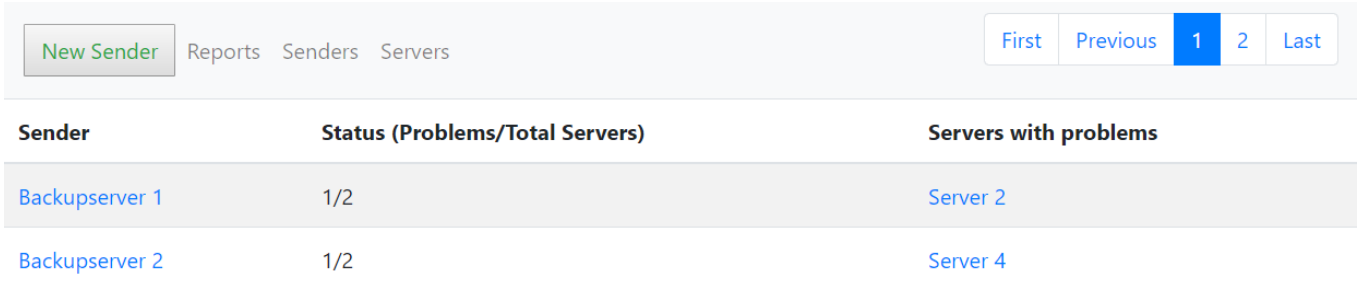


Figure 4-4-Vy över backupservrar

I figuren visas en överblick över alla backupservrar som data samlas från. Genom att välja en backupserver har användaren möjlighet att se rapporter från endast den valda servern.

4.3.6.3 Servrar

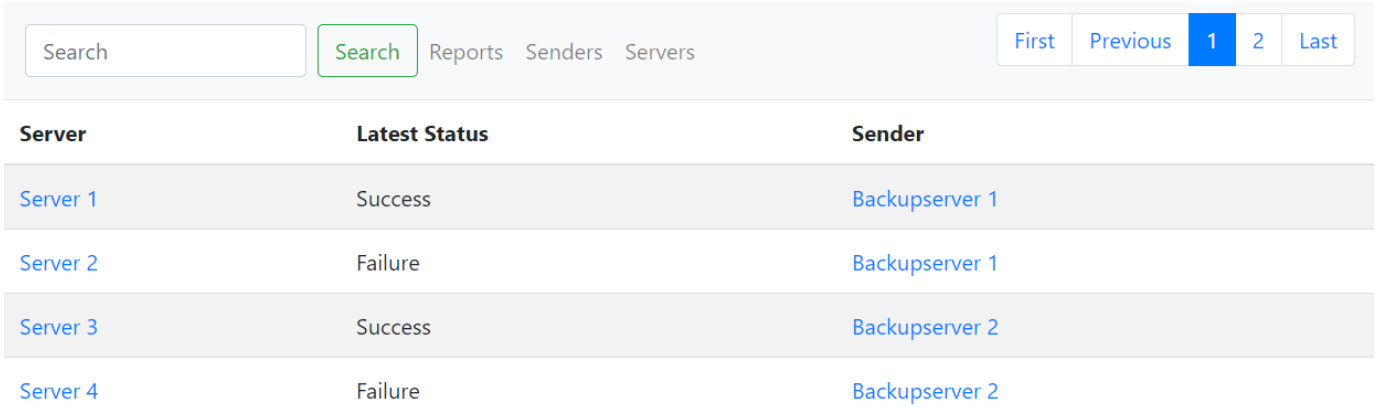


Figure 4-5-Vy över alla servrar som backats upp

I figuren visas namnen och status för varje system som backats upp.

4.3.6.4 Refine Data

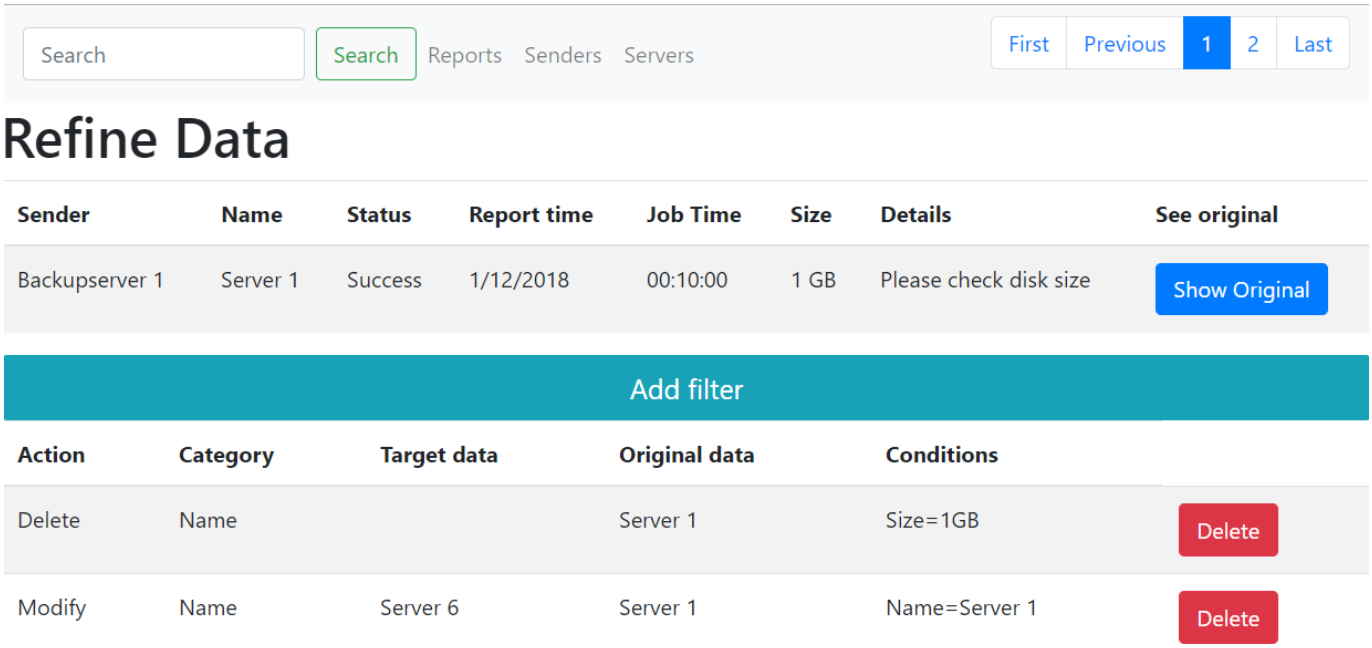


Figure 4-6-Vy för att modifiera och förfina data

Här har användaren möjlighet att förbättra den insamlade data genom att skapa filter och anpassa data efter dess behov.

4.3.6.5 New sender

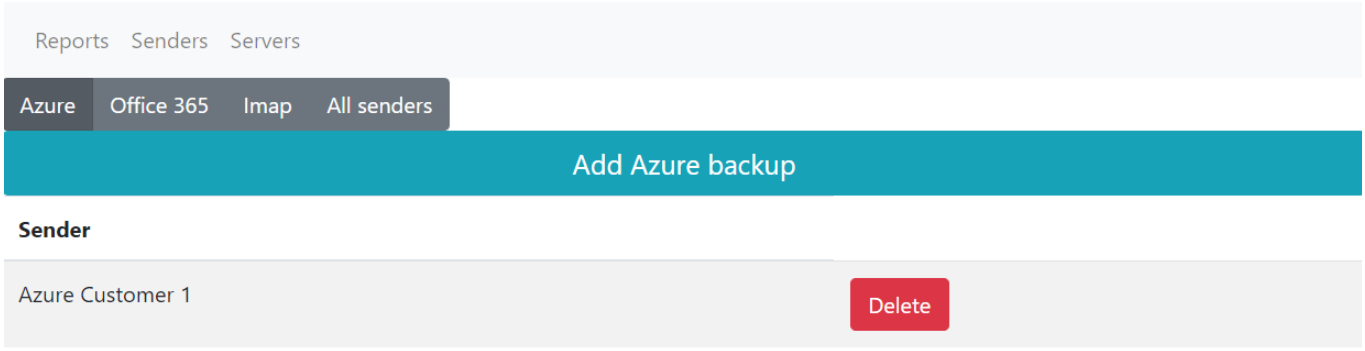


Figure 4-7-Vy för att lägga till fler källor till backupdata

Här kan användaren se över sina mailkonton och Azure-konton samt lägga till fler om behovet finns

4.3.7 Flöde från backend till frontend

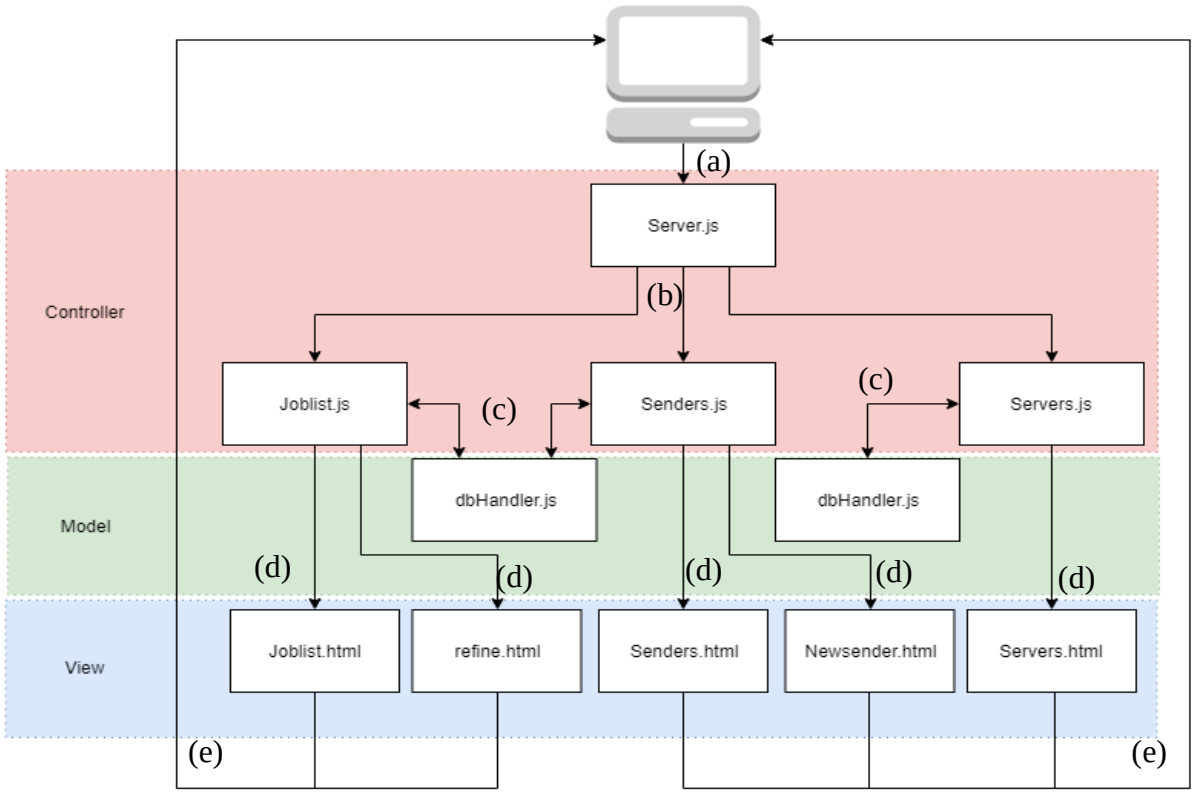


Figure 4-2-Flöde från förfrågan till svar

a) Klienten skickar en förfrågan efter en sida. Server.js tar emot förfrågan
b) Server.js delegerar förfrågan till den routen som har hand om den förfrågade sidan
c) Routen hämtar relevant data från databasen via databashanteraren (dbhandler.js)
d) Routen genererar en hemsida med hjälp av motsvarande html template.
e) Den genererade sidan skickas till klienten och visas för användaren

4.3.8 Design av hemsidorna

Då projektet innebär att visa upp data på ett enkelt överskådligt sätt ansågs en tabell vara den bästa strukturen för att visa data. Varje kolumn innebär en kategori av data och varje rad en instans av data.

Skapandet av designen till hemsidorna förenklades genom att använda designbiblioteket ”Bootstrap”. Designbiblioteket innehåller verktyg och material för att enkelt kunna göra responsiva och anpassade sidor. [24] Förutom att visa backupdata användes Bootstrap för att skapa allt visuellt på sidorna.
Det var viktigt att data skulle vara enkelt att hitta på hemsidorna. För att lösa detta implementerades sökfunktioner samt sorteringsfunktioner av backupjobben, detta gjordes i javascript.

4.3.9 Användarfilter

Då metoden för att extrahera data i nuvarande lösning inte är perfekt förekommer skräpdata i den insamlade datan. För att kunna sortera ut denna data implementerades en filterfunktion där användare kan skapa filter för att ändra eller ta bort data baserat på om angivna villkor uppfylls eller ej. Detta filter kallas även ”data refinement”. Detta kan även användas för att modifiera data om till exempel användaren vill ha ett ”A” med när data inkluderat siffran ”1”, då skulle alla förekomster av en etta blivit ”A1” eller ”1A”.

För att skicka informationen om att ett nytt filter ska skapas implementerades realtidskommunikation med Socket.IO. När ett filter skapas fyller användaren i fälten för ett filter och trycker sedan på ”apply”. Informationen om filtret skickas via websockets till backenden vilket gör att användaren inte behövde lämna sidan för att skapa fler filter.

5

Resultat

Två olika applikationer utvecklas för att hantera backupdata. En pythonapplikation för att hantera backupdata från mail och backupservrar och en webapplikation för att skapa ett webinterface där administratörerna kan läsa rapporterna.

5.1 Sidor som utvecklades för översikt av backups

Fem sidor utvecklades för att ge en god översikt över backups

5.1.1 Lista över alla backupjobb

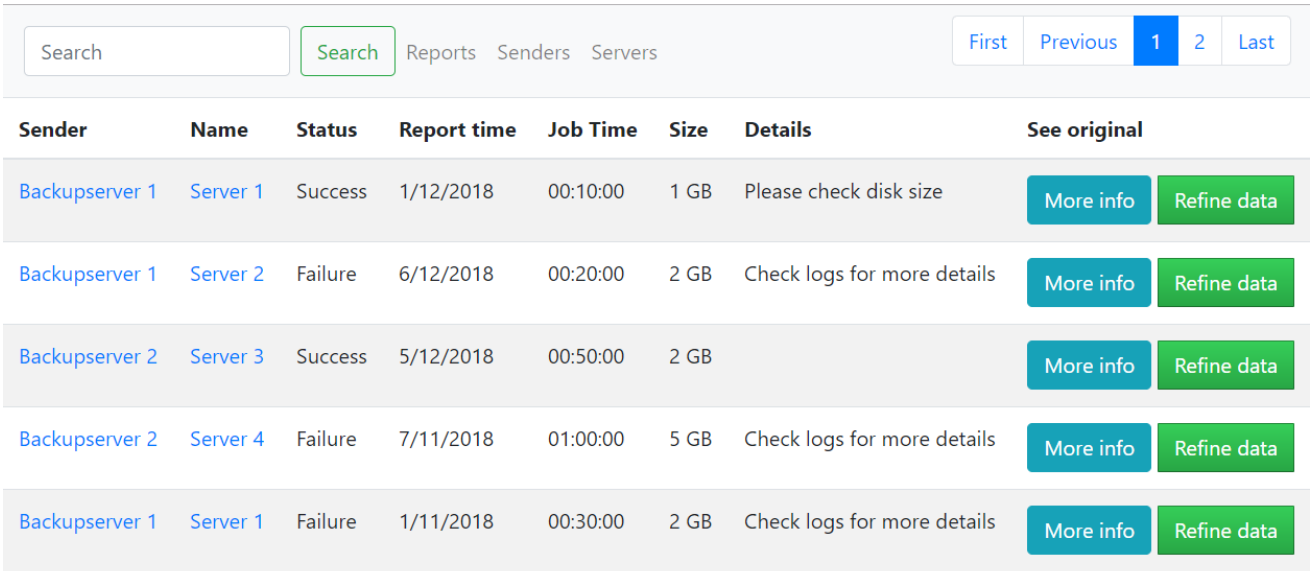


Figure 5-1-Vy över alla rapporter

I detta interfacet listas statusen hos olika backups med diverse information. Report time syftar på tiden då statusen rapporterades och job time syftar på tiden det tagit att backa upp servern. Det finns även en knapp som möjliggör funktionalitet för att visa det mail statusen kom ifrån samt en knapp för att modifiera data. Administratören kan även trycka på varje Server och Backupserver (sender) för att se alla statusuppdateringar från dessa.

5.1.2 Lista över alla servrar som backas upp

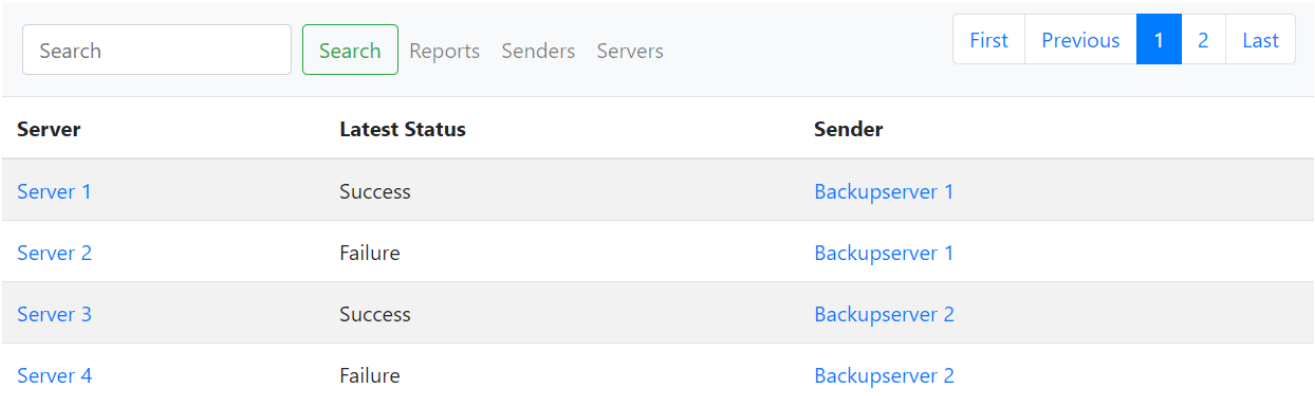


Figure 5-2-Vy över all servrar som backats upp

I detta interfacet listas alla servrar som backats upp och dess senaste status. Administratören kan även trycka på varje Server och Backupserver (sender) för att se alla statusuppdateringar från dessa.

5.1.3 Lista över alla backupservrar som skickar rapporter

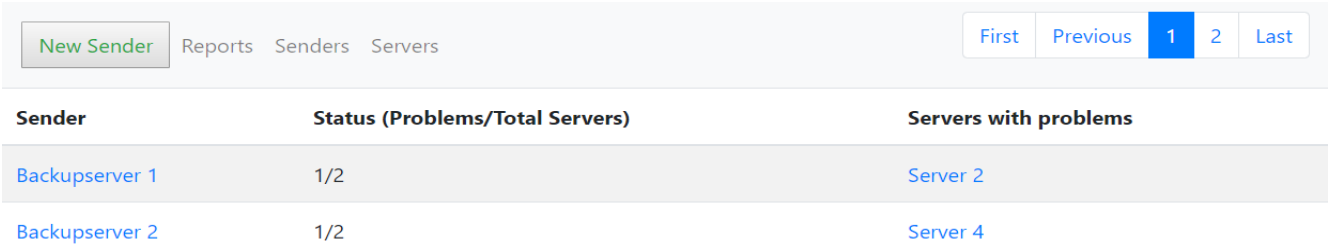


Figure 5-3-Vy över alla backupservrar

I detta interfacet listas backupservers (senders) samt en sammanfattning av statusen hos servarna den backar upp.

5.1.4 Interface för att hantera backupsystem att hämta data från

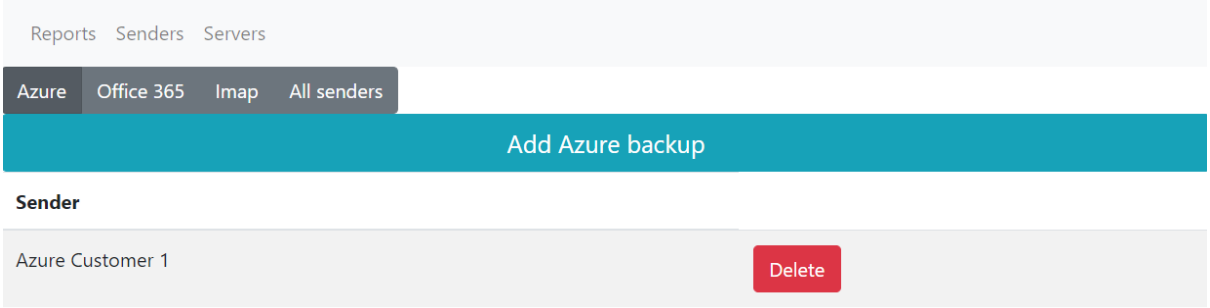


Figure 5-5-Vy för att lägga till fler källor till backupdata

I detta interfacet visas alla konton applikationen var kopplad till, här kan även fler konton läggas till.

5.1.5 Interface för att skapa filter för att modifiera data

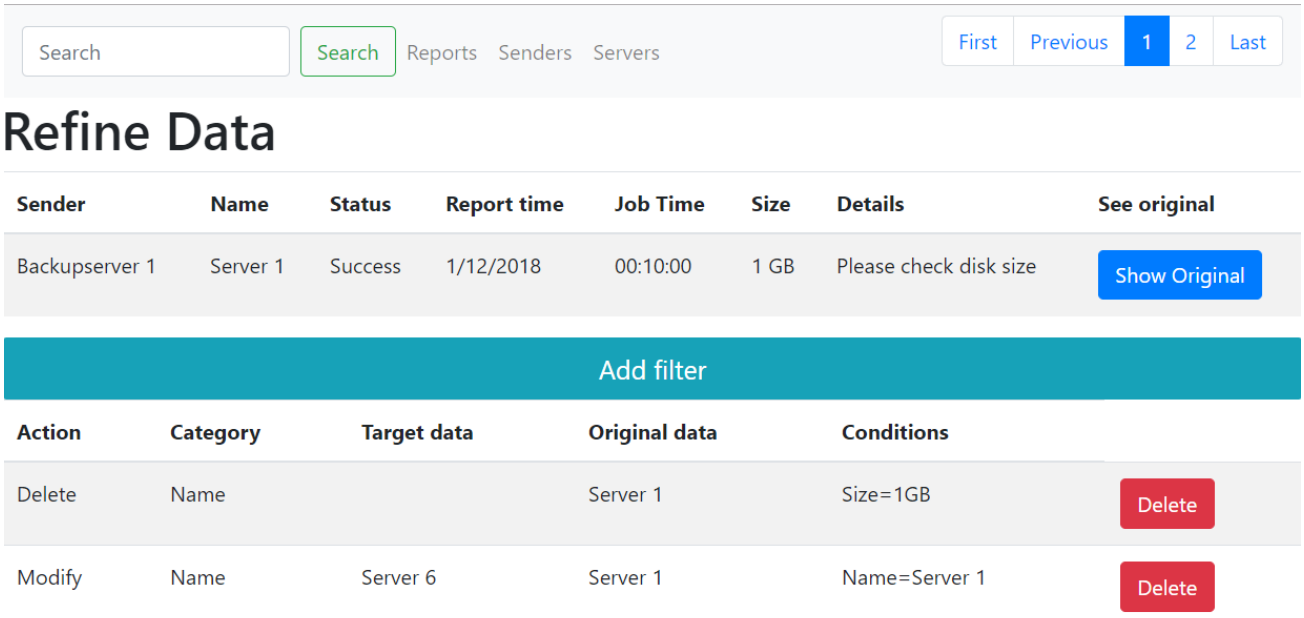


Figure 5-4-Vy för att modifiera och förfina data

I detta interfacet visas olika filter som används för att modifiera existerande data. Administratören kan här välja data den data som skulle modifieras eller raderas. För att förenkla skapandet av filtret visas data som användaren valt högst upp på sidan. I figuren har ett deletefilter skapats som raderar all data där storleken är exakt 1 GB. Figuren innehåller även ett modifikationsfilter som ändrar namnet ”server 1” till ”server 6”.

5.1.6 Slutord kring resultat

Båda applikationerna resulterade i fungerade prototyper. Ingen data samlades in kring huruvida applikationerna effektiviserade administratörernas arbete då inte tillräckligt med tid fanns för detta. Diskussionen kring prototypen med Consats anställda gav positivt gensvar.

6

Diskussion

6.1 Generell diskussion

Innan projektets arbete och innehåll diskuteras hålls här en diskussion kring de generella lärdomarna kring projektets uppstart och utförande.

6.1.1 Uppstart av projektet

En projektgrupp bestående av endast en person visade sig vara komplicerat och ineffektivt. Det var viktigt att kunna diskutera fram lösningar annars var det väldigt enkelt att fastna vid uppkomna problem. Om projektet hade startats på nytt hade en projektgrupp på minst 2 personer varit ett krav för ett lyckat projekt.

Consats anställda agerade bollplank i generella designidéer vilket hjälpte projektet framåt. De hjälpte även till att hålla motivationen uppe genom att vara gemytliga och villiga att diskutera olika frågor.

6.1.2 Projektets omfattning

Vid starten av projektet diskuterades ett grundläggande syfte av produkten med Consat, presenterat under rubriken ”syfte”. Detta gav projektet en bred utgångspunkt med många val att göra. Tidsplanen lades upp och projektet såg ut att vara välstrukturerat. Efter mer än halvvägs in i projektet upptäcktes detta inte vara korrekt estimerat. Alla val som behövdes göras tog längre tid att testa och undersöka än väntat. Detta ledde till att projektet tog längre tid och att en del val gjordes baserat på erfarenhet före funktionalitet.

Då det var många val som behövdes testas och undersökas i projektet var det nödvändigt att mycket tid och fokus lades på detta. Av denna anledningen lades även inget fokus på rapporten för projektet under utvecklingens gång.

6.2 Pythonapplikation

De olika metoderna som användes i pythonapplikationerna diskuteras här.

6.2.1 Olika sätt att extrahera data

När projektet planerades var tanken att data skulle transporteras från backupservrarna till applikationen på tre olika sätt: SNMP, mail och http. När projektet började visade det sig dock snabbt att SNMP inte var ett bra val då backupservrarna inte hade funktionalitet för att överföra sina rapporter via SNMP. Dock så implementerades grundfunktionalitet för SNMP i början av projektet. Detta tyder starkt på att den förstudie som gjordes innan planeringen inte var tillräcklig för att skapa en bra planering.

6.2.2 Hämta data via mail

Att hämta mail från Office 365 förväntades ta mycket lång tid till att testa och implementera men tack vare välskriven dokumentation tog detta endast en dag.

I brist på erfarenhet av RFC-standarder tog det längre tid att läsa, testa och implementera IMAP än väntat. Att läsa och lära sig strukturen av RFC standarder visade sig vara en god kunskap då de flesta av Internets standarder är definierade i dessa.

6.2.3 Extrahering av data från mail

Tidigt i projektet blev det tydligt att det generella sättet att extrahera data från backupservrarna var via mail som skickades när en backup gjorts. Alla mail såg olika ut men innehöll viktiga data som Consat ville kunna avläsa. Det enda sättet för att extrahera data var att hitta ett generellt sätt att tolka mailen trots olikheterna.

Här stod projektgruppen inför valet att använda en tredjeparts textparser eller skapa en egen för att implementera funktionaliteten. Vid detta valet tog nyfikenheten över och valet att skapa en egen textparser gjordes. Detta visade sig vara ett dåligt val då det krävdes en hel del tid och skicklighet för att lyckas testa och implementera en parser. En viktig lärdom från detta är att det inte är effektivt att försöka återuppfinna hjulet trots att det verkar intressant.

Även om det tog väldigt mycket tid att utveckla textparsern så var det mycket givande i form av kunskap.

6.2.4 Extrahering av data från Azure via REST

De backupservrar som inte använde sig av mail var de som låg på Azure. För att kunna hämta data om Azures backups hittades metoder för att kommunicera med deras servrar. Innan kommunikation med Azures servrar kunde implementeras testades det. Azure har en dokumentation för detta men det var svårt att navigera i den och att hitta precis rätt information. Detta gjorde att testningen och implementationen tog en vecka att bli klar. Det var dock väldigt lärorikt att se hur olika dokumentationer kan vara strukturerade och hur kommunikation med Azure kan implementeras via REST.

6.2.5 Säkerhet

Ungefär halvvägs in i projektet upptäcktes det att det var ont om tid för att implementera grundfunktionaliteten. Därför gjordes valet att man inte skulle fokusera på säkerheten för applikationerna. Om projektet skulle göras om skulle mer fokus läggas på säkerheten då backupstatus kan vara känslig information.

6.3 Webapplikation

Här diskuteras valen som gjordes kring webapplikationen.

6.3.1 Applikationens struktur

MVC-mönster i Node.JS applikationer kom väldigt naturligt vilket var oväntat. Mönstret gjorde att flödet från request till response blev enkelt att följa och gjorde hela applikationen enkel att strukturera.

6.3.2 Paketen

Att hitta paket till Node.JS visade sig inte vara några svårigheter. Det visade sig dock att det fanns många paket med samma funktionalitet. Urvalet av paketen gjordes beroende på hur populära de var på npmjs.com eftersom hög popularitet tyder på ett välfungerande och stabilt paket. Detta var dock inte en pålitlig metod och ett exempel på detta är att det paket som används för att skapa dynamiska data till hemsidan (express-handlebars) var populärt men det fanns ytterligare ett paket som var ännu mer populärt och passade bättre för funktionaliteten som efterfrågades. Detta paketet hittades inte i förstudien vilket är en stark indikation på att förstudien inte var tillräckligt bred eller djup.

6.3.3 Design

Designen av sidorna gjordes med hjälp av biblioteket Bootstrap. Då andra delar av projektet tog för lång tid blev valet av design och verktyg för design ett snabbt val. Eftersom projektgruppen hade erfarenhet av Bootstrap var valet enkelt. Tyvärr så spenderades inte tillräckligt mycket tid på att planera och bestämma hur utseendet av hemsidorna skulle se ut och hur användaren skulle interagera med denna. Detta ledde till att webinterfacet inte blev så intuitivt men fyllde sin funktionalitet.

6.4 Miljö och hållbarhet

Backups behövs för att enkelt kunna återställa servrar vid fel. Om en backup misslyckas kommer backupservern försöka backa upp igen vilket resulterar i extra använd energi.

Kärnkraftverk och reningsverk styrs av datorer och servrar. Om inte backups finns för dessa kan en serverkrasch orsaka stora problem för miljö och människor. Därför är det viktigt att ha god översikt över backups vilket detta systemet ger.

6.5 Kritisk diskussion

Här reflekteras över vad som hade gjorts annorlunda om projektet startats om.

6.5.1 Planering

Mycket av tiden av projektet gick åt till att undersöka Consats behov och undersöka de existerande systemen, något som kunde förbättras med en djupare förstudie innan planeringen började. Ett exempel på förbättring är att göra en behovsanalys av vad Consat har för exakta krav på systemet. Efter behovsanalysen skulle det behövas göras en inventering tillsammans med anställda på Consat för att se vad för system som finns och vad för egenskaper dessa innehar. Först efter det skulle en ordentlig planering kunna läggas upp.

6.5.2 Användning av tredjepartskod

Något som tog väldigt lång tid var att i flera fall återuppfinna hjulet genom att skriva komplicerad kod som redan fanns att hitta hos tredje part. Detta var extremt ineffektivt och ledde delvis till att tiden för projektet inte räckte till. Den tiden skulle istället kunna använts till att hitta och implementera bättre lösningar för att uppnå samma funktionalitet.

6.5.3 Databas

Vid valet av databas missades en egenskap hos databasen, flyttbarhet. Node.JS är en modulär mjukvara som enkelt går att flytta från server till server. Detta är svårt med PostgreSQL som databas. För att fylla egenskapen skulle till exempel SQLite, som sparar sin data i flyttbara filer, kunna användas.

6.6 Slutsats

Många olika metoder för olika funktionaliteter identifierades under projektets gång vilket tog längre tid att hantera än väntat. Detta gav mycket mer kunskap om varje funktionalitet samt mer insikt i applikationer av större skala.

I slutet av projektet hölls ett möte med Consats anställda där de fick möjlighet att ge sina åsikter om systemet. Det generella intrycket från mötet var att webinterfacet fyller Consats behov.

7
Källor

1. Debian -- About Debian [Internet]. [cited 2019 Jan 16]. Available from: https://www.debian.org/intro/about
2. SciPy.org — SciPy.org [Internet]. [cited 2019 Jan 20]. Available from: https://www.scipy.org/
3. The Web framework for perfectionists with deadlines | Django [Internet]. [cited 2019 Jan 20]. Available from: https://www.djangoproject.com/
4. Python Software Foundation. Pip-PyPi [Internet]. Available from: https://pypi.org/project/pip/
5. V8 JavaScript engine [Internet]. [cited 2019 Jan 19]. Available from: https://v8.dev/
6. Npm Inc. Node.JS Package Manager [Internet]. Available from: https://www.npmjs.com/
7. GitHub. Atom-GitHub [Internet]. Available from: https://github.com/atom
8. GitHub. Atom Packages [Internet]. Available from: https://atom.io/packages
9. Git [Internet]. [cited 2019 Jan 20]. Available from: https://git-scm.com/
10. Massé M. REST API design rulebook. O'Reilly; 2012. 94 p.
11. Get access tokens to call Microsoft Graph - Microsoft Graph | Microsoft Docs [Internet]. [cited 2019 Jan 20]. Available from: https://docs.microsoft.com/en-us/graph/auth-overview
12. Schwaber K, Sutherland J. The Scrum Guide™ The Definitive Guide to Scrum: The Rules of the Game [Internet]. 2017 [cited 2018 Nov 28]. Available from: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100
13. Murugaiyan D. International Journal of Information Technology and Business Management WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. 2012 [cited 2018 Nov 28];2[1]. Available from: www.jitbm.com
14. Natural Language Toolkit — NLTK 3.3 documentation [Internet]. [cited 2018 Sep 17]. Available from: https://www.nltk.org/
15. TextBlob: Simplified Text Processing — TextBlob 0.15.1 documentation [Internet]. [cited 2018 Sep 17]. Available from: https://textblob.readthedocs.io/en/dev/
16. 3.5.6 Documentation [Internet]. [cited 2018 Sep 17]. Available from: https://docs.python.org/3.5/
17. <mrc@cac.washington.edu> MRC. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. [cited 2018 May 16]; Available from: https://tools.ietf.org/html/rfc3501

18. 21.15. imaplib — IMAP4 protocol client — Python 3.5.6 documentation [Internet]. [cited 2018 Sep 17]. Available from: https://docs.python.org/3.5/library/imaplib.html

19. Overview of Microsoft Graph - Microsoft Graph | Microsoft Docs [Internet]. [cited 2019 Jan 20]. Available from: https://docs.microsoft.com/en-us/graph/overview

20. Recovery Services | Microsoft Docs [Internet]. [cited 2019 Jan 21]. Available from: https://docs.microsoft.com/en-us/rest/api/recoveryservices/

21. Carlson B. PG-Node [Internet]. Available from: https://www.npmjs.com/package/pg

22. Ferraiuolo E, Yalkabov S. Express-Handlebars [Internet]. Available from: https://www.npmjs.com/package/express-handlebars

23. Arrachequesne D. Socket.io - NPM [Internet]. Available from: https://www.npmjs.com/package/socket.io

24. Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. [Internet]. [cited 2018 Sep 17]. Available from: https://getbootstrap.com/

25. 19.2. json — JSON encoder and decoder — Python 3.5.6 documentation [Internet]. [cited 2019 Jan 21]. Available from: https://docs.python.org/3.5/library/json.html

26. 20.2. html.parser — Simple HTML and XHTML parser — Python 3.5.6 documentation [Internet]. [cited 2019 Jan 21]. Available from: https://docs.python.org/3.5/library/html.parser.html

27. 6.3. difflib — Helpers for computing deltas — Python 3.5.6 documentation [Internet]. [cited 2019 Jan 21]. Available from: https://docs.python.org/3.5/library/difflib.html

28. Palme J. Common Internet Message Headers. [cited 2018 Sep 24]; Available from: https://tools.ietf.org/html/rfc2076

29. Psycopg2 Tutorial - PostgreSQL wiki [Internet]. [cited 2019 Jan 21]. Available from: https://wiki.postgresql.org/wiki/Psycopg2_Tutorial

30. Broberg N. Model-View-Controller [Internet]. [cited 2019 Jan 21]. Available from: http://www.cse.chalmers.se/edu/year/2017/course/DIT952/slides/6-1a - Model-View-Controller.pdf

31. Express - Node.js web application framework [Internet]. [cited 2019 Jan 22]. Available from: https://expressjs.com/

32. Node.JS – About Node.JS [Internet]. [cited 2019 Feb 20]. Available from: https://nodejs.org/en/about/

33. Websocket - NPM [Internet]. Available from: https://www.npmjs.com/package/websocket

34. Bird S Klein E Loper E Natural language processing with Python [Internet]. 2009 [cited 2019 Mar 10]. Available from: https://books.google.se/books?id=KGIbfiiP1i4C&lpg=PR5&ots=Y3Dhy3ODO8&dq=text%20handling%20python&lr&hl=sv&pg=PP#v=onepage&q=text%20handling%20python&f=false