CHALMERS

Visualisering av systemintegrationers driftstatus över tid

Examensarbete inom Data- och Informationsteknik

JOHANNES UHR

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Visualisering av integrationers driftstatus över tid

JOHANNES UHR

Examinator: Christer Carlsson

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik

Göteborg 2016

FÖRORD

Rapporten beskriver ett examensarbete (15 hp) som utfördes på helfart över 10 veckor under vårterminen 2016. Arbetet gjordes som det sista momentet i utbildningen Datateknik (högskoleingenjör, 180 hp) på Chalmers Tekniska Högskola.

Jag vill ge ett stort tack till samtliga anställda på e-man AB:s huvudkontor i Örebro som funnits till stöd under arbetets gång. Extra stort tack till Joakim Restadh och Sofia Sundqvist som trodde på mig och gav mig den här uppgiften.

Jag vill också ge ett stort tack till Joachim von Hacht som har hjälpt mig med utformandet av rapporten.

SAMMANFATTNING

e-man AB är ett företag med huvudkontor i Örebro som specialiserar sig inom integration av IT-system. Företaget har ett serviceavtal mot sina kunder där de utlovar en viss aktiv drifttid för varje integration. Ett verktyg saknas för att visualisera om dessa avtal upprätthålls.

Syftet med projektet var att ta fram en lösning som sammanställer och visualiserar integrationers drifttid. Arbetet har utförts på e-mans huvudkontor i Örebro med stöd från anställda på företaget. Resultatet blev en webbapplikation som uppfyller projektets syfte.

Rapporten beskriver hur denna lösning togs fram. Från utbyggnad av ett äldre system, till utnyttjandet av molntjänster och slutligen utvecklandet av en webbapplikation som visualiserar integrationers driftstatus över tid.

Nyckelord: webbapplikation, systemintegration, Amazon Web Services, angularJS

ABSTRACT

e-man AB is a company based in Örebro that specializes in system integrations. The company signs a service-level agreement with their customers in which they promise a monthly running uptime for their integrations. e-man lacks a tool to visualize if these service-level agreements are kept.

The purpose of the project was to develop a solution which visualizes running uptime of integrations. The work has been done at e-man's office in Örebro with help from the employees. The result is a web application which fulfills the purpose of the project.

This report describes how this solution was developed. From an expansion of an existing system, to the usage of cloud services and lastly a web application that visualizes integrations' running uptime over time.

Keywords: web application, system integration, Amazon Web Services, AngularJS

BETECKNINGAR

AJAX	Asynchronous JavaScript and XML, tekniker för utveckling av asynkrona webbapplikationer.
AWS	Amazon Web Services, plattform för molntjänster.
MVC	Model-View-Controller, designmönster.
NoSQL	Not only SQL, databastyp.
SNS	Simple Notification Service, meddelandetjänst på AWS.

Innehåll

1 Inledning

1.1 Bakgrund

Ett informationssystem är en IT-lösning som används till att samla in, skapa och bearbeta information. Informationssystem används av företag och organisationer till att strukturera, effektivisera och övervaka sin verksamhet och på så sätt bli mer konkurrenskraftiga.

En vanlig typ av IT-system är affärssystem. Ett affärssystem används av företag till att koppla samman produkter, fabriker, lagersaldon och ordrar. Företag kan då organisera och dela information om sina ordrar, lagersaldon och produktion utan att kommunicera manuellt.

Systemintegrering innebär att en lösning utvecklas som får flera fristående informationssystem att utbyta information och arbeta tillsammans. Störst behov av detta har företag som använder flera system i sin verksamhet där de måste föra över information mellan de olika systemen manuellt.

Ett vanligt scenario är ett företag som har ett system för att skapa ordrar och ett annat för att fakturera sina kunder. Det finns då ett behov att integrera systemen med varandra så att avslutade ordrar automatiskt genererar en faktura. Ett exempel på en integration i detta fallet är att skapa en applikation som läser in färdiga ordrar från ett system, bearbetar datan och lägger den i en kö som ett faktureringssystem läser från.

e-man AB är ett IT-företag som specialiserar sig inom integration av informationssystem. e-mans huvudkontor ligger i Örebro, övriga kontor finns i Västerås, Stockholm och Göteborg.

Företaget har konsulter med stor erfarenhet inom systemintegration. Konsulterna arbetar med att ta fram integrationslösningar åt olika kunder. e-man har också anställda som driftövervakar sina integrationer.

e-mans kunder omfattas av ett serviceavtal, Service-level Agreement. En vanlig specifikation på ett serviceavtal är att en integration ska ha körts felfritt under minst 99.6% av tiden under varje månad.

För närvarande saknar e-man verktyg för att tydligt se hur kundernas integrationer har fungerat över tid. Företaget vill också ha möjlighet att se om de krav som specificerats i serviceavtalen har uppfyllts.

1.2 Syfte

Syftet med arbetet är att ta fram en lösning som sammanställer och visualiserar integrationers driftstatus. Lösningen ska tydligt visa om integrationerna för varje månad uppfyllt serviceavtalen för respektive kund.

1.3 Mål

Projektets mål är att att utveckla en webbapplikation som analyserar och visualiserar den driftinformation som över tid skickas från e-mans integrationer.

1.4 Avgränsningar

För att avgöra om systemet tydlig förmår att visa integrationernas driftstatus per månad måste systemet köras under ett antal veckor. Tiden medger inte att detta ingår i arbetet. En skattning av systemets förmåga kommer därför göras.

2 Metod

2.1 Förkunskaper

Arbetet kräver grundläggande kunskaper om hur globala molntjänster används eftersom dessa kommer att ligga till grund för projektet. Instuderingen av molntjänster kommer att ske utifrån hur e-man använder dessa. Därutöver kommer introduktionskurser att genomgås och studier av API:er att göras.

2.2 Arbetssätt

Arbetet kommer delas upp i tre iterationer:

1. Under den första iterationen kommer e-mans nuvarande system för övervakning av integrationer att modifieras till att skicka information till molntjänster.
2. Den andra iterationen omfattar en implementation av molntjänster för att ta emot, hantera och spara den data som skickas från e-mans system.
3. Under den tredje iterationen kommer en webbapplikation att utvecklas med syftet att visualisera den ackumulerade datan.

2.3 Testning

För att testa lösningen kommer en lokal integrationsmiljö att sättas upp som liknar den miljö kunderna kör sina integrationer på. Testningen kommer utföras manuellt genom att starta, stoppa och störa integrationer på sätt som speglar en verklig kundmiljö.

3 Teknisk bakgrund

3.1 Java

Java är ett objektorienterat, statiskt typat, programmeringsspråk. Språket används i stor utsträckning till webbapplikationer, mobilapplikationer och komplexa IT-system. Koden kompileras till bytekod och exekveras på en virtuell maskin. Detta gör att språket blir oberoende av plattform [1].

3.2 JavaScript

JavaScript är ett dynamiskt typat, objektorienterat skriptspråk. JavaScript används främst i webbläsare för att göra webbsidor dynamiska och interaktiva. Språket har på senare tid använts till webbapplikationers serversida [2].

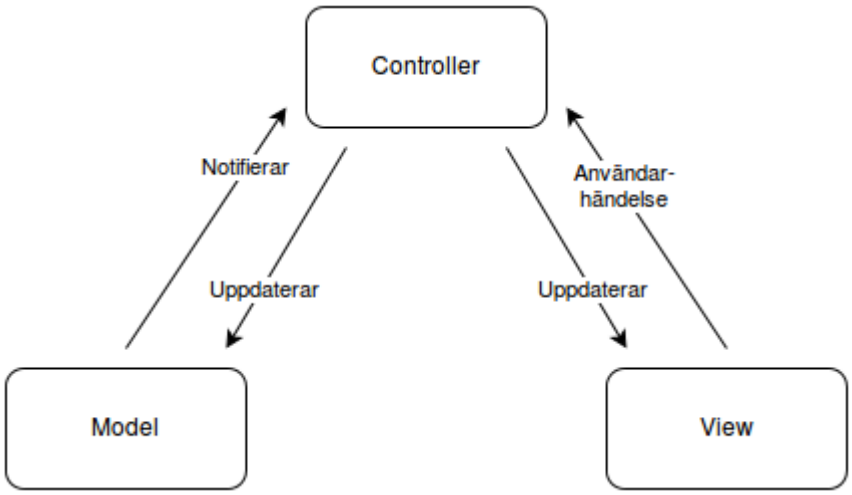
3.3 AJAX

AJAX (eng. Asynchronous JavaScript and XML) är ett namn för en samling tekniker som kan användas till att göra asynkrona webbapplikationer. En populär teknik som tillhör AJAX är XMLHttpRequest-objektet. Detta objekt kan användas av JavaScript på en webbsida för att anropa en webbserver. Anropet kommer att ske utan att webbsidan laddas om [3].

3.4 MVC

Model-View-Controller, MVC, är ett designmönster där utvecklaren separerar det grafiska gränssnitt (view) som användaren interagerar med och den data (model) som programmet bearbetar.

Den komponent som hanterar kommunikationen mellan model och view kallas controller. Controllerns uppgift är att förse vyn med data från modellen och när användaren interagerar med gränssnittet ser controllern till att modellens tillstånd ändras [4]. För övergripande struktur av MVC, se figur 3.1.



Figur 3.1: Övergripande struktur av MVC.

3.5 Ramverk

Ett ramverk är en struktur som finns till hjälp vid utveckling av mjukvara. Strukturen är ofta uppdelad i lager som indikerar hur ett program ska utvecklas. Ramverk ger tillgång till färdig kod, API:er och regler för hur dess API:er ska användas [5].

Idag finns ramverk för de flesta områden av mjukvaruutveckling. Inom webbapplikationer finns ramverk för både server som klientsida. Normalt används alltid något ramverk för utveckling av webbapplikationer.

Två exempel på ramverk för klientsidan är Bootstrap och AngularJS. Bootstrap har tagits fram av Twitter till front-end webbutveckling. Ramverket används till att designa och ge webbsidor ett fördefinierat enhetligt utseende. AngularJS är ett ramverk som Google tagit fram, se vidare kapitel 3.7.

3.6 Öppen källkod

Öppen källkod, (eng. open source), innebär att ett programs källkod är fri för allmänheten att läsa, använda och bygga vidare på. För programmerare innebär detta att man kan återanvända kod utan att betala licenskostnad till ägaren [6].

3.7 AngularJS 1

AngularJS är ett ramverk med öppen källkod för klientsidan av en webbapplikation. Ramverket baseras på JavaScript och har tagits fram av Google. All kod exekveras på klientsidan i webbläsaren [7].

Genom att erbjuda en MVC-arkitektur kan utvecklaren använda en utökad HTML-syntax för att visa upp webbapplikationens datamodell i användargränssnittet. Ramverket använder databinding vilket innebär att när ändringar sker i modellen via controllern visas de upp i vyn automatiskt genom en variabel som heter scope. Figur 3.2 visar ett kodexempel på hur en controller förser vyn med data genom en scope-variabel.

```
<html>
  <head>
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.5/angular.min.js"></script>
    <script/>
      angular.module('myApp', [])
      .controller('myController', ['$scope',
        function($scope){
          $scope.myMsg = "Hello world"
        }
      ]);
    </script>
  </head>
  <body ng-app="myApp">
    <div ng-controller="myController">
      {{myMsg}} <!-- $scope.myMsg i controllern -->
    </div>
  </body>
</html>
```

Figur 3.2: Exempel på AngularJS och HTML kod.

Under hösten 2014 släpptes AngularJS 2.0 vilket är en helt ny version av Angular som skiljer sig markant från tidigare versioner. AngularJS 2.0 är inte kompatibel med tidigare versioner av AngularJS 1.

3.8 Enterprise Integration Patterns

Enterprise Integration Patterns är en bok skriven av Gregor Hohpe och Bobby Woolf. Boken beskriver flera metoder för utveckling av systemintegrationer och behandlar vanliga problem som kan uppstå [8].

3.9 Apache Camel

Apache Camel är ett ramverk för systemintegrationer som syftar till att göra systemintegrering enklare för utvecklaren genom att bidra med standardiserade metoder för att koppla samman informationsflöden hos it-system.

Ramverket ger tillgång till API:er som kan anropas i Java-kod. Apache Camel bygger på utvecklingsprocesser som beskrivs i *Enterprise Integration Patterns*. Det handlar ofta om att skicka meddelanden, hantera köer och analysera data [9].

3.10 NoSQL

En databas av typen NoSQL använder inte samma struktur som en relationsdatabas för att lagra data. En NoSQL-databas har en dynamisk tabellstruktur och data lagras på en plats i form av större dokument. En relationsdatabas lagrar data i form av rader i en eller flera tabeller.

En NoSQL-databas kan skalas upp horisontellt, vilket innebär att ägaren av databsen kan lägga till fler servrar för att få ökad kapacitet. En relationsdatabas kan i regel bara skalas upp vertikalt, vilket innebär att ägaren måste öka prestandan på den server som kör databasen [10].

Webbapplikationer som behöver stöd för ett stort antal samtidiga användare lämpar sig för NoSQL. Exempel på sådana webbapplikationer är Twitter och Facebook.

Viktiga behov som NoSQL möter är:

- Hantering av ostrukturerad data.
- Stöd för ett stort antal samtidiga förfrågningar.
- Snabbt anpassningsbart till nya krav och uppdateringar.

3.11 Molntjänster

Molntjänster är resurser som finns tillgängliga över Internet i form av exempelvis processorkraft, säkerhet och databaser. Molntjänster används ofta av företag för att slippa underhålla och investera i en egen infrastruktur. Användaren köper tjänster istället för egen hårdvara. Detta innebär normalt att användaren kan anpassa sina resurser och skala upp och ner efter behov.

3.12 Amazon Web Services

Amazon Web Services, AWS, är en plattform för molntjänster. AWS erbjuder tjänster som exempelvis virituella servrar och databaser. AWS tillhandahåller också verktyg för att som kund övervaka de tjänster som används [11].

3.12.1 Simple Notification Service

Simple Notification Service, SNS, är en tjänst på AWS för att skicka meddelanden. SNS använder publish/subscribe-modellen som innebär att en användare först skapar en kanal (eng. topic). En resurs eller användare publicerar meddelanden via dessa kanaler. Meddelanden som publiceras skickas vidare till prenumeranter på respektive kanal [12]. En prenumerant kan exempelvis vara en webbapplikation, en e-mailadress eller en AWS Lambda-service, se vidare kapitel 3.12.3.

3.12.2 Cloudwatch

Cloudwatch är ett övervakningsverktyg som kan nyttjas för att övervaka de resurser som används på AWS. Resurser som övervakas kan exempelvis vara databastabeller och processoranvändning för servrar. Resurserna kommunicerar med Cloudwatch genom att skicka mätdata (eng. metrics) som består av ett namn och ett numeriskt värde. Användaren kan övervaka egna resurser och logga mätdata via olika API:er som AWS tillhandahåller.

För varje resurs som övervakas kan ett larm initieras. Larmet aktiveras när ett mätvärde överskrider en definierad tröskel. Vilka åtgärder som tas när ett larm aktiveras får användaren bestämma. Åtgärder kan också tas när mätdata för en resurs går tillbaka till ett önskat tillstånd. En tröskel kan exempelvis vara om CPU-användning för en virtuell server överskrider 75% av genomsnittlig användning under en period av 5 minuter. En vanlig åtgärd är att skicka ett meddelande via Simple Notification Service [13].

3.12.3 AWS Lambda

En AWS Lambda-service är en tjänst som exekverar kod som svar på någon händelse. Olika händelser som kan utlösa en AWS Lambda-service kan vara meddelanden från en SNS, anrop från en webbapplikation och ändringar i en databas. En AWS Lambda-service har stöd för Java, NodeJS och Python [14].

3.12.4 DynamoDB

DynamoDB är en databastjänst av typen NoSQL på AWS. Tjänsten har stöd för flera programmeringsspråk, exempelvis Java, Node.js, Python, Ruby och PHP [15].

4 Kravspecifikation

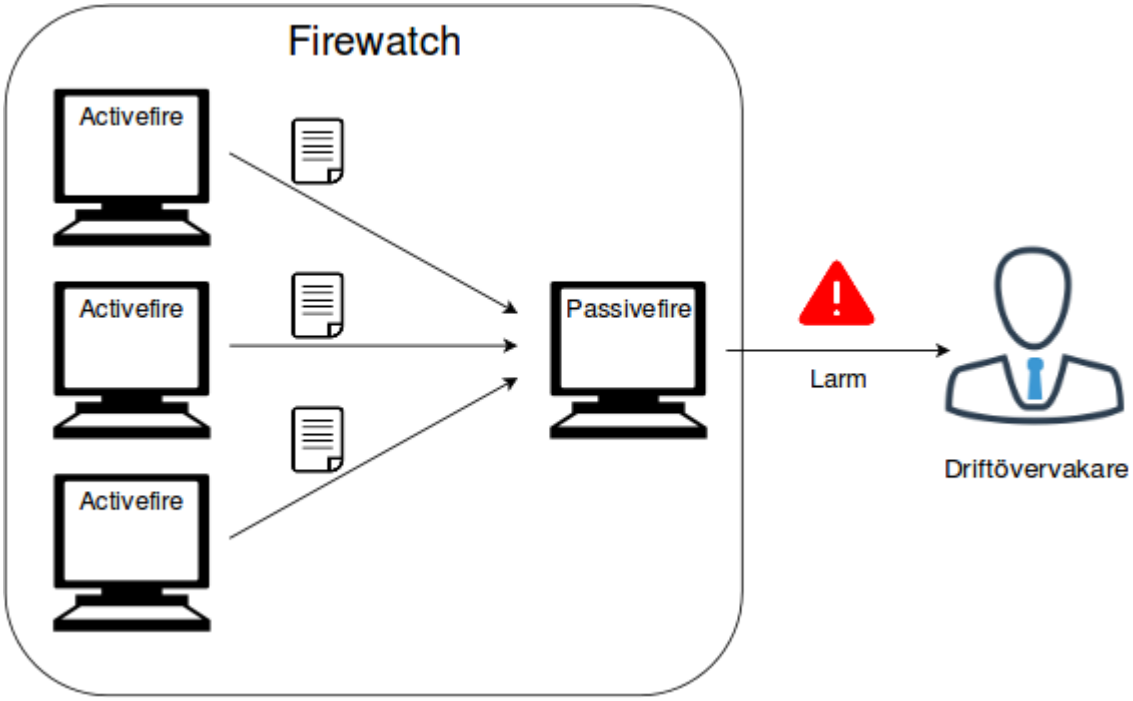
Webbapplikationen kommer att implementeras utifrån ett antal tekniska krav från e-man vad gäller val av programmeringsspråk och infrastruktur. Som infrastruktur för sina projekt använder e-man moln-tjänster från Amazon Web Services. Företaget vill av den anledningen att webbapplikationen också använder AWS som plattform.

I dagsläget är e-mans webbapplikationer utvecklade med AngularJS som front-end och AWS Lambda som back-end. Ett krav är därför att projektet utvecklas med samma tekniker.

5 Analys av nuvarande system

Det system som i dag övervakar e-mans systemintegrationer är utvecklat för att larma om någon integration råkar ut för driftstörningar.

Övervakningssystemet består av ett antal Apache Camel-processer som tillsammans bildar ett verktyg som går under namnet Firewatch. Firewatch är i sin tur uppdelad i två delar som heter Activefire och Passivefire som är skrivna med Java som programmeringsspråk. Figur 5.1 illustrerar övergripande hur Firewatch fungerar.



Figur 5.1: Nuvarande övervakningssystem.

5.1 Activefire

De kunddatorer som kör e-mans integrationer har en instans av Activefire installerad. Varje instans kommer kontinuerligt, med bestämd periodicitet, att sammanställa data om alla aktiva integrationer som körs på samma dator. Den sammanställda datan består, för varje integration, av:

- Id (namn på integrationen).
- State (tillstånd, exempelvis: startad/installerad/stoppad).
- Uptime (hur länge integrationen varit aktiv).
- Last exchange (tidpunkt för senast utförda arbete).

Datan skickas som en rapport till en instans av Passivefire.

5.2 Passivefire

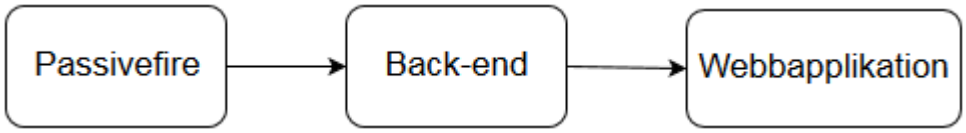
Varje kund har en instans av Passivefire. Passivefire har som uppgift att ta emot och analysera rapporter från instanser av Activefire. Passivefire söker igenom rapporterna efter drifstörningar. Om Passivefire upptäcker en avvikelse skickas ett larm i form av en incidentrapport. En incidentrapport innehåller information om vilken integration som har problem och vilken typ av problem som har inträffat. Incidentrapporten skickas till en anställd på e-man vars uppgift är att avgöra vilken åtgärd som är nödvändig.

Activefire meddelar Passivefire med vilken periodicitet den skickar rapporter med. Om Activefire inte rapporterat inom sin periodicitet kommer Passivefire att upptäcka detta och skickar ett larm.

6 Design

Under analysen uppdagades det att viss funktionalitet saknades i det nuvarande övervakningssystemet. Innan webbapplikationen utvecklades behövde funktionalitet implementeras som sparar undan information om integrationers driftstatus. Utan detta sparas ingen data som webbapplikationen kan visualisera.

Projektets övergripande design består av tre moduler, se figur 6.1, varav en redan existerar, nämligen Passivefire.



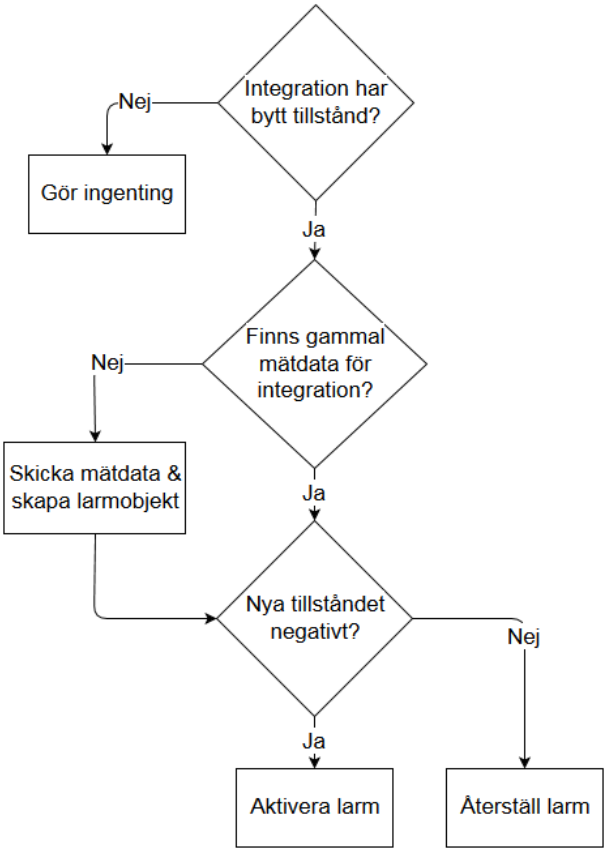
Figur 6.1: Övergripande design.

6.1 Modifiering av Passivefire

I e-mans nuvarande system är Passivefire konfigurerat för att skicka larm i realtid när en integration inte fungerar som önskat. Detta är inte tillräckligt om användaren över tid vill se hur en integration har varit i drift.

För att spara information om driftproblemen över tid utökades Passivefire med funktionalitet för att skicka avvikelser till Cloudwatch i form av mätdata. Dessutom skickar Passivefire mätdata till Cloudwatch när en integration återställs till ett godkänt tillstånd efter ett driftproblem.

Flödesdiagrammet i figur 6.2 illustrerar hur Passivefire kommunicerar med Cloudwatch när den bearbetar rapporter från Activefire.

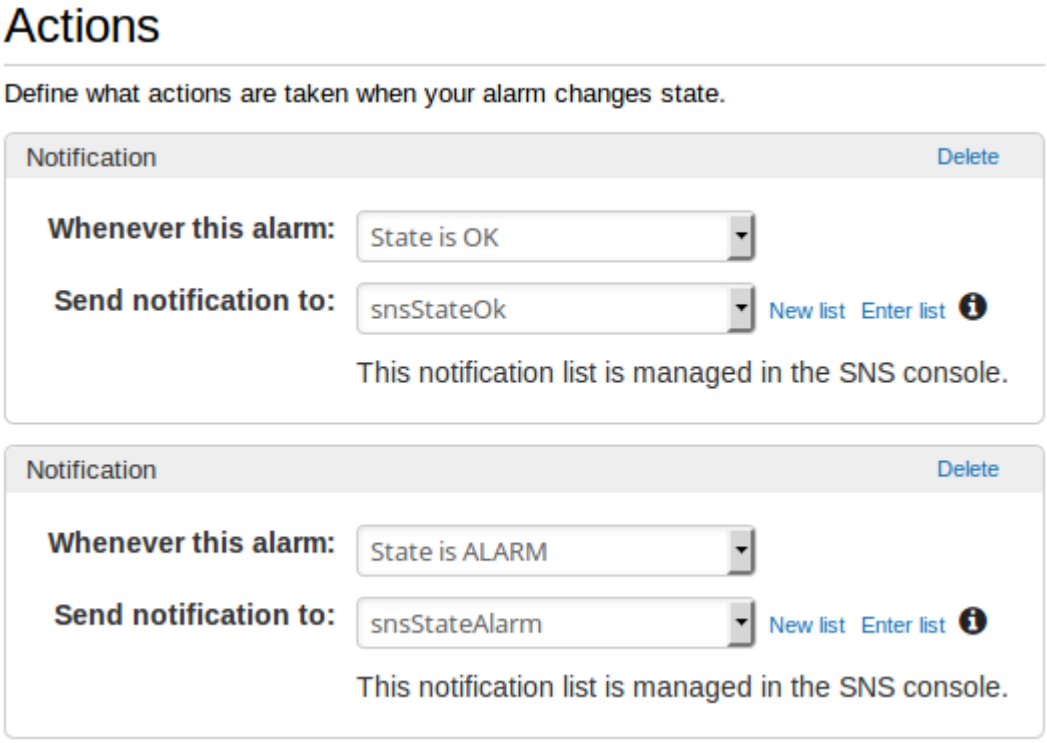


Figur 6.2: Flödesdiagram av Passivefires påbyggda funktionalitet.

Första gången mätdata skickas för en integration kommer ett larmobjekt att skapas. Dessa larm används till att larma när de upptäcker avvikelser i mätdatan. AWS har funktionalitet för att aktivera larm manuellt utan att utvärdera mätdatan. Det betyder att Passivefire kan ta en genväg och aktivera larmet direkt utan att skicka mätdata. Varje skickad mätdata kostar pengar och det blir därför billigare att manuellt aktivera larm vid driftändringar istället för att kontinuerligt skicka mätdata.

Larmen publicerar meddelanden till en SNS-kanal som representerar ett oönskat tillstånd. När ett larm återställs skickas ett OK till en annan SNS-kanal som representerar ett önskat tillstånd.

Utvecklaren kan med hjälp av ett grafiskt användargränssnitt (se figur 6.3) välja vilka åtgärder som ska tas när ett larm aktiveras och återställs.



Figur 6.3: Val av åtgärder för larm i AWS.

6.2 Back-end

När Passivefire aktiverar larm på AWS behöver det sparas undan för att larmets innehåll ska kunna visualiseras över tid. Det finns ett behov av en back-end som bearbetar larm, sparar information om larm till en databas och erbjuder en service som webbapplikationen senare kan anropa.

SNS-meddelanden, se exempel i figur 6.4, innehåller bland annat namn på den resurs (integration) som larmas, tidpunkt, namn på den kund som äger integrationen och ett tillstånd som antar värdet ALARM eller OK beroende på om ett larm har aktiverats eller återställts.

Figur 6.4: Exempel på SNS-meddelande.

En AWS Lambda-service skapades som prenumererar på dessa SNS-meddelanden. Lambda-servicen har i uppgift att filtrera fram data om integrationsnamn, kundnamn, larmtyp och tidpunkt. Lambdan-servicen sparar sedan datan i en DynamoDB-tabell. Figur 6.5 illustrerar hur driftinformation kan se ut i databasen.

Example.Stub0.TestCase0.ExampleOutAdapter	2016-04-05T12:33:10.245+0000	devCustomerName	ALARM
Example.Stub0.TestCase0.ExampleOutAdapter	2016-04-05T13:01:45.245+0000	devCustomerName	OK

Figur 6.5: Information om två driftändringar i databasen.

En tabell skapades i databasen som innehåller information om larm. Värdet ALARM i figur 6.5 innebär att en integration antagit ett ogiltigt tillstånd och värdet OK innebär att en integration antagit ett giltigt tillstånd.

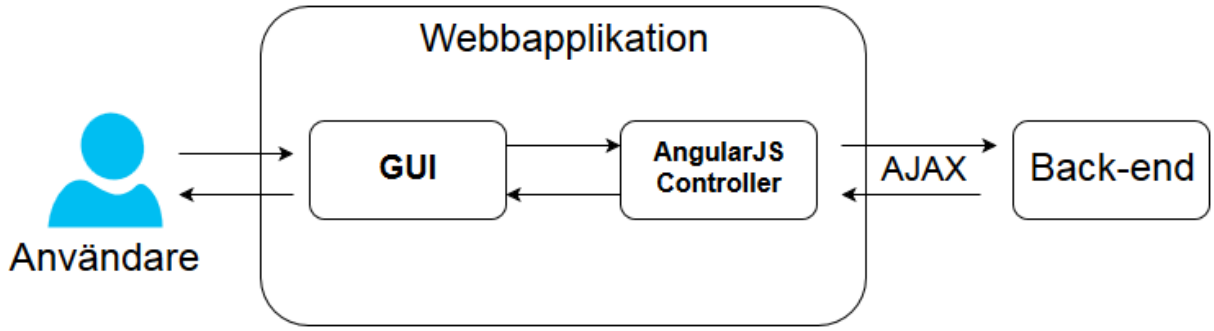
Två Lambda-servicar utvecklades med JavaScript. Lambda-servicarna körs när de får AJAX-förfrågningar från webbapplikationen.

Den första Lambda-servicen förser webbapplikationens första sida med data om alla kunder och deras integrationers drifttid över en månad. Funktionen anropas med en parameter som anger vilken månad som avses.

Den andra Lambda-servicen anropas med två parametrar, en för månad och en för integrationsnamn. Funktionen förser webbapplikationen med data avseende hur en specifik integration har presterat under en specifik månad.

6.3 Webbapplikation

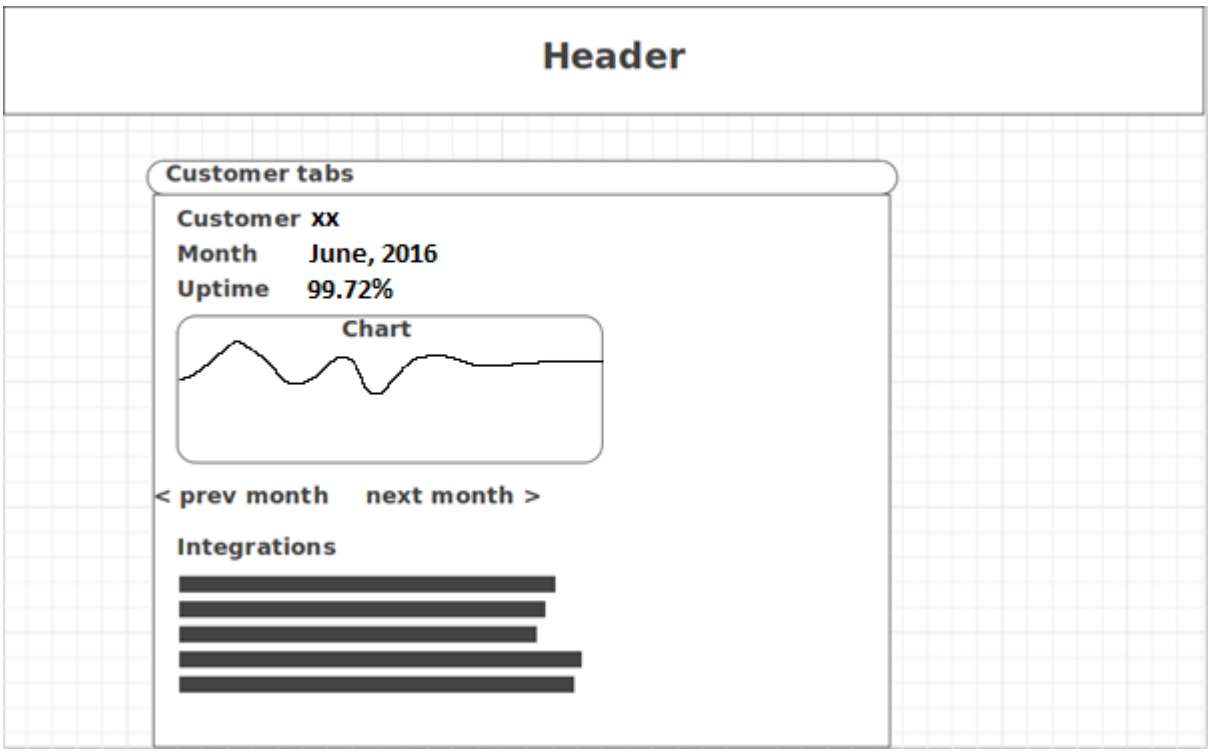
Figur 6.8 illustrerar hur webbapplikationens komponenter kommunicerar med användaren och back-enden.



Figur 6.6: Övergripande kommunikation i webbapplikationen.

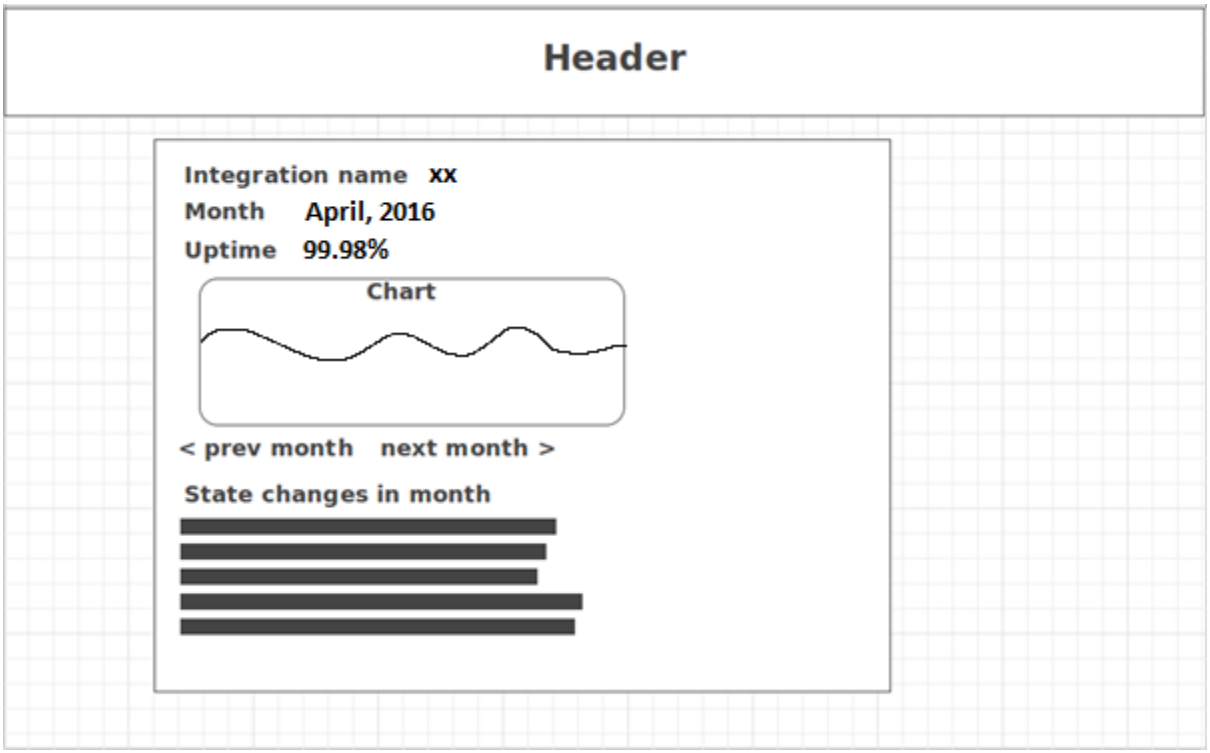
6.3.1 GUI

Tillsammans med e-man har det diskuterats hur tillståndsändringar och drifttid ska visualiseras. En övergripande bild av gränssnittet har ritats upp där fokus ligger på att kundvis och månadsvis visa genomsnittlig drifttid för varje kunds integrationer. Användaren ska sedan ha möjlighet att se på dagsnivå hur en integration har varit i drift. Se figur 6.7 för skiss av webbapplikationens första sida.



Figur 6.7: Skiss av webbapplikationens första sida.

Första sidan ger användaren en överblick över aktuell månad med flikar för varje kund (Customer tabs). För varje kund ritas en graf upp (Chart) över dennes integrationers sammanlagda drifttid under aktuell månad. Nedanför grafen listas integrationerna tillsammans med sin genomsnittliga drifttid under månaden. Användaren kan trycka på någon av kundens integrationer och navigeras då till den andra vyn, se figur 6.8.



Figur 6.8: Skiss av webbapplikationens andra sida.

I den andra vyn ritas en graf över den specifika integrationens drifttid under aktuell månad. Nedanför listas integrationens tillståndsändringar (State changes) under månaden.

Båda vyerna har knappar för att byta månad. Om månaden inte har inträffat kommer knappen för att navigera dit att vara avstängd.

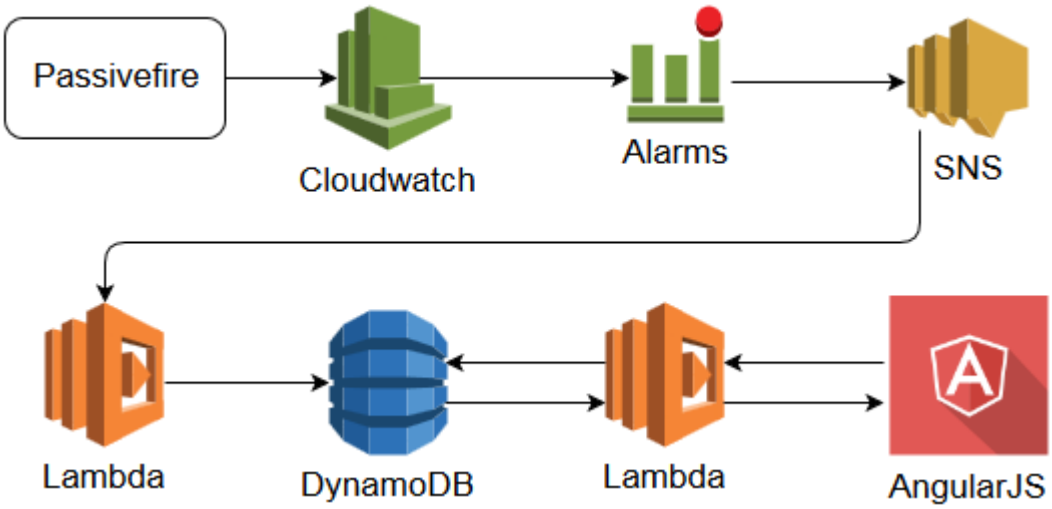
6.3.2 Controller

Webbapplikationens AngularJS-controller hanterar AJAX-anrop till back-endens två Lambda-servicar för respektive vy. När första sidan anropas kommer ett AJAX-anrop att göras med en parameter som representerar den aktuella dagens månad och år. Svaret från back-enden representerar data för alla kunders integrationer under den aktuella månaden. Datan skickas därefter vidare till användargränssnittet där den ritas upp.

Om användaren väljer en specifik integration görs ett nytt AJAX-anrop med en parameter som anger samma månad och år som på föregående vy, samt en parameter som anger namnet på den valda integrationen. Svaret från back-enden representerar data för den valda integrationen under den aktuella månaden.

6.4 Dataflöde

Figur 6.9 summerar hur data flödar genom designens komponenter.



Figur 6.9: Summering av designens dataflöde.

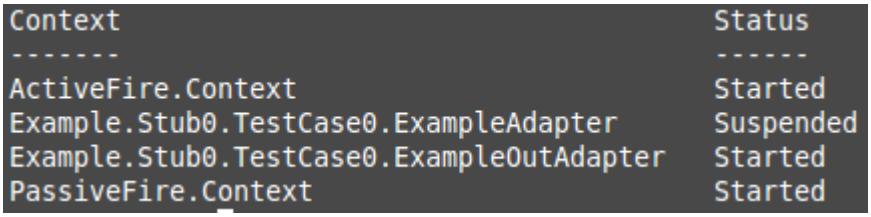
7 Implementation

7.1 Utvecklingsmiljö

En testmiljö sattes upp lokalt på en dator under de första veckorna av arbetet. Testmiljön består av Activefire, Passivefire och två exempelintegrationer:

- Example.Stub0.TestCase0.ExampleAdapter
- Example.Stub0.TestCase0.ExampleOutAdapter

Dessa integrationer startades och stoppades manuellt under projektetets implementation. Detta utfördes för att testa Passivefires funktionalitet då den upptäcker driftproblem. I figur 7.1 har en integration driftstatus “Suspended”. Detta är ett tillstånd tillsammans med “Stopped” som betraktas som driftproblem.



Figur 7.1: Utvecklingsmiljö.

7.2 Iteration 1: Passivefire

Passivefire är skriven i Java och hade redan logik för att upptäcka om en integration har fått ett driftproblem. Nya metoder skapades i Passivefire för att skicka mätdata, skapa larmobjekt och aktivera larm.

I figur 7.2 visas exempelkod för hur ett larm skapas för en integration.

```
private void createAlarm(String namespace, String metricName, String alarmName){
    List<String> alarmARNs = new ArrayList<>();
    alarmARNs.add("arn:aws:sns:eu-west-1:941582724539:alarmState");
    List<String> okARNs = new ArrayList<>();
    okARNs.add("arn:aws:sns:eu-west-1:941582724539:okState");
    PutMetricAlarmRequest request = new PutMetricAlarmRequest();
    request.setActionsEnabled(true);
    request.setNamespace(namespace);
    request.setMetricName(metricName);
    request.setAlarmName(alarmName);
    request.setComparisonOperator("LessThanThreshold");
    request.setThreshold(1.0);
    request.setStatistic("Average");
    request.setPeriod(60);
    request.setEvaluationPeriods(1);
    request.setAlarmActions(alarmARNs);
    request.setOKActions(okARNs);
    request.setAlarmDescription("customer:" + customerName);
    try {
        awsClient.putMetricAlarm(request);
        LOG.debug("PUT ALARM SUCCESS!");
    }
    catch(AmazonClientException e) {
        LOG.debug("COULD NOT PUT ALARM: " + e);
    }
}
```

Figur 7.2: Metod för att skapa larm på en integration.

7.3 Iteration 2: Back-end

Lambda-servicen som sparar data till en DynamoDB är skriven med JavaScript. Input till Lambda-servicen är en variabel som heter event. Variabeln innehåller SNS-meddelandet som i sin tur håller information om ett larm.

Figur 7.3 visar ett kodexempel hur en tillståndsändring lagras till databasen.

```
const message = event.Records[0].Sns.Message;
var messageJson = JSON.parse(message);
var metricName = messageJson.Trigger.MetricName;
var newStateValue = messageJson.NewStateValue;
var stateChangeTime = messageJson.StateChangeTime;
var customer = messageJson.AlarmDescription.split("customer:")[1];
var itemParams = {'Item': {'integrationId': {'S': metricName},
                           'changeTime': {'S': stateChangeTime},
                           'state': {'S': newStateValue},
                           'customerName': {'S': customer}}
                  };
try{
    ddb.putItem(itemParams, function() {
        context.done(null,'');
    });
```

Figur 7.3: Tillståndsändring lagras i en databas.

Lambda-servicarna, som webbapplikationen anropar, är också skrivna med JavaScript. De fungerar som en service för webbapplikationen. Servicens uppgift är att läsa från databasen och bearbeta datan till strukturerad information som är lätt för webbapplikationen att visualisera.

Figur 7.4 visar ett kodexempel hur en Lambda-service frågar databasen om tillståndsändringar för en specifik integration under en specfik månad.

```
var params = {
  TableName: table,
  KeyConditionExpression: "integrationId = :id AND changeTime < :endOfMonth",
  ExpressionAttributeValues: {
      ":id":id,
      ":endOfMonth":yearMonth.concat('-32')
  },
};

docClient.query(params, function(err, data) {
```

Figur 7.4: Databasförfrågan.

Datan som lagras i databasen representerar tillståndsändringar i integrationers drift. Detta utgör ett problem då driftinformation för varje dag i en månad ska visualiseras.

Problemet löstes genom att hitta senaste tillståndsändringen innan specifik månad och sätta det som starttillstånd för månaden. Det vill säga klockan 00:00:00 första dagen i månaden. Samtidigt sätts tillståndet för sista dagen i månaden klockan 23:59:59 (och 999 ms) till den sista tillståndsändringen för vald månad.

När månaden tilldelats ett start- och sluttillstånd enligt ovan, är det enkelt att för varje dag räkna hur en integration varit i drift. De dagar som inte har en tillståndsändring kommer tilldelas 100% eller 0% drifttid, beroende på den föregående tillståndsändringen. Det är bara de dagar som har tillståndsändringar som kräver matematiska beräkningar för att bestämma en drifttid.

7.4 Iteration 3: Webbapplikation

Webbapplikationen bygger på ett färdigt kodskelett kallat Angular-seed. Kodskelettet gör det enklare och snabbare att komma igång eftersom att utvecklaren får en färdig projektstruktur i AngularJS.

Webbapplikationen är relativt okomplicerad och hanterar framför allt ett antal beräkningar där genomsnittlig drifttid fastställs för integrationer och kunder. Den data som webbapplikationen arbetar med är redan bearbetad i back-enden för att passa de olika vyerna.

Figur 7.5 visar ett kodexempel på ett AJAX-anrop från webbapplikationen till back-enden. En parameter skickas med som representerar en specifik månad.

```
$http
  .get('https://l1biqgajg4.execute-api.eu-west-1.amazonaws.com/dev/getByMonth/'
    + window.encodeURIComponent(month))
  .then(function successCallback(response) {
```

Figur 7.5: AJAX-anrop till en lambda-service.

8 Resultat

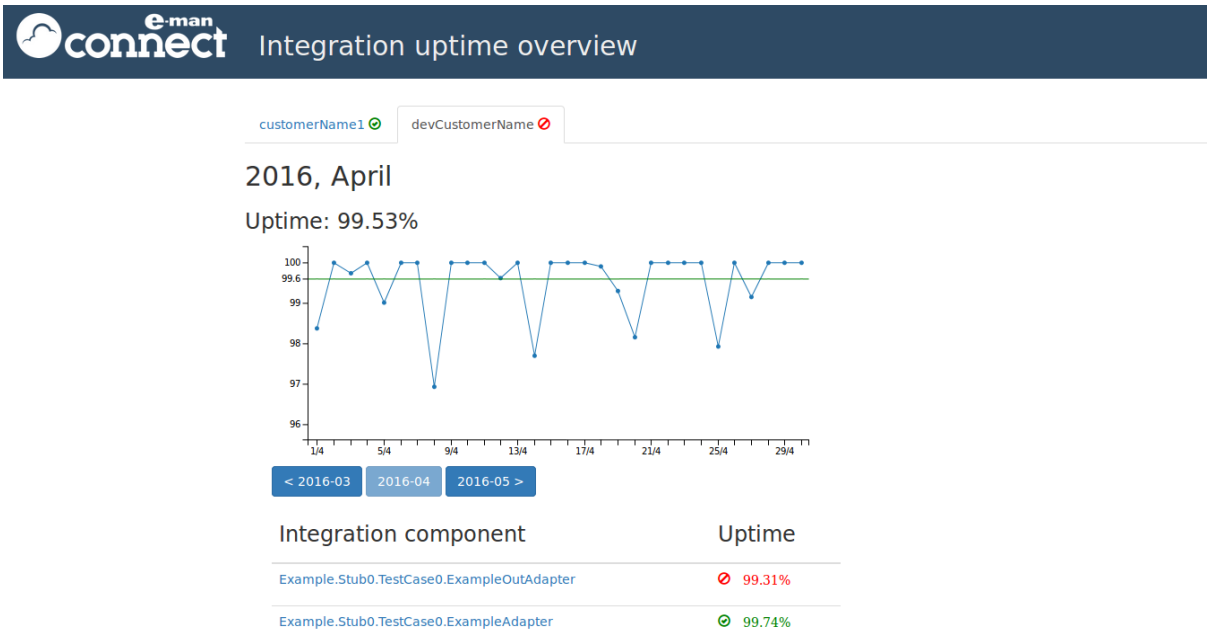
Målet för projektet var att ta fram en webbapplikation som analyserar och visualiserar den driftinformation som över tid skickas från e-mans integrationer. Resultatet blev en webbapplikation med två vyer.

8.1 Kundbaserad månadsvy

Webbapplikationens kundbaserade månadsvy, se figur 8.1, har en flik för varje kund. I vyn finns det två kunder med påhittade namn. För den markerade kunden (devCustomerName) kan användaren se en total drifttid (Uptime) för kundens integrationer. Om en kunds totala drifttid är lägre än 99.6% är serviceavtalet brutet och en röd symbol kommer att visas vid kundens flik.

Centralt i vyn är en graf över den genomsnittliga drifttiden för kundens integrationer under månaden. Varje punkt i grafen representerar en dag i den valda månaden. En grön horisontell linje är utritad vid 99.6%. Användaren kan hålla muspekaren över de olika datapunkterna i grafen för att se den exakta procentsatsen för den markerade dagen.

Längst ner i vyn listas kundens integrationer tillsammans med sina genomsnittliga drifttider. Integrationerna är klickbara vilket tar användaren till nästa vy.

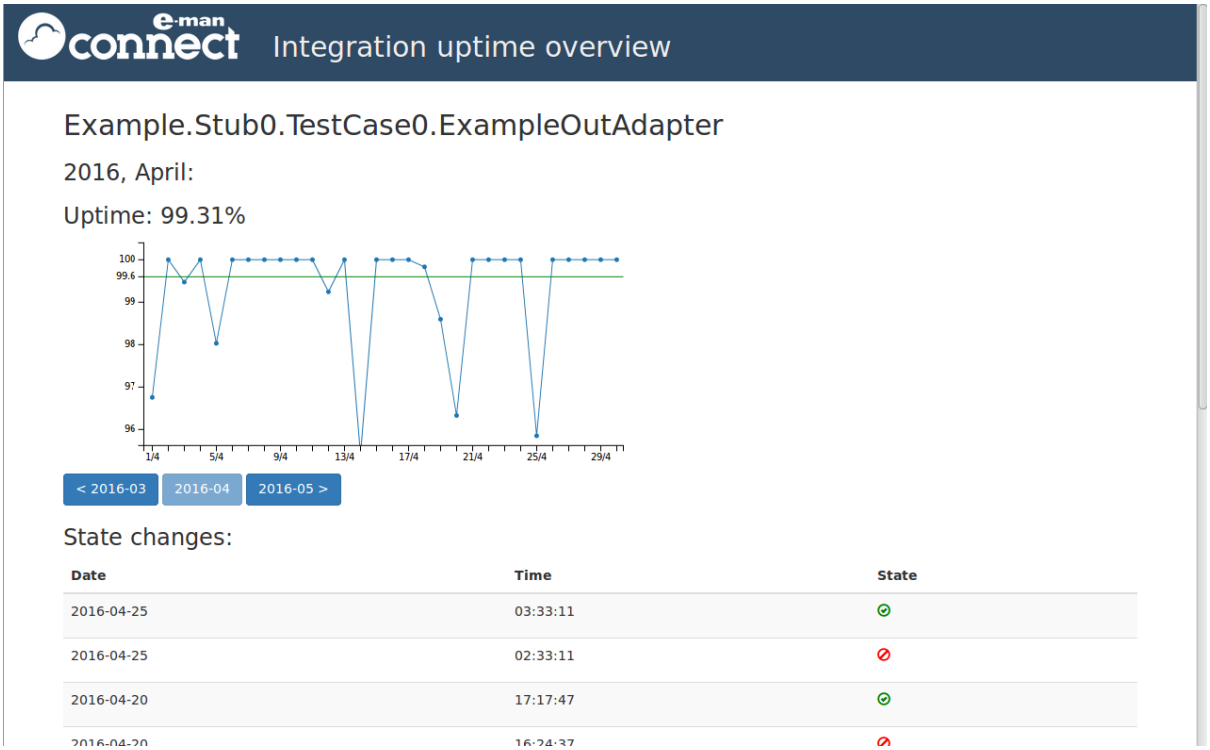


Figur 8.1: Webbapplikationens kundbaserade månadsvy.

8.2 Integrationsspecifik månadsvy

Den integrationsspecifika månadsvyn, se figur 8.2, visar driftinformation för en integration. En graf ritas upp på samma sätt som i den kundbaserade månadsvyn. Skillnaden är att denna grafen bara visar drifttid för den valda integrationen.

Längst ner i vyn listas alla tillståndsändringar för integrationen under vald månad. Tillståndsändringarna är sorterade med den senaste tillståndsändringen högst upp.



Figur 8.2: Webbapplikationens integrationsspecifika månadsvy.

9 Slutsats

En webbapplikation som visar tydligt om e-man uppfyller sitt serviceavtal mot de kunder som integrationerna tillhör har tagist fram. Webapplikationen kommer, utifrån den bedömning e-man gjort, att betydligt förenkla e-mans dialog med kunder när e-man undersöker om serviceavtal är uppfyllda för integrationer. Därmed anses att projektets syfte har uppnåtts.

10 Diskussion

En tidig frågeställning var vilken komponent i Firewatch som skulle skicka data till Cloudwatch. En potentiell lösning är att låta Activefire skicka data kontinuerligt till Cloudwatch. Beslutet att istället skicka data från Passivefire togs för att Passivefire redan har funktionalitet för att upptäcka när en integration får driftproblem. Dessutom skulle betydligt mer data skickas till Cloudwatch när den skickas kontinuerligt. Detta blir dyrare då e-man betalar för varje datamängd som skickas.

När projektet är färdigt har en alternativ lösning på back-enden observerats. Den alternativa lösningen skulle låta Passivefire skicka data direkt till databasen. Den självklara fördelen med denna lösning är att ett antal mellansteg elimineras mellan e-mans nuvarande system och webbapplikationen. En fördel med den implementerade lösningen är en lösare koppling mellan Passivefire och databasen vilket innebär att framtida funktionalitet enklare kan implementeras.

Lösningen kommer att sättas i drift och den nya versionen av Passivefire kommer att distribueras till kundernas integrationsmiljöer. Förslag på utökad funktionalitet har getts från anställda på e-man. Bland annat finns ett intresse av funktionalitet där användaren kan skriva kommentarer som kopplas till integrationers driftproblem.

En positiv miljöaspekt som arbetet bidrar med är att e-mans kunder kan se hur deras integrationer har körts och därmed behöver de inte föra samma dialog med e-man som tidigare. Detta minskar antalet möten och därmed antalet resor mellan kund och e-man.

11 Referenser

[1] java.com, ‘What is Java and why do I need it?’.
[Online]. Tillgänglig: https://java.com/en/download/faq/whatis_java.xml [Använd 14 April 2016].

[2] w3.org, ‘Your first look at JavaScript’.
[Online]. Tillgänglig: https://www.w3.org/wiki/Your_first_look_at_JavaScript [Använd 14 April 2016].

[3] wikipedia.org ‘Ajax (programming)’.
[Online]. Tillgänglig: https://en.wikipedia.org/wiki/Ajax_(programming) [Använd 7 Juni 2016].

[4] developer.chrome.com, ‘MVC Architecture’.
[Online]. Tillgänglig: https://developer.chrome.com/apps/app_frameworks [Använd 15 April 2016].

[5] whatis.techtarget.com, ‘What is framework?’.
[Online]. Tillgänglig: http://whatis.techtarget.com/definition/framework [Använd 16 April 2016].

[6] wikipedia.org ‘Open-source software’.
[Online]. Tillgänglig: https://en.wikipedia.org/wiki/Open-source_software [Använd 18 April 2016].

[7] Officiell introduktion till AngularJS.
[Online]. Tillgänglig: https://docs.angularjs.org/guide/introduction [Använd 12 April 2016].

[8] camel.apache.org, ‘Apache Camel: Enterprise Integration Patterns’
[Online]. Tillgänglig: http://camel.apache.org/enterprise-integration-patterns.html [Använd 25 April 2016].

[9] camel.apache.org, ‘Apache Camel:What is Camel’
[Online]. Tillgänglig: http://camel.apache.org/what-is-camel.html [Använd 25 April 2016].

[10] aws.amazon.com, ‘What is NoSQL?’
[Online]. Tillgänglig: https://aws.amazon.com/nosql/ [Använd 29 April 2016].

[11] aws.amazon.com, ‘What is AWS?’.
[Online]. Tillgänglig: https://aws.amazon.com/what-is-aws/ [Använd 13 April 2016].

[12] aws.amazon.com, ‘Amazon SNS’
[Online]. Tillgänglig: https://aws.amazon.com/sns/ [Använd 29 April 2016].

[13] aws.amazon.com, ‘Amazon Cloudwatch’
[Online]. Tillgänglig: https://aws.amazon.com/cloudwatch/ [Använd 29 April 2016].

[14] docs.aws.amazon.com, ‘What is AWS Lambda?’
[Online]. Tillgänglig: http://docs.aws.amazon.com/lambda/latest/dg/welcome.html [Använd 29 April 2016].

[15] aws.amazon.com, ‘Amazon DynamoDB Documentation’
[Online]. Tillgänglig: https://aws.amazon.com/documentation/dynamodb/ [Använd 29 April 2016].