



Slutrapport

Naviga filuppladdning-applikation

Uppdragsgivare:

Naviga

Projektgrupp:

Sebastian Källstedt

Mosa Kasem Rasol

Emil Larsson

Alex Hjortenkrans

Yasmin Mushahdi

Innehållsförteckning

| | |
|---|-----------|
| Sammanfattning | 3 |
| Inledning och bakgrund | 4 |
| Syfte och mål | 5 |
| Projektorganisation | 6 |
| Roller och Ansvarsfördelning | 6 |
| Genomförande: metod och teknik | 7 |
| Bitbucket | 7 |
| Produkt backlog | 8 |
| Kommunikation | 9 |
| Tekniker | 9 |
| Backend | 10 |
| Frontend | 11 |
| Resultatbeskrivning | 12 |
| Frontend | 13 |
| Backend | 15 |
| Diagram över systemet | 15 |
| Visualisering av state machine | 17 |
| Avvikelse | 19 |
| Slutsats | 20 |
| Övertagande organisation | 24 |
| Förslag till förbättringar inför kommande projekt | 24 |
| Förslag på vidareutveckling | 25 |
| Litteraturförteckning & dokumentationshänvisning | 26 |
| Projekthänvisningar | 26 |
| Dokumentationshänvisningar för tekniker och verktyg | 26 |

Sammanfattning

Naviga arbetar med att tillhandahålla lösningar och applikationer till mediehus och tidningar och arbetar mycket med att ligga i framkant för nya tekniker och innovationer. Naviga har ett verktyg/applikation som heter Writer som är ett artikel skriververktyg som journalister kan skriva artiklar i och sedan ladda upp till sin egen tidnings applikation eller hemsida. Idag har Naviga ett komplicerat system för att ladda upp bilder för artiklarna via detta verktyg och vårt uppdrag var att utvärdera en lösning på detta som Naviga är intresserad av att använda. Denna lösning bygger på att skapa ett serverless system med AWS som använder en state machine. Rapporten som följer här beskriver vårt projekt, vad vi har gjort, hur det har gått och vad vi själva tycker om både projektet och produkten.

Inledning och bakgrund

Naviga har idag ett system som hanterar uppladdning av bilder, validering check för duplicerade och skapa en thumbnail. System som används idag arbetare med flera onödiga processor och körningar i bakgrunden. Det har också blivit svårt att hantera och har blivit lite av en "ball of mud". Deras system gav inte användaren någon feedback om hur det gick för bilduppladdningen

Naviga ville att vår grupp skulle göra ett proof-of-concept på ett uppladdnings system till bilder som skulle använda sig utav AWS tjänster. Det skulle också ge användaren feedback hur långt bilden hade kommit i systemet. Vi skulle helt enkelt utvärdera AWS som system och arkitektur och användandet av en state machine för att kunna ge dom en fingervisning om AWS skulle kunna användas för att göra om deras egna system med.

Syfte och mål

Syftet med projektet var att ge Naviga ett proof-of-concept över hur ett system för bilduppladdning skulle kunna se ut i AWS. Naviga var främst intresserad av att få veta om våra upplevelser, eventuella bottlenecks och problem eller lösningar som vi har stött på. Dom ville helt enkelt ha ett findings dokument med saker som var bra att veta, positivt eller negativ under projektet gång mot det kraven som vi fick.

Projektet var riktat mot vår kund Naviga och det var endast dom som vi identifierade som målgrupp då själva applikationen inte var lika viktig som våra upplevelser. Vi fokuserade på att lösa vår kunds krav av applikationen och problemområdet som vår kund har nämnt.

Målet med projektet var att ha en applikation i AWS som skulle hantera bilduppladdning och göra följande när en bild laddas upp:

- Validera om filen som hade blivit uppladdad var en bild av typen JPG,PNG, GIF och HEIC/HEIF.
- Kontrollera om bilden är en duplicering av en bild som redan finns i en S3 Bucket och i så fall ta bort den.
- Hämta ut exif datan ifrån bilden
- Göra en thumbnail utav bilden

Målet med klienten under projektet gång var:

- Kunna ladda upp en bild
- Kunna se feedback om hur det går bilduppladdning processen

Projektorganisation

Projektet har genomförts med en blandning av två olika projektmetoder, Unified Process och SCRUM. Vi använde oss av Unified process för att det är ett iterativt arbetssätt som passade oss väl. Vi lade mycket fokus på att strukturera upp en plan för varje iteration vilket gav en generell bild av projektets faser. Detta i sin tur underlättade då när högst prioriterade områden skulle fördelas mellan roller i gruppen.

SCRUM arbetssätt implementeras för att genomföra en bra dokumentation över vårt projekts alla delar och för att kunna underlätta kommunikationen, samt det självständiga arbetet.

Roller och Ansvarsfördelning

Vi fördelade våra roller ganska snabbt i början av projektet och slutade på en projektledare, en kundansvarig, en testansvarig medan alla var lika ansvariga för implementationen samt den egna testningen av sin del under hela processen. Dokumentationen var också uppdelad till, antingen dem med tillhörande ansvarsområden, eller uppdelat ospecifikt och jämnt till varje gruppmedlem. Den generella fördelningen:

Kundansvarig/frontend: Sebastian

Projektledare: Emil

Testansvarig: Alex

Backend: Yasmin

Backend: Mosa

Genomförande: metod och teknik

Projektet har genomförts som en Unified Process som har varit uppdelat i 4 delar där varje del utöver vår egen utveckling har innehållit en feedback runda och en presentation. De fyra delar eller makro iterationer som genomförts är: Inception, Elaboration, Construction och Transition. Varje iteration har innehållit mål som ska uppnås, en redovisning av dessa mål i form av en presentation för de andra grupperna och handledare, och en genomgång av en annan grupps kod och dokumentation för iterationen för att ge feedback på förbättringar. Varje makro iteration delade vi även in i mikro iterationer en mikro iteration motsvarade en vecka. Varje ny iteration inleddes med ett handledarmöte där vi gick igenom föregående iteration för att sedan utifrån detta skrev en ny iterationsplan.

Projektets tidsresurs har varit 200 arbetstimmar per utvecklare vilket har gjort att 20 timmar har varit uppskattad tid att lägga varje mikro iteration per utvecklare.

Vi har under projektets gång skapat en produkt backlog, egna milstenar, issues på bitbucket och en trello-board för att försöka få en överblick över vem som gör vad och vad som behöver göras.

Bitbucket

Vi valde utifrån rekommendation från kund att använda oss av Bitbucket istället för Github för versionshantering och skrivandet av vår dokumentation. Bitbuckets pipeline var enligt kunden väldigt smidiga att använda och vi valde Bitbucket mycket ifrån detta då vi kände att detta kunde vara något att prova på under projektets gång. I slutändan använde vi dock inte pipelines då tiden inte räckte till och vi valde att prioritera andra saker. I Bitbucket finns hela projektets kodbas och all tillhörande dokumentation, både för hur man använder systemet eller kommer igång med en utvecklingsmiljö till vision och projektplan finns i tillhörande WIKI. Vår kodbas är versionshanterad och vi har hela tiden utgått från en development branch. Från development branchen har varje utvecklare skapat en ny branch när hen ska påbörja en ny feature eller testa en ny teknik. Då projektet har kretsat kring en helt ny teknik och en state machine har byggts i AWS vilket ingen av oss utvecklare hade någon erfarenhet i sedan innan så har det blivit många test branscher och kod som överskrider olika arbetsområden. Efter att en utvecklare känner sig nöjd med sin kod och vill lägga till den i

development branschen så skapade hen en pull request där minst en annan utvecklare sedan har gått igenom koden och antingen godkänt förfrågan eller avslagit den för att förbättringar krävs. Vi har varit konsekventa med att minst en annan utvecklare ska kolla igenom koden innan en pull request godkänns för att vara säkra på att bara kod av god kvalitet har hamnat i development branschen.

Projektet som vår kund Naviga ville att vi skulle utföra är som tidigare nämnt ett "proof-of-concept" där vi använder oss av AWS. Utifrån detta så fick vi en del krav, men dessa var väldigt lösa då de själva inte visste om eller hur lätt kraven skulle gå att lösa. Då projektet redan från början inte var så väldefinierat och ingen av oss utvecklare har arbetat med AWS tidigare så har det under projektets gång varit väldigt svårt att skapa konkreta issues på Bitbucket eller TODOS via en trello board. Vi har haft en produkt backlog där vi har skrivit in features och delar som vi vill att systemet ska ha och har försökt att följa detta och uppdatera när det har varit möjligt. I och med att AWS är så otroligt stort och allting går att göra på många olika sätt så har vår största utmaning varit i att hitta hur man faktiskt löser en uppgift. När man har undersökt en del som man jobbar med så har man hittat en del som man ser skulle kunna användas för en annan feature och vi märkte därför tidigt att det skulle bli väldigt svårt för oss att hålla oss till en strikt issue eller TODO. Vi har därför varit väldigt flexibla med vad som behöver göras och vi har under stora delar av projektet jobbat överskridande mellan olika delar, vilket istället har gjort att vår dokumentation av issues och saker som en utvecklare har jobbat med för stunden har blivit lite lidande. Detta har även synts i både iterationsplaner och tidrapporteringen som har varit svår, både att säga vad varje utvecklare specifikt kommer att jobba med men att sedan tidslogga detta då man ofta har förflyttat sig mellan olika delar i projektet. Arbetet med issue eller trello eller någon annan metod för att hela tiden hålla koll på vad varje utvecklare har arbetat med och vad som behövs göras är något vi hade försökt göra annorlunda, även om vi är mycket nöjda med produkten som vi fått fram.

Produkt backlog

Då projektet som tidigare nämnt har varit väldigt svårt att förutse vad som kommer att behöva göras och hur det faktiskt kommer göra har bestämts väldigt mycket efter vad för information om hur just den feature skulle kunna göras i AWS man har hittat. Vi har försökt att skapa egna milstenar i vår Wiki på Bitbucket vilket till största del har varit bra vid kontakt med kund för att se att vi prioriterar rätt.

För all annan dokumentation av vilka krav vi har kvar att arbeta med och vilken status det är på dessa så har vi använt oss av en produkt backlog.

Kommunikation

Under hela projektets gång har vi haft möten tillsammans med vår handledare på tisdagar klockan 10. Detta blev början på vår nya iteration vilket passade bra då dessa möten ofta behandlade saker från senaste iterationen och sedan vad vi skulle göra under veckan. Vi själva har haft kortare möten varje morgon de dagar vi jobbat tillsammans på campus, vilket har varit 2-3 dagar i veckan. Dessa möten har då behandlat vad vi själva gjort sedan vi senast sågs och vad vi ska gå vidare med. Vi har haft kontinuerlig kontakt med vår kund via Slack och vi har haft 3 möten uppe på deras kontor där vi har gått igenom var vi är, vad vi ska jobba vidare med och hur vi ska prioritera. I och med att kunden har ställt väldigt låga krav då de helst av allt vill veta om sättet vi skapat applikationen på skulle fungera för dem eller inte. Detta har gjort att vi själva har fått sätta upp vilka krav vi tycker att systemet ska ha och hur vi prioriterar dessa. Det viktigaste vi har fått ut av kundmöten är att se att vi inte faller utanför ramen för vad kunden vill ha med sitt eget system om de skulle implementera detta för att skapa ett så realistiskt system som möjligt.

I och med att vi har suttit och arbetat mycket tillsammans så har vi hela tiden diskuterat beslut och hur vi ska implementera krav. Detta är även det en anledning till att vårt arbete med issues eller TODOS har blivit lidande. Vår kommunikation har istället varit väldigt god och väldigt mycket av krav som implementerats och instruktioner för hur man gör detta i AWS har förmedlats mellan alla gruppmedlemmar så att alla alltid har vetat vad de andra håller på med, och hur man tekniskt gör något. När vi inte suttit tillsammans så har kommunikationen skett över Slack vilket också har fungerat utmärkt. Detta har gjort att gruppen hela tiden har koll på vad de andra medlemmarna gör och kan ge feedback på detta eller hjälpa till om något har behövts.

Tekniker

Vår kund Naviga var redan från början väldigt tydlig i att det proof-of-concept dom ville att vi skulle skapa skulle vara byggt i [AWS](#). Deras nuvarande system för att ladda upp bilder använder sig av två NodeJS servrar och en Java server och är både invecklat och optimerat. AWS är Amazon web services och innefattar otroligt många olika funktioner och applikationer och det går att skapa otroligt stora och komplicerade arkitekturer och system.

Styrkan med AWS är att eftersom allt ligger i molnet så när vi kör vår backend så kan systemet automatiskt skala beroende på antalet requests som görs samtidigt vilket reducerar väntetiden när flera använder tjänsten samtidigt.

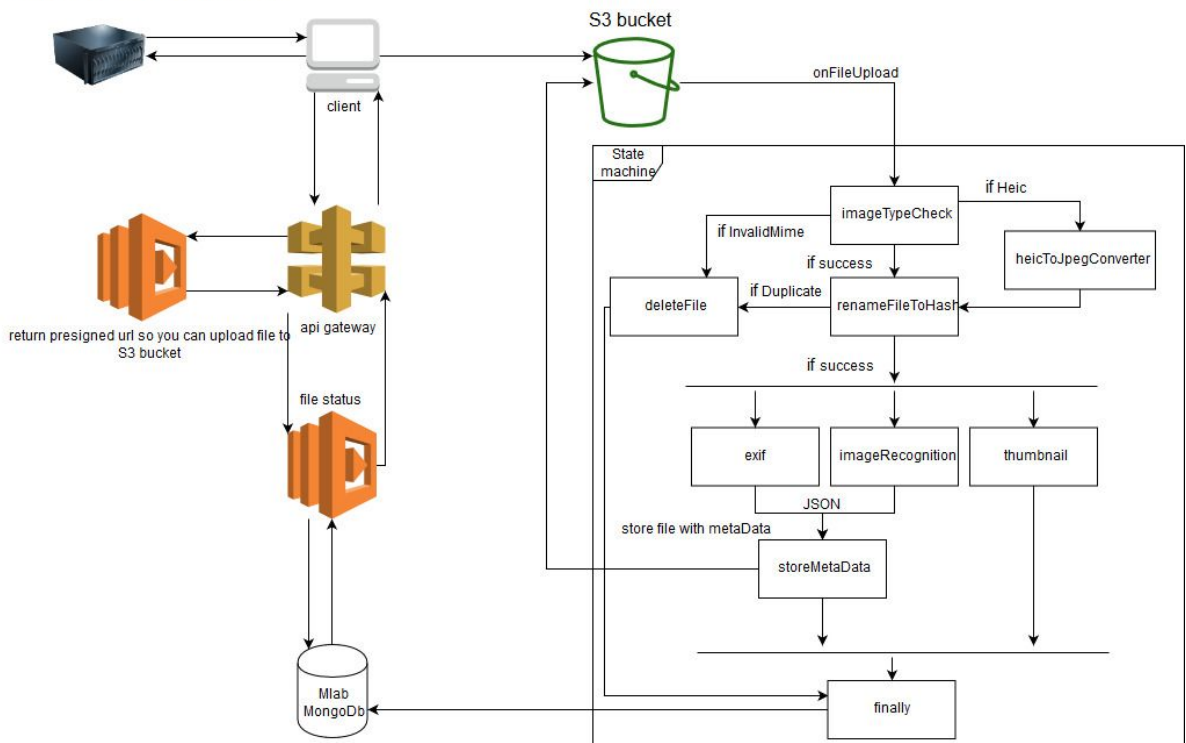
Backend

Vår backend är alltså byggd i AWS och exakt vad den innehåller och fungerar kan läsas i kapitlet **Resultatbeskrivning**. I AWS har vi skapat en state machine som innehåller lambda funktioner som fungerar som step functions som triggar varandra efter att föregående har exekverat klart. Vår arkitektur för hela backend systemet kan ses på nästa sida.

De services vi använder i AWS förutom våra egenskapade funktioner är S3 bucket vilket är ett lagringsutrymme för filer, API gateway för att sätta upp routes för att komma åt vår data, CloudTrail och CloudWatch för eventhantering och för att se till att vår state machine startar när en bild laddas upp från klienten till S3 bucket.

Utöver services från AWS så använder vi mongoDB från mLab som dokumentdatabas för att spara informationen om hur det har gått med hanteringen av bilden i state machine.

Companion (Google drive/Instagram/Upload from uri)



Mer om hur vår backend fungerar finns i resultatdelen.

Frontend

Vår frontend är byggd i javascript med hjälp av ramverket [ReactJS](#). Vi har använt oss av en modul som heter [Uppy](#) som är en modul för just filuppladdning. Uppy valde vi även det att använda efter rekommendation från kunden. **Uppy** löser på ett smidigt sätt uppladdning från webbläsare, via google drive och det ska även fungerar med URL, vilket dock vi fick ta bort i sista stund på grund av tillfälliga fel i modulen. Som tillägg till uppy så har vi skapat en NodeJS server som Uppy modulen behöver för tredje parts integration. Denna server ligger produktionssatt på **Heroku**.

Resultatbeskrivning

Vid projektets slut har en leverans gjorts och en presentation inför kunden kommer att ske veckan efter att denna rapport inlämnas. Leverans består i ett findings dokument med saker som vi har upptäckt under projektets gång och saker som vi tror kan vara av intresse vid kundens egen utveckling av ett liknande system. Leveransen har även bestått i att kunden fått tillgång till vår kod och alla tillhörande dokumentation om något skulle vara av intresse för dem i framtiden.

En presentation inför företaget kommer alltså att ske där vi ska presentera systemet som det ser ut, vi kommer att presentera för och nackdelar med systemet men framför allt med AWS och hur det är att bygga ett serverless system i AWS.

Vid leverans har 73 % av baskraven som kunden gav oss uppfyllts. Anledningen till att inte alla baskrav har uppfyllts är att AWS tog mycket tid att ens komma igång med då upplägget och hela systemet är väldigt komplext och dokumentationen inte alltid håller den standard man skulle vilja. Tröskeln är väldigt hög för att kunna arbeta effektivt i AWS, dock anser vi att vi nu när projektet slutförts med mycket större säkerhet kunnat utveckla i AWS.

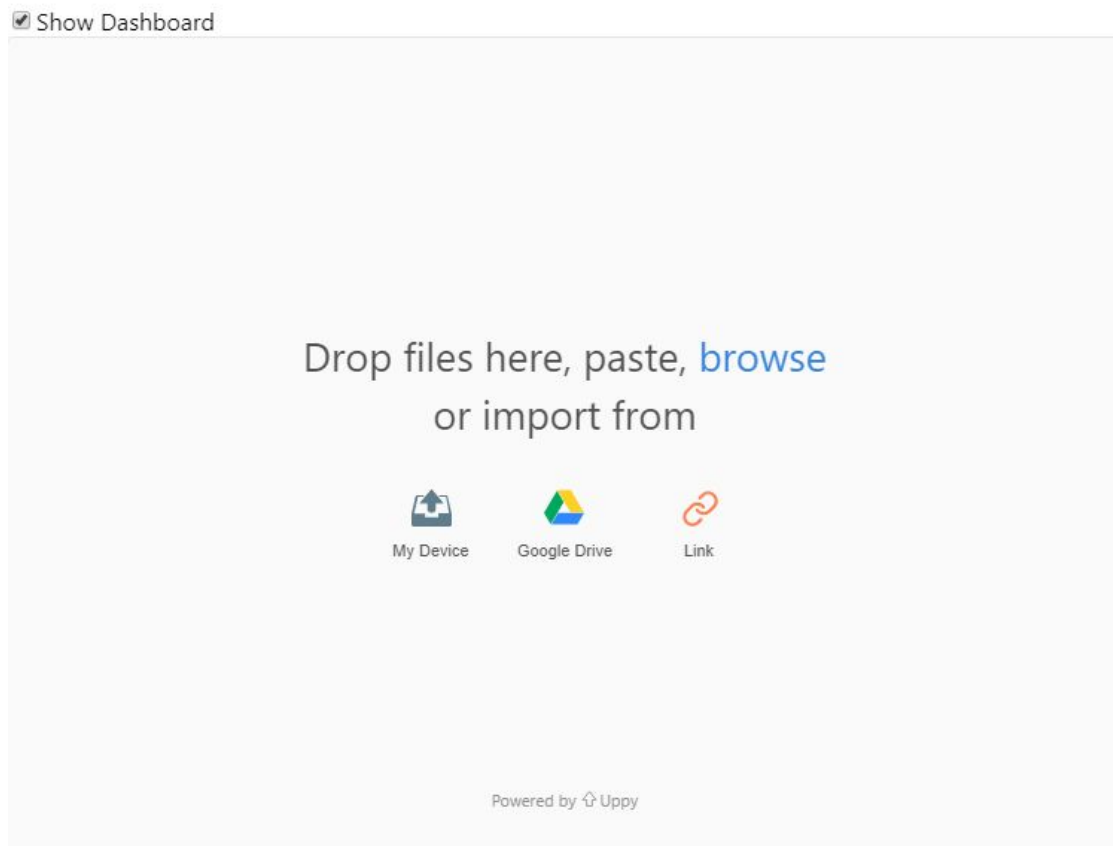
Kunden har dock varit tydlig i att det är konceptet som är viktigt och att inte alla baskrav behöver vara med utan att det är våra egna tankar och saker vi stött på som är viktiga.

Gruppen har även implementerat extra funktionalitet som var önskat krav som framkommit från kunden efter senare iteration.

Frontend

Writer upload

Inline Dashboard

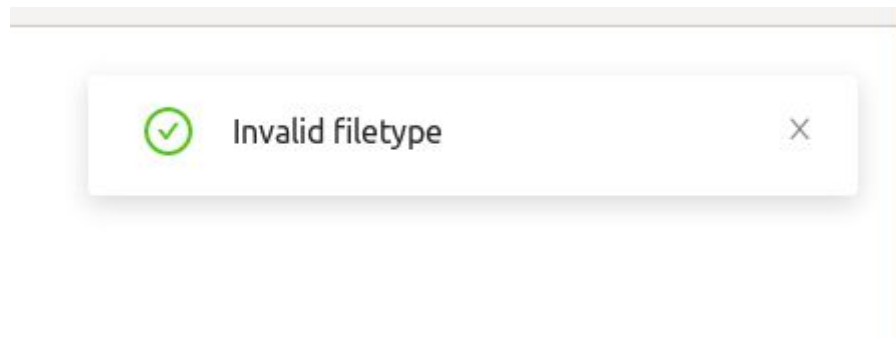


Progress Bar

Ovan visas en bild på front-end. Detta är den vyn som visas vid första uppstart av applikationen, där möjlighet finns för att ladda upp bilder ifrån datorn och google drive.

Frontend är fristående react-applikation, och den fungerar som en SPA. Utseende var inte viktigt utan det var fungerande funktionalitetskrav som kunden var mest intresserade av. Som nämnt i metodkapitlet så användes modulen Uppy som är en modul för just bilduppladdning som gjorde att vi kunde lägga mindre tid på klienten och fokusera på vår huvuddel som var backend i AWS.

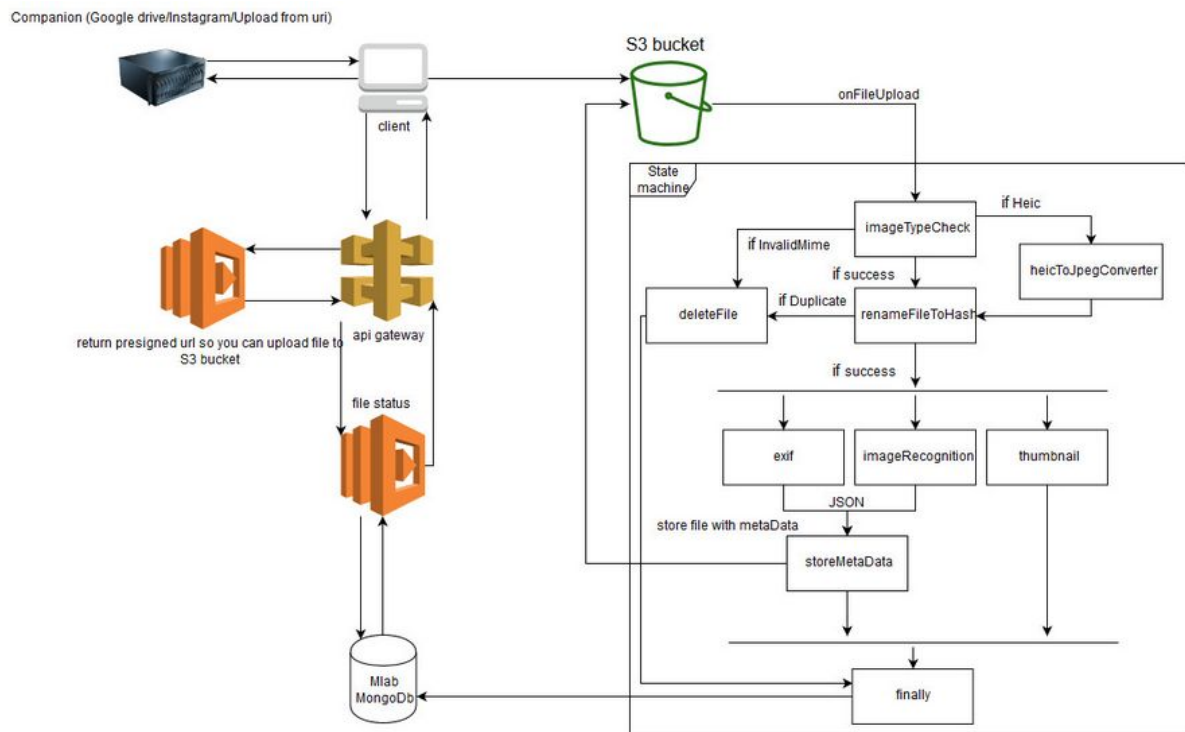
När en bild är uppladdad och har gått igenom vår state machine eller när något har gått fel så skickar vi tillbaka ett meddelande till klienten som ser ut som bilden nedan. Denna bild visar på meddelandet om en fil som inte är av en giltig bildtyp laddas upp.



Tillhörande klienten så byggde vi även en NodeJS server som modulen Uppy behövde för att kunna få tredjeparts integration att fungera korrekt. Denna server byggdes utefter dokumentation från Uppy. Detta visade sig dock vara mycket svårare än vi trodde då vi till en början ville att denna server även den skulle vara hostad hos AWS. Vi försökte både att hosta den direkt i en S3 bucket vilket ska fungera, men inte gjorde så, och sedan försökte vi hosta den med hjälp av en AWS service som heter [Elastic beanstalk](#). Detta fick vi dock inte heller att fungera då vi inte lyckades lista ut hur man skulle sätta upp API gateway för att göra re-directs från vår server tillbaka till klienten. I slutändan hostade vi server på [Heroku](#) vilket fungerade mycket bättre.

Backend

Diagram över systemet



Ovan visas en övergripande bild på arkitekturen för den applikation som har utvecklats.

Steg ett är att skicka en HTTP POST till en lambda funktion genom en gateway som returnerar en URL som klienten sedan kan ladda upp till, bilden hamnar då i en Bucket vilket fungerar ungefär som en katalog. När bilden har laddats upp helt så triggas en state machine som innehåller ett flöde där bilden behandlas.

I första funktionen `imageTypeCheck` så läser funktionen in de 24 första byten från bilden och kontrollerar mime typen om det är en godkänd bild. Är den inte det så går flödet till `deleteFile` och senare `finally` för att visa ett meddelande till klienten. Skulle bilden vara en HEIC så tar bilden vägen genom den och omvandlas till en vanlig jpg.

Nästa funktion är `renameFileToHash`, funktionen läser in hela binären i minnet och skapar en hashad sträng. Efter det kontrollerar den om det finns en fil redan i bucket med det

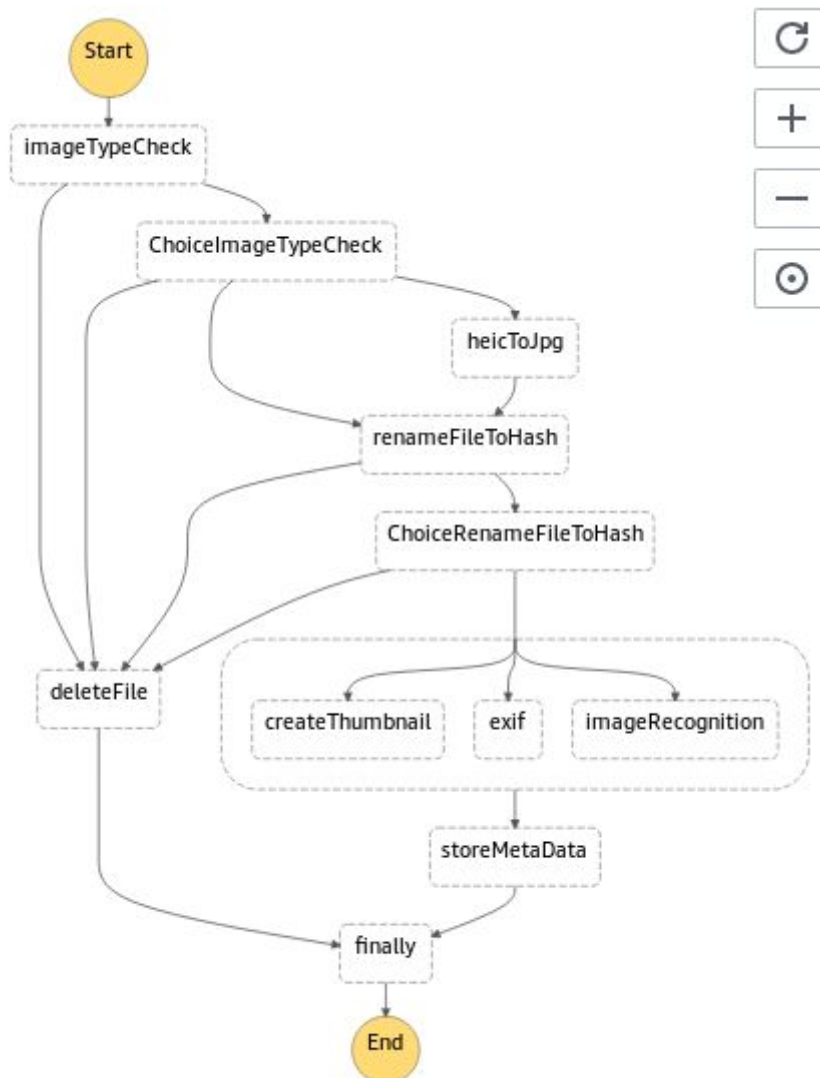
filnamnet om så är fallet räknas det som en dubblett och filen tas bort och klienten får ett meddelande om detta. Annars döps filen om och flyttas ned till rotkatalogen.

Efter detta sker tre parallella steg, exif data plockas ut från bilden, bildigenkänning data och även skapas det mindre bilder av den stora bilden så kallade thumbnails. exif och bild igenkänningen sammanstrålar och det läggs ett json objekt med samma namn som bild filen fast med .json på slutet.

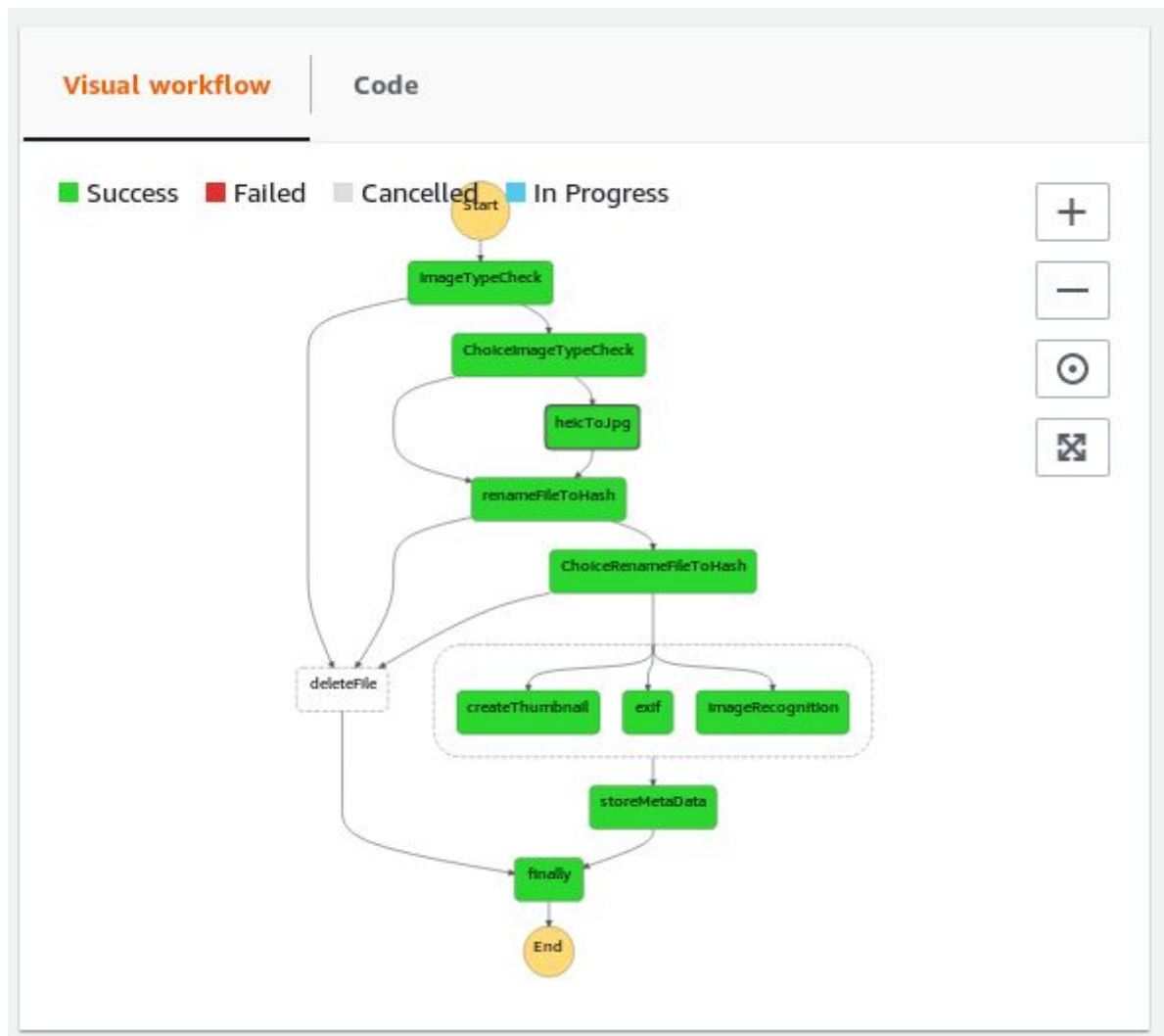
När alla dessa steg är färdiga kommer man till finally lådan som sparar information om hur det gick till en databas. Klienten börjar att fråga databasen när filen är uppladdad om det är färdigt och upprepar processen till den har fått svar.

Visualisering av state machine

Bilden här under visar på hur vår state machine ser ut i AWS. En väldigt stor fördel med AWS state machines är att man visuellt kan se vilken väg som har tagits vid exekvering.



På bilden under kan man se vilken väg en bild har tagit efter att den har lagts upp och det går att se att bilden har gått igenom hela state machinen utan problem då alla funktioner har exekverats utan att några undantag har kastats eller att något har gått fel.



Avvikelser

Det var vissa krav som vi inte hann att fixa, dessa krav var:

kravet 1.1 uppladdning av bild med hjälp av url

kravet 3.0 användaren ska kunna se sina bilder som har laddats upp.

kravet 3.1 att man som användare ska kunna se bildens metadata på klienten

Dessa avvikelser uppkom efter att vi dragit slutsatsen att vi inte skulle hinna med allt och frågade vår kund vad som var viktigt och hur vi skulle prioritera. Det som vi kom fram till vara vi skulle fokusera mera på back-end delen av systemet för att kunna ge kunden så mycket findings som möjligt till findings dokumentet.

Totalt så lyckades vi med 73% av alla krav som vår kund ville ha med i applikationen, ett findings dokument och en prestation.

Slutsats

Vi i gruppen är mycket nöjda med projektet som levererats och kommer att presenteras inom en snar framtid. Då ingen av gruppmedlemmarna sedan tidigare arbetat med AWS så blev det en stor tröskel att ta sig över, vilket vi nu känner att vi gjort. Vi har hela tiden försökt att alla gruppmedlemmar ska vara delaktiga i användningen av AWS så att alla får kunskap om det och hur man använder alla services, vilket vi känner att vi har uppnått. I och med att projektet var ett proof-of-concept där vi var väldigt fria i att utforska hur vi skulle göra, och även till ganska stor del vad som det slutgiltiga systemet skulle innehålla. Vi tycker att vi har implementerat de viktigaste baskraven och vi känner att vi kan besvara frågan om kunden borde använda AWS för sitt eget system med ja.

Gruppen som helhet har fungerat mycket bra och vi är väldigt nöjda med vår egen prestation och sammanhållning. Alla har varit delaktiga i princip alla olika steg under utvecklingen och har tagit eget ansvar när så har krävts.

Då vi direkt i projektet kände att vi hade ganska fria tyglar och kunde snabbt dra igång med att lära oss AWS så blev vår dokumentation lidande. Om vi är väldigt nöjda med vår implementation så är vi inte helt lika nöjda med dokumentationen. Vi känner dock att dokumentationen har förbättrats under projektets gång och då med stor hjälp av den feedback från vår opponerings-grupp som hjälpt oss mycket i vad och hur vi bör skriva och göra om dokumentationen. Vad som även blev lite lidande var vår tidrapportering och tidsplanering. Då det var extremt svårt att veta vad de olika gruppmedlemmarna skulle jobba med nästa iteration så var det till en början svårt att skriva iterationsplaner. Detta var på grund av att man hela tiden hittade ny dokumentation eller sätt att implementera en funktion som kanske inte var exakt det man höll på med just nu, och det blev därför mycket hoppande mellan vilka delar man arbetade med. Detta gjorde även att tidrapporteringen blev svår att slutföra på ett bra sätt då som sagt det kunde vara svårt att veta vad man faktiskt hade arbetet med då man hoppade runt väldigt mycket. Tidrapporter hade dock gjorts på ett annat som om vi gjort projektet igen genom ett program eller liknande så att vi hade kunnat få ut diagram och liknande över våra tider, då vi nu endast skrev dessa i vår wiki på bitbucket. Tidsplaneringen och rapporteringen blev dock bättre över tiden då vi i senare

skeden av projektet fick bättre kontroll över allt, också detta efter feedback från den andra gruppen. I början var vi som sagt tidigare ivriga att köra igång med implementationen och detta märktes på kvalitén av all dokumentation.


Kontakten med kunden tycker vi har fungerat bra. Då kunden lämnade oss ganska fria så har i tagit mycket eget ansvar i vad vi har behövt göra. Vi har dock haft kunden tillgänglig på Slack för frågor och vi har även under större delen av projektet suttit på kundens kontor minst 1 dag i veckan både för att ha möten men också för att kunna diskutera problem och lösningar mera informellt. Vår kontakt hos kunden har dock varit upptagen en del så vi har inte lyckats hålla möten varje vecka, vilket dock inte har behövts. Vid våra träffar har vi dock ofta visat vår prototyp för att få feedback och vi har även diskuterat mycket om hur vi ska prioritera och på vad, vilket har varit väldigt givande.

De projektmöten vi har haft tillsammans med vår handledare Tobias tycker vi även dom har varit bra. Vi har dock alltid haft en bra plan och har egentligen inte stött på några stora problem vilket har gjort att vi inte har fått ut så mycket av möten. De har dock varit bra för att hålla koll på deadlines och vad som behöver göras inför dessa. Den feedback som vi fått från handledaren angående vår dokumentation har dock varit mycket bra och tillsammans med feedbacken från vår opponerings-grupp så har vi strukturerat om dokumentationen mycket och är i slutändan nöjda med den med. Vi tycker att alla möten och presentationer under projektets gång har varit bra, lagom långa och även om man inte alltid har fått ut så mycket från just mötena så har de varit en bra fast punkt varje iteration och vi kan inte se hur dessa skulle ha gjorts annorlunda. Den dokumentation som funnits runt och till kursen och projektet har hjälpt oss mycket även den hur och vad vi behöver ha med till de olika delarna, framför allt till opponering och dokumentation. Den har varit mycket hjälpsam.

Då vi har haft vår presentation och officiella överlämning till kund ännu är det tyvärr svårt att veta om vi har motsvarat kundens krav. Vi har dock under projektets gång fått riktigt bra indikationer på att vi gör det kunden har velat och när vi visat prototyper och diskuterat lösningar så har vi alltid fått bra respons. Då projektet är gjort i AWS så finns det många olika lösningar på samma problem och hade vi gjort projektet igen är det inte säkert att lösningen hade blivit densamma. Som det är nu kan vi dock inte se att vi hade gjort projektet annorlunda då ett av kraven faktiskt var att göra det i AWS. Under projektets gång så funderade vi ofta på om AWS verkligen är en bra lösning att bygga liknande system i då det finns alternativ i t.ex. google som har liknande system. Men efter att ha utvärderat AWS så

kan vi säga att det är otroligt kraftfullt och trots sin höga inlärningströskel så skulle vi gärna arbeta med det igen.

Vi är även nöjda med hur testningen har fungerat i projektet. Vi satte tidigt Alex som ansvarig på testning som har läst in sig på hur man bäst testar en backend i AWS och har skött testning av de delar som har gått kontinuerligt under hela projektet. I slutet av projektet genomförde vi ett större stresstest där vi lät alla som ville att ladda upp bilder till vårt system för att försöka hitta problem med det. Detta gjordes via Slack där vi informerade om hur och när och från detta så fick vi fram mycket bra data, och även en del fel som i sedan kunde åtgärda. Som kan ses på bilden på nästa sida så kördes vårt system runt 1000 gånger under denna testning varav 688 gick utan problem medans 312 blev fel. Dessa fel har vi identifierat och åtgärdat och vi är därför väldigt nöjda med hur testningen har gått.

| | Name | Creation date | Status | Running | Succeeded | Failed |
|--|-------------------------|---------------------------------------|--------|---------|-----------|--------|
|  | onFileUploadCoursepress | May 31, 2019 07:54:17.536 AM | Active | 0 | 688 | 312 |

Resultatet av vårt projekt är att vi både utifrån resultatet av projektet och själva utvecklingen av det rekommenderar vår kund att använda AWS för att bygga ett liknande system som kan integreras i deras nuvarande arkitektur. Vi hoppas att kunden kommer ta lärdom av problem och utmaningar som vi har haft för att på ett så smidigt och enkelt sätt kan skapa ett system som kan ersätta deras nuvarande.

Övertagande organisation

Arbetet som vi har gjort är ett proof-of-concept för att ta reda på så mycket som möjligt om hur det kan fungera med filuppladdning på Amazon. Kunden är mest intresserad av våran sammanfattning av för och nackdelar vilket vi har i ett findings dokument. Det är även planerat ett möte där vi gör en överlämning och redovisar för våra upptäckter med att arbeta med filuppladdning på Amazon. Vi tror att det inte är speciellt intresserad av att ta våran kod rakt av och skicka ut den i produktion utan mest våra slutsatser av att använda denna tekniken.

Förslag till förbättringar inför kommande projekt

Eftersom ingen av oss i gruppen hade använt varken Amazon eller någon form av serverless lösning tidigare var planeringen av projektet ganska svårt. Det var en väldigt tekniskt svår uppgift med många i början av projektet okända faktorer. Det som vi gjorde bra var att vi delade upp oss i olika områden och sedan träffades och sammanfattade det vi hade gjort.

Det som också fungerade väldigt bra var att vi använde en väldigt agil process. Vi gjorde research inom ett område och tog sedan ett beslut om detta var en bra eller dålig lösning och anpassade oss till den. Vi fick skapa hypoteser om hur det skulle fungera och faktiskt testa om det fungerar som vi hade tänkt oss. Ibland gjorde vi fel val men då var vi snabba med att byta till något annat sätt som faktiskt visade sig vara bättre.

Det som var mindre bra var att eftersom samtliga i gruppen var på campus så satt vi mycket tillsammans och arbetade. När vi inte satt tillsammans och arbetade så kommunicerar vi en hel del på slack. Resultatet av detta var att vi inte använde issues eller dokumenterade i wikin för att kommunicera brister i systemet. Kunskap och lösningar kunde ibland försvinna i flödet. Resultatet av detta var att det ibland missades lösningar som vi redan hade gjort.

Något som vi hade väldigt mycket hjälp av i gruppen var att vi på ett tidigt stadium satte oss ned och ritade upp en generell arkitektur. Denna uppdaterade vi kontinuerligt under hela

projektet när vi kom till lägen där vi behövde fatta nya beslut om hur vi skulle göra. Dessa ritningar blev ett väldigt bra sätt att kommunicera inom gruppen hur vi tänkte och hur vi skulle göra.

Förslag på vidareutveckling

För smidigare upplevelse hade varit vi velat implementera WSS (websocket), för tyvärr i lösning nu är att servern sitter och frågar databas hela tiden om det har kommit in något nytt för att skicka notis till användaren.

Använda bättre repository hanterar, Bitbucket har inte funkat så smidigt, dels beror det på att det har varit väldigt slö ibland och bete sig på ett annorlunda sätt där t.ex. det frågar om inloggningsuppgifter för att kunna pusha till repo men ändå så går det att pusha utan att behöva skriva inloggningsuppgifter.

Intressant hade också varit att produktionssätta vårt system eller ett liknande system tillsammans med kundens applikation för att se hur det fungerar i produktion.

Något som också hade varit intressant hade varit att skapa ett system i t.ex. googles web services för att kunna jämföra vilket som är bäst ur flera olika synvinklar. Dels hur lätt det hade varit att utveckla projektet med tanke på inlärningströskel och hur lätt det är att hitta i dokumentationen etc. Delas ur synvinkel av exekveringstid, kostnad för "samma" system och hur det faktiskt fungerar.

Litteraturförteckning & dokumentationshänvisning

Projekthänvisningar

- Naviga: <https://navigaglobal.com/>
- InfoMaker (Företaget rebrandade under projektes gång, hette innan InfoMaker): <https://www.infomaker.se/>
- Projektdokumentation: <https://bitbucket.org/infogrooup/file-uploading/wiki/Home>
- Source code: <https://bitbucket.org/infogrooup/file-uploading/src/development/>

Dokumentationshänvisningar för tekniker och verktyg

- NodeJs: <https://nodejs.org/en/>
- ReactJs: <https://reactjs.org/>
- Uppy: <https://uppy.io/>
- Uppy companion server: <https://uppy.io/docs/companion/>
- AWS: <https://aws.amazon.com/>
- JavaScript: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Bitbucket: <https://bitbucket.org/>
- Ant design (användes för styling med React): <https://ant.design/>