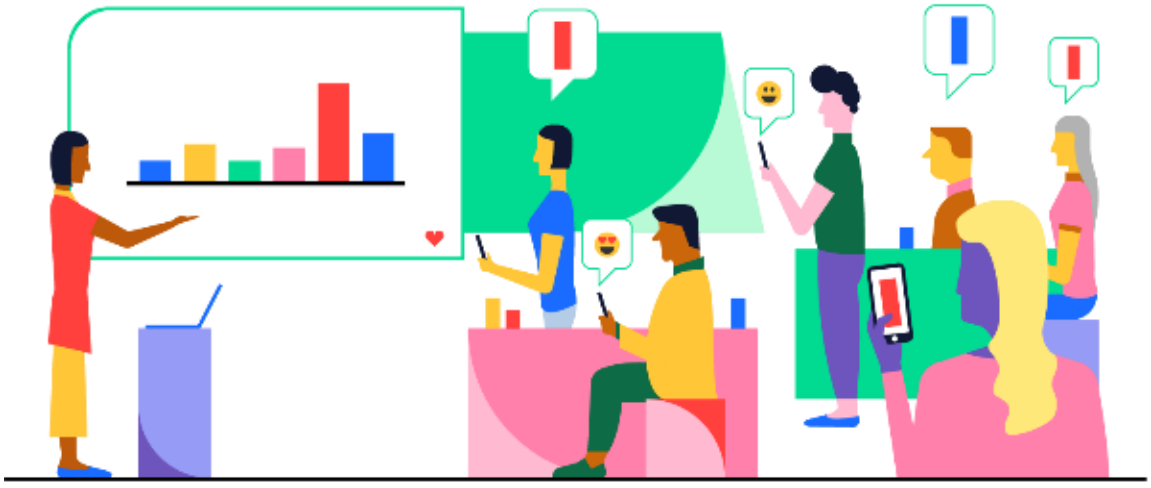


Produktbeskrivning
Reconciliation Script

Applikation för kundfordran hos Mentimeter

Innehållsförteckning

Introduktion

Mentimeter har ett skript skrivet i skriptspråket Typescript (Microsoft 2021) vars funktion är att jämföra finansiell data mellan filer och ett finansiellt system. Applikationen är ursprungligen implementerad uteslutande för internt bruk på Mentimeter. Syftet med applikationen är att avlasta IT-personal genom att erbjuda ett verktyg åt ekonomipersonal med lekmannamässig datorvana att sammanställa, arkivera och historiskt överblicka digitala kundfordranrapporter.

Erbjudande

Produkten gör det möjligt för personal utan avancerad IT-kunskap att genomföra den finansiella jämförelsen med hjälp av ett användarvänligt grafiskt gränssnitt.

Produkten är tillgänglig via en hemsida där personal kan ladda upp de filer som ska jämföras. Applikationen kommer sedan att ta dessa filer och köra det skript som Mentimeter tillhandahållit. Skriptet kommer returnera ett resultat som sedan presenteras på hemskärmen. Detta resultat sparas i en databas.

På hemsidan finns möjlighet att se information om tidigare jämförelser. Dessa visas upp i en tabell.

Unika egenskaper

Applikationen är skriven nästan utanslutande i JavaScript. Skriptet som blev tilldelat av produktägaren är skrivet i Typescript, vilket är ett superset av JavaScript (Tech World 2019). Applikationens serverdel drivs med hjälp av Node.js (Node 2021). Att få Node.js att översätta JavaScript-kod till maskinkod och samtidigt bearbeta Typescript-kod är möjligt men kräver implementation av Node.js:s Typescriptmotor och åtgärder för att få denna att samverka med den övriga exekveringen av JavaScript-kod (Github 2020). Istället förutsätter applikationen att skriptet är omkompilerat till JavaScript i förväg. Applikationen behandlar därefter skriptet som en “black box”, och programmerar abstrakt mot dess interface.

Specifikation

Identifierade distinkta användare är anställda hos ekonomiavdelningen och IT-administratörer.

Specifikation för ekonomipersonal

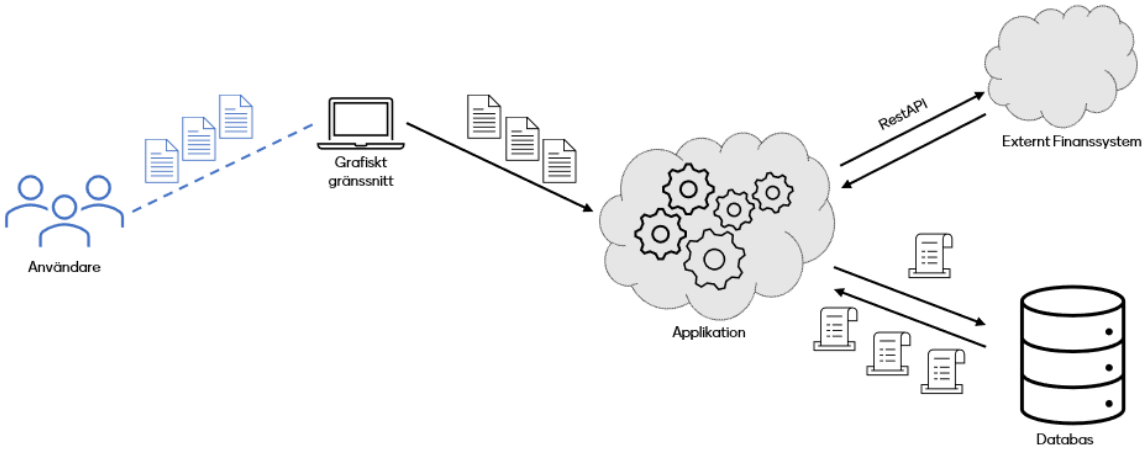
Applikationens frontend är byggd i JavaScript-biblioteket React.js. Förutsatt att servern är aktiv kan användaren nyttja applikationen genom att mata in serverns URI och portnummer i en webbläsares adressfält. UI-funktionaliteten körs på klientsidan (localhost). Sidan ger användaren möjlighet att ladda upp filer i par av typerna PDF och CSV från hårddisken till företagets egna server. Det är även där som all djupare funktionalitet som bl.a. filbearbetning, sammanställning av rapport och kommunikation med databas sköts. Användaren kan även navigera vidare till en vy där hen med ett knapptryck kan hämta in och rendera en lista av tidigare arkiverade rapporter från företagets PostgreSQL-databas.

Specifikation för IT-administratörer

Applikationens backend är huvudsakligen skriven i JavaScript och körs helt och hållet på företagets server via JavaScript-motorn Node.js. Det är IT-administratörens uppgift att hålla applikation, server och företagets databas igång för att möjliggöra ekonomipersonalens nyttjande av applikationen. Detta görs via terminalkommando på serversidan.

Applikationen är implementerad mot ett JavaScript interface tillhörande det skript som dikterar hur betalningskalkylerna ska utföras baserat på informationen i de uppladdade filerna. Skriptfilerna förvaras i en dedikerad mapp och företaget ansvarar för att utforma dem korrekt, kompilera dem till JavaScript och placera dem i mappen för körning.

Systemarkitektur



Figur 1 Generell systemarkitektur över applikationen och dess samverkan med grafiskt gränssnitt, databas och externt finanssystem

Produkten är uppbyggd enligt Server-Klient-designmönstret (TechTerms, 2016) där det grafiska gränssnittet utgör klienten och applikationens kod körs på en server. Användaren anropar det grafiska gränssnittet via en webbläsare, under förutsättning att användaren har tillgång till Mentimeters nätverk. Användaren väljer filer för utvärdering. För en funktionell körning av programmet krävs både en CSV och PDF version av filerna, enligt krav från produktägaren. Båda exemplaren måste heta samma sak och detta är även enligt krav från produktägaren. Användaren kan ladda upp fler filer än två, men varje par av filer måste stämma överens enligt angivna krav ovan.

Applikationens serverkod innehåller skriptet från Mentimeter. Skriptet gör en jämförelse mellan filerna och det externa finanssystemet. Serverkoden hanterar även all kommunikation till en PostgreSQL databas. I PostgreSQL databasen lagras resultatet av alla jämförelser mellan användarens finansiella filer och det externa finansiella systemet.

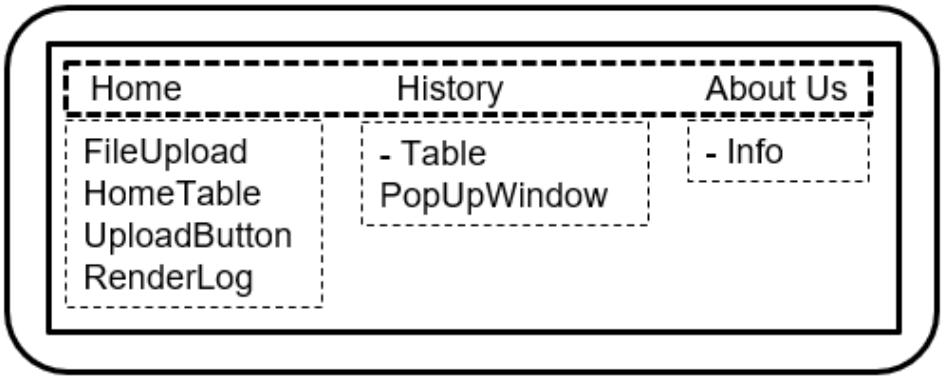
Grafiskt gränssnitt

Det grafiska gränssnittet utgörs av komponenter skrivna i React och JavaScript vilket gör gränssnittet modulärt. Modulariteten gör det möjligt att enkelt underhålla och byta delar av gränssnittet utan att påverka hela gränssnittet.

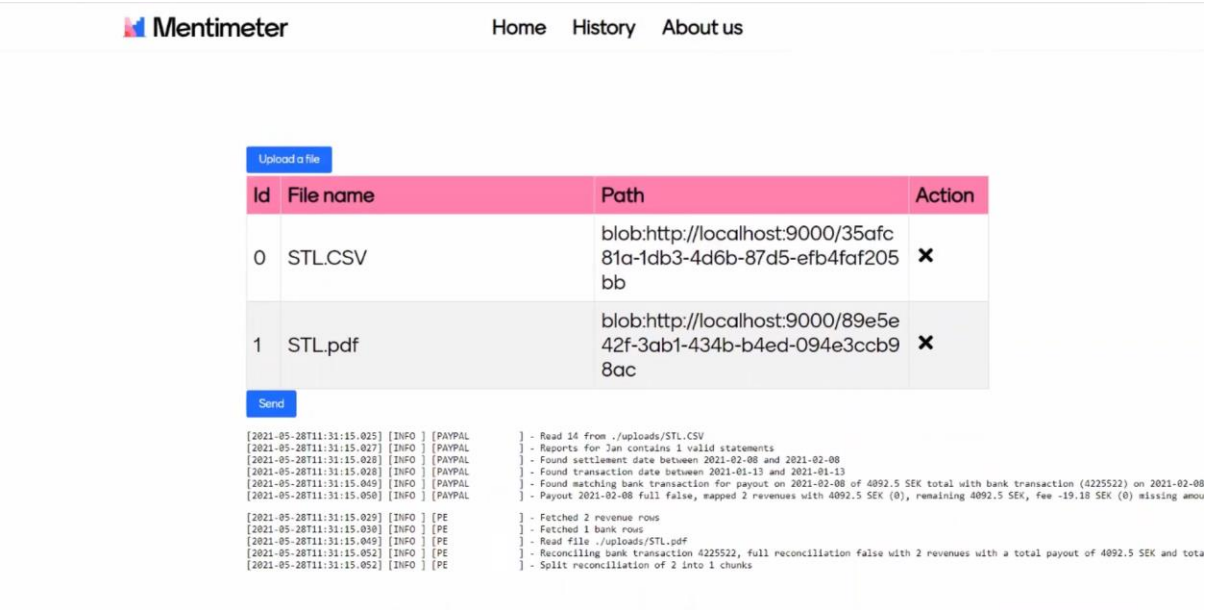
Komponenterna i gränssnittet:

- App
  - Grunden för hela gränssnittet
  - Håller koll på sidans URI
- Home
  - Gränssnittets första sida
  - Innehåller komponenter som sköter kommunikation för filuppladdning, kommunikation med skriptet och servern.
- FileUpload
  - Sköter kommunikationen med användarens filsystem
- HomeTable
  - Visar uppladdade filer

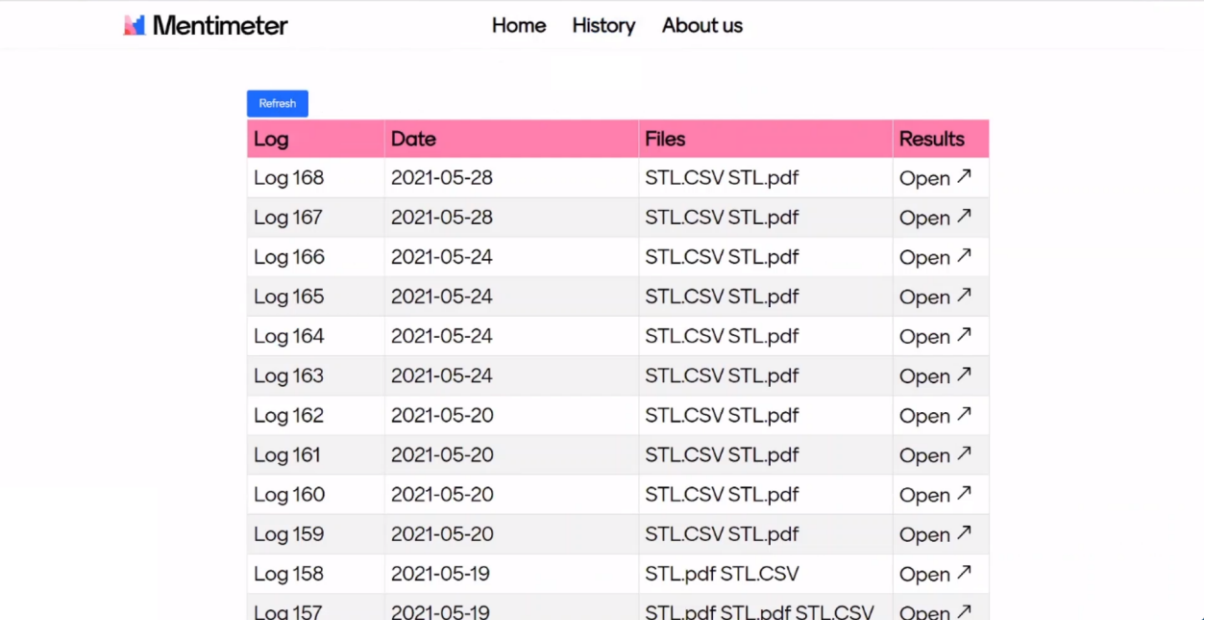
- UploadButton
  - Sköter kommunikation med skriptet och databasen
- RenderLog
  - Visar resultatet från skriptet
- History
  - Visar och sköter kommunikation med databasen
- Popup Window
  - Visar mer utförlig information från databasen
- NavigationBar
  - Sköter och visar navigationsmenyn på det grafiska gränssnittet
- AboutUs
  - Visar kort information om oss i gruppen



Figur 2 - Översikt komponenter. (“-” framför innebär att detta är ett element och inte en egen komponent)



Figur 3 - Exempel av vyn för Home efter uppladdning av filer. Texten under tabellen är resultatet från svarsloggen.



Figur 4 - Exempel av vyn för History. Innehållet av varje historiskt inlägg kan granskas ytterligare genom att klicka på dess Open-knapp och visas i en popup.

Överväganden

Vilket ramverk ska användas för applikationen? Frågan har inneburit ett val mellan att bygga applikationen enbart i HTML, CSS och JavaScript eller alternativt att bygga i React.js eller Next.js för att sedan kombineras med HTML, CSS och JavaScript.

Vår produktägare föreslog Next.js i början av projektet. Next är ett full-stack ramverk för utveckling av nätbaserade applikationer. Förslaget hade dock nackdelen att förkunskaperna som krävdes för att effektivt kunna använda detta ramverk var för stora för att tillskansas under den begränsade tidsram som fanns för projektet. Istället valdes React-biblioteket eftersom det är bättre lämpat för grafiska gränssnitt medan Next.Js är bättre lämpat för serverlösningar. React har existerat länge och det finns mycket information tillgänglig för enkel inlärning. React underlättar klient utveckling och körning av produkten. Mentimeter hade sedan en tid tillbaka byggt sina projekt i React så ytterligare en applikation som använder samma bibliotek skulle göra senare hantering och förändringar av applikationen lätt för anställda utvecklare.

Lösningen att bygga enbart i HTML, CSS och JavaScript var det första tillvägagångssättet som presenterades för oss. Den blev dock utkonkurrerad senare i utvecklingsprocessen p.g.a. de ovan nämnda fördelarna med att använda sig av React-biblioteket som grund.

Ett beslut som togs gemensamt mellan utvecklingsteamet och produktägaren var att bygga hela produkten som en applikation. Alternativet till det var att bygga det som en tvådelad lösning, det grafiska gränssnittet hade då utgjort en egen applikation och servern en annan applikation. Hade produkten byggts som en tvådelad lösning hade det grafiska gränssnittet behövt anropa servern som ett externt system med hjälp av ett API.

Källförteckning

Microsoft. 2021. *TypeScript Language.* 05. Hämtad: 26-05-2021. https://www.typescriptlang.org/.

Tech World IDG. 2019. *Vad är det som plötsligt gör språket TypeScript så populärt?* Hämtad: 26-05-2021. https://techworld.idg.se/2.2524/1.716630/typescript-framtidens-javascript.

Node. 2021. *About | Node.js*. Hämtad: 02-06-2021. https://nodejs.org/en/about/.

Github. 2020. *TypeStrong/ts-node: TypeScript execution and REPL for node.js*. Hämtad: 02-06-2021 https://github.com/TypeStrong/ts-node

TechTerms. 2016. *Client-Server Model?* Hämtad: 26-05-2021. https://techterms.com/definition/client-server_model.

Appendix

Installationsanvisningar

1. Installera Node.js och PostgreSQL
2. Starta PostgreSQL databasen
3. Kör skriptfilen ’database.sql’ i databasen
4. I en kommandotolk, navigera till mappen ’backend’
5. Kör kommandot ’node server’

Efter att ha följt anvisningarna ovan är applikationen igång och så länge installationen har skett på någon form av web hosting server kan användarna använda applikationen.

Detaljerade tekniska beskrivningar av intressanta delar

En intressant utmaning under arbetet med att ta fram produkten var översättningen av skriptet från Typescript till JavaScript. Skriptet som används för bearbetning av filer var skrivet i Typescript vilket är en utvidgning av JavaScript. Problemet med detta var att kallande och exekvering av detta skript från vår applikation, skriven i JavaScript och driven med hjälp av Node.js, inte gick att tillämpa på ett enkelt sätt.

Testning

Testningen av produkten skedde löpande under utvecklingen men i större utsträckning mot slutet av utvecklingen. Testningen delades in i olika delar beroende på storlek och komplexitet. Testen utformades också individuellt efter funktionaliteterna hos varje testobjekt och är anpassade efter de krav som de ska uppfylla. En kort förklaring på vilka delar som har testats och hur sammanställs här.

Testindelning:

- På klientsidan:
    - Filhantering
    - Filförmedling till server
    - Komponenter
    - Felvisning
- På serversidan:
    - Filhantering
    - Script
    - Databas
    - Kommunikation med klient
    - Felhantering

- Filhantering (klient):
    - Funktion: välja nya, visa nuvarande och borttagning av filer
    - Testning: uppladdning och borttagning av filer (legala och illegala för en körning)
    - Problem: indexering av valda filer vid olika uppladdningar och borttagning misslyckas vid upprepade uppladdning
    - Lösning: tilldelning av ID vid upprepad uppladdning skrivs om så det fungerar som tänkt
- Filförmedling till Server:
    - Funktion: läsa uppladdade filer och förmedla dessa till server för vidare hantering

    - Testning: skicka filer och information i olika format och storlek
    - Problem: formatering av data för skickande
    - Lösning: formatera utefter bestämd json struktur
- Komponenter:
    - Funktion: element som står för funktionalitet och visning av data
    - Testning: olika indata för att se att element förhåller sig liknande oberoende av vad som ska visas
    - Problem: vid uppdatering av data uppdaterar inte komponenter vad som visas
    - Lösning: placering och delning av state-data och funktioner som styr uppdatering för komponenter
- Felhantering
    - Funktion: Vid inkorrekt hantering av programmet (val av filer, inga filer serverproblem...) förmedla detta till användare
    - Testning: Låt applikation kasta fel vid alla delar som kan kasta fel och förmedla dessa för användaren
    - Problem: skillnad i format av felmeddelanden från olika delar av applikationen
    - Lösning: definiera en json-struktur på hur felmeddelanden ska vara formaterade
- Filhantering Server:
    - Funktion: ta emot filer, ladda upp dessa i relevant mapp, ta bort filer efter körning
    - Testning:
        - Ladda upp olika filer (för att se att format behålls och placeras i rätt mapp)
        - Fyll mappar med olika filtyper och olika namn (se att alla slags filer kan tas bort)
    - Problem:
        - Olika mängd filer går inte att lagra på samma sätt
        - Felloggar från ett skript exekvering tas inte bort
        - Misslyckad körning och felförmedling till klient kunde resultera i att servern slutade köras
    - Lösning:
        - Skild filhantering beroende på indata
        - Skild hantering av felloggar från uppladdade filer
        - Asynkron felförmedling möjliggjorde fångande av felmeddelande
- Script:
    - Funktion: blackbox från produktägare som ska kunna läsa filer som tilldelas och generera output
    - Testning: ladda filer till korrekt map och exekvera skriptet. Se om outputen som genereras är korrekt och placeras i output-map
    - Problem:
        - Samarbete mellan JavaScript och Typescript fungerade inte
        - Script exekvering är asynkron (trådad) och resulterar i att ordning inte följs
    - Lösning:
        - Kompilering av Typescript kod till JavaScript för att göra den kompatibel med resterande program
        - Göra hela funktionen för serverkommunikation asynkron och vänta in körningen av skript
- Databas:

    - Funktion: ta emot en sammanställning av skriptloggar och spara denna för senare visning
    - Testning:
        - Fylla databasen med sparade körningar av skriptet från application
        - Läsa data som sparats i databasen från applikationen
    - Problem: Format av data i applikationen går inte att spara och utvinna från databasen
    - Lösning: formatera om data som ska till databasen för lättare kunna spara och läsa från denna
- Kommunikation med klient:
    - Funktion: skicka utvärdering av en körning (felmeddelanden, loggar, databasinformation…)
    - Testning: skicka information i olika former (loggar, felmeddelanden, databasinformation…) och mängder från server till klienten
    - Problem: olika format på data som ska skickas
    - Lösning: standardisera format av data till för överföring

- Felhantering
    - Funktion: avbrott av programmet och förmedling av problem till klient
    - Testning: få varje del som kan resultera i en ofullständig körning att kasta fel, hantera dessa och skicka information till klient
    - Problem: beroende på vart fel kastas måste kommunikation med klient och hantering av fel göras på olika sätt. Detta p.g.a. hur programmet påverkas av olika fel som kan uppstå
    - Lösning: specifika lösningar för varje fel som kan uppstå istället för ett generellt sätt