CHALMERS

Visualisering av loggdata från Blurt

Examensarbete inom Data- och informationsteknik

ERIK ANDERSSON
WILLIAM CARPENFELT

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2026

Examensarbete 2026

Visualisering av loggdata från Blurt

ERIK ANDERSSON
WILLIAM CARPENFELT

CHALMERS

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborg 2026

Visualisering av loggdata från Blurt
Erik Andersson, William Carpenfelt

Handledare: John J. Camilleri
Examinator: Johannes Åman Pohjola

Examensarbete 2026
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg

Skriven i LaTeX
Göteborg 2026

Visualisering av loggdata från Blurt
ERIK ANDERSSON
WILLIAM CARPENFELT
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Detta examensarbete behandlar utvecklingen av ett verktyg för visualisering av loggdata från det webbaserade responsverktyget Blurt. Syftet var att möjliggöra analys av studenters aktivitet och engagemang genom tydliga och lättolkade visualiseringar.

Utvecklingsprocessen omfattade bland annat analys av Blurts loggformat, implementation av funktionalitet för inläsning och bearbetning av loggdata samt utveckling av interaktiva visualiseringar. Webbapplikationen utvecklades med TypeScript som huvudsakligt programmeringsspråk genom hela systemet i en Node.js-miljö.

På backend-sidan användes en struktur baserad på så kallade Data Transfer Objects (DTO), vilket möjliggjorde säker hantering och överföring av loggdata mellan backend och frontend. Backend-sidan av applikationen hade ansvar för bland annat inläsning och bearbetning av loggdata från Blurt, samt att lagra denna data i en PostgreSQL-databas för att sedan kunna skicka relevant data till frontend-sidan.

På frontend-sidan av verktyget användes ramverket Vue.js för att skapa en responsiv användarupplevelse. För att skapa dynamiska och lättolkade diagram användes biblioteket Chart.js, vilket möjliggjorde att applikationen kunde presentera visualiseringar baserade på den bearbetade loggdatan. Frontend-sidan hade ansvar för att presentera statistik och användaraktivitet på ett överskådligt och lättförståeligt sätt.

Resultatet är en fristående webbapplikation som kan visualisera användarstatistik i form av linje- och stapeldiagram. Detta kommer förhoppningsvis ge lärare och kursansvariga bättre möjligheter att utvärdera användningen av Blurt och dra slutsatser kring studenters deltagande. Lösningen utvärderades genom funktionella tester samt genom ett frågeformulär som personal på Chalmers fick fylla i. I formuläret fick personalen bedöma användbarhet och användarvänlighet, och ge förslag för framtida vidareutveckling.

Nyckelord: Blurt, loggdata.

Förord

Vi vill rikta ett stort tack till vår handledare John J. Camilleri som har gett konstruktiv feedback och stöd under projektets gång. Vi vill också tacka den personal på Chalmers som gav sin feedback på projektet via ett frågeformulär.

Innehåll

Figurer

1

Inledning

I detta kapitel presenteras bakgrunden till projektet, dess syfte och mål, samt de avgränsningar som gjorts.

1.1 Bakgrund

Blurt är en webbaserad applikation som utvecklats av John J. Camilleri vid Institutionen för Data- och Informationsteknik (Computer Science Engineering) på Chalmers tekniska högskola, där det används som ett digitalt responsverktyg för undervisning. Med hjälp av Blurt kan lärare ställa frågor i realtid som studenterna besvarar via sina egna enheter, vilket möjliggör snabb återkoppling och ger lärare en bättre bild av studenternas förståelse av kursinnehållet.

Applikationen är open-source och genererar kontinuerligt loggdata baserat på användarnas interaktioner med applikationen. Denna data inkluderar bland annat studenternas svar på frågor, tidsstämplar för när svaren skickas in samt information kopplad till specifika kurstillfällen.

Trots att denna insamlade data innehåller stor potential för vidare analys har det saknats ett effektivt och användarvänligt sätt att visualisera denna data. Detta begränsar möjligheterna att använda Blurt-data som stöd för pedagogisk utveckling och kursutvärdering, och för lärare och kursansvariga att dra slutsatser kring studentaktivitet.

Det finns alltså ett behov av en fristående applikation som kan läsa in och bearbeta rå loggdata från Blurt, och presentera denna data på ett överskådligt sätt genom olika visualiseringar.

1.2 Syfte

Syftet med detta projekt var att visualisera loggdata från det webbaserade responsverktyget Blurt för att ge meningsfulla insikter om studenters aktivitet och engagemang i undervisningssammanhang. Projektet syftade även till att bidra till förbättrade förutsättningar för lärare och kursansvariga att utvärdera användningen av Blurt som ett pedagogiskt verktyg.

1.3 Mål

Målet med detta projekt var att ta fram en fristående, open source-webbapplikation som kan läsa in, bearbeta och analysera rå loggdata från Blurt. Applikationen var tänkt att utformas med fokus på användarvänlighet och möjliggöra presentation av relevant data i form av lättolkade och överskådliga visualiseringar, i form av till exempel grafer och diagram.

De huvudsakliga graferna som skulle implementeras var stapel- eller linjediagram som kunde illustrera antal användare per dag och antal rum per dag. Utöver det så skulle också ett stapeldiagram där staplarna kunde delas upp i olika sektioner implementeras för att visa svarsfördelning på varje fråga i varje enskilt rum.

På så sätt skulle det enkelt gå att visualisera till exempel antalet elever som svarar på en fråga, svarsfördelning per fråga samt antal användare, rum och besvarade frågor per dag i realtid. Utöver det var det tänkt att applikationen skulle vara modulärt uppbyggd och att utvecklingsprocessen skulle vara väldokumenterad, för att underlätta framtida vidareutveckling.

1.4 Avgränsningar

Avsikten med applikationen var att endast ett begränsat antal fördefinierade visualiseringar skulle implementeras, medan ytterligare visualiseringar endast skulle utvecklas i mån av tid. Applikationen var inte heller avsedd att integreras i Blurts ordinarie användargränssnitt, utan var snarare tänkt att användas som ett fristående verktyg. Slutligen skulle applikationen endast anpassas för moderna webbläsare, och mobilanpassning var ingen prioritet.

En annan avgränsning som påverkade hur applikationen behövde byggas var det faktum att Blurt och dess loggdata inte skulle behöva modifieras på något sätt. Formatet på loggdatan som Blurt generar fick inte ändras eftersom den nya applikationen skulle byggas fristående från Blurt. Anledningen till att loggdatan inte skulle modifieras var att det redan fanns ett stort antal genererade loggfiler, och att modifiera dessa filer hade krävt mycket extraarbete. Eftersom loggdatan var strukturerad på ett visst sätt och innehöll begränsad information så påverkade det vissa val i designprocessen.

2

Teknisk bakgrund

Denna sektion beskriver de tekniker och ramverk som använts i projektet.

2.1 Teknikval

På grund av att Blurt redan kördes i en containerbaserad miljö med Docker beslutades det tidigt i utvecklingsprocessen att även denna applikation skulle utvecklas som en egen container som kunde kommunicera med de befintliga tjänsterna. Docker användes i projektet för att starta och hantera de befintliga containrarna i Blurts miljö, samt ytterligare containers för visualiseringsapplikationen och dess databas.

2.2 Blurt

Blurt är ett webbaserat publikresponsverktyg som används för att samla in svar från en publik i realtid, och är utvecklat av John J. Camilleri vid Institutionen för data- och informationsteknik på Chalmers tekniska högskola. [1] I undervisningssammanhang kan verktyget användas av lärare för att ställa frågor under föreläsningar, där studenterna svarar via sina egna enheter. På så sätt kan läraren snabbt få en uppfattning om studenternas förståelse, åsikter och aktivitet under ett undervisningstillfälle.

Användningen av Blurt bygger på att en presentatör, exempelvis en lärare, skapar ett rum. Publiken kan därefter ansluta till rummet genom att ange rummets namn eller genom att skanna en QR-kod. När deltagarna har anslutit kan presentatören välja vilken typ av fråga som ska ställas. Blurt stödjer flera olika svarsformat som textsvar, numeriska svar, ja/nej/kanske-frågor och flervalsfrågor. När publiken skickar in sina svar visas resultatet direkt i applikationens gränssnitt, vilket möjliggör snabb återkoppling under pågående aktivitet.

En viktig egenskap hos Blurt är att applikationen är utformad med en minimalistisk design. Systemet saknar exempelvis användarkonton, inloggning, användarnamn och förberett frågematerial. Detta gör verktyget enkelt att använda, men innebär samtidigt att användare inte kan identifieras på samma sätt som i system där varje användare har ett konto. I stället hanteras deltagare genom temporära id-nummer som skapas när en användare ansluter till ett rum. Detta påverkar hur loggdatan

kan tolkas, eftersom ett användar-id inte nödvändigtvis motsvarar en unik fysisk person över längre tid. Dessutom får samma användare ett nytt unikt id i samma rum om användaren lämnar rummet och sedan ansluter till det igen.

Alla centrala interaktioner i Blurt skickas till servern i realtid och skrivs till loggfiler. Loggfilernas data inkluderar bland annat när rum skapas, när användare ansluter till eller lämnar ett rum, när frågor ställs, när svar skickas in och när svar rensas från skärmen. Utöver det så kan också ändringar av svarsformat och upprepade svar från samma användare förekomma i loggdatan. Det innebär att loggfilerna inte enbart innehåller slutresultatet för varje fråga, utan även en mer detaljerad händelsehistorik över hur rummet har använts.

Blurt sparar loggdata i dagsbaserade loggfiler, där varje fil innehåller händelser från flera rum under samma dag. För att kunna analysera användningen av Blurt behöver denna råa loggdata därför tolkas, filtreras och struktureras. Exempelvis behöver händelser kopplas till rätt rum, frågor kopplas till rätt frågetyp och svar kopplas till rätt fråga och tidpunkt. Denna strukturering är en central förutsättning för att loggdatan senare ska kunna visualiseras på ett överskådligt sätt, som till exempel via diagram som visar svarsfördelning, antal svar per fråga eller aktivitet över tid.

2.3 Node.js

Node.js är en exekveringsmiljö som gör det möjligt att köra JavaScript-kod utanför en webbläsare. Plattformen används huvudsakligen inom mjukvaruutveckling för att bygga skalbara nätverksapplikationer och serverlösningar. [2] I detta projekt användes Node.js för hantering av filinläsning, databehandling och kommunikation mellan frontend och backend. Node.js användes även som utvecklingsmiljö för både frontend- och backendapplikationen.

Node.js användes genom hela systemet för att möjliggöra användning av samma programmeringsspråk och utvecklingsmiljö i både frontend och backend, vilket förenklade utvecklingen av systemets olika delar.

2.4 Vue.js

Vue.js är ett komponentbaserat JavaScript-ramverk som bygger på vanlig HTML, JavaScript och CSS, och används för att skapa dynamiska användargränssnitt. Ramverket Vue.js användes för att dynamiskt uppdatera användargränssnittet, eftersom det är väl anpassat för utveckling av interaktiva och komponentbaserade webbapplikationer.

I Vue delas gränssnittet upp i komponenter, där varje komponent kapslar in struktur, logik och stil för en viss del av applikationen. En komponent kan exempelvis representera en navigationsmeny, ett diagram, en lista eller en hel vy. Två centrala egenskaper i Vue är deklarativ rendering och reaktivitet. Deklarativ rendering innebär att användargränssnittets utseende beskrivs utifrån komponentens interna till-

stånd, i stället för att den så kallade dokumentobjektmodellen (DOM) manipuleras manuellt med JavaScript. Detta sker genom Vue:s template-syntax, där komponentens data kan bindas till HTML-strukturen. Reaktivitet innebär att Vue automatiskt spårar förändringar i komponentens data och uppdaterar de delar av DOM:en som påverkas av förändringen. [3]

I Vue byggs användargränssnitt ofta upp som en komponenthierarki. Detta innebär att applikationen kan ses som en trädstruktur där en komponent längre upp på trädet innehåller en eller flera underkomponenter. Roten i trädet är vanligtvis applikationens huvudkomponent, medan det längre ner i trädet finns mer specialiserade komponenter som exempelvis vyer, diagram, knappar och listor. Genom denna struktur kan större gränssnitt delas upp i mindre och mer avgränsade delar, där varje komponent ansvarar för en specifik del av funktionaliteten eller presentationen. Detta gör större projekt mindre komplexa genom att koden blir mer kompakt och lättare att följa. Varje komponent kan i sin tur använda ett obegränsat antal komponenter och varje komponent är i praktiken återanvändbar.

Kommunikationen mellan komponenter i Vue bygger vanligtvis på ett enkelriktat dataflöde. Det innebär att data skickas från en förälder-komponent till en barn-komponent genom så kallade props. Barn-komponenten använder denna data för att presentera innehåll, men bör inte ändra datan den får via props direkt. Om en användare interagerar med barn-komponenten kan den i stället skicka ett så kallat event till förälder-komponenten. Förälder-komponenten ansvarar därefter för att hantera detta event och uppdatera sitt interna tillstånd. När tillståndet ändras uppdaterar Vue.js de delar av komponentträdet som är beroende av den ändrade datan.

2.5 Chart.js

Chart.js är ett bibliotek för datavisualisering som används för att skapa dynamiska och interaktiva diagram i webbläsaren. I detta projekt användes biblioteket för att skapa grafer gällande användaraktivitet och svarsfördelning. [4]

Chart.js valdes framför andra alternativ såsom D3.js eftersom det erbjuder färdiga och lättimplementerade diagramkomponenter som passade projektets behov väl. Eftersom projektet endast behövde relativt standardiserade diagramtyper ansågs Chart.js vara en mer tidseffektiv lösning.

2.6 PostgreSQL

PostgreSQL är ett relationsdatabashanteringssystem som används för att lagra och hantera data. I detta projekt implementerades en databas för att lagra och hantera tabeller med rum, användare, frågor och svar utifrån Blurts loggdata. [5]

2.7 Express

Express är ett ramverk för Node.js som används för att bygga webbservrar och API:er. Det förenklar hantering av HTTP-förfrågningar och routing, och användes i projektet för att implementera kommunikationen mellan frontend och backend. [6]

2.8 Docker

Docker är ett program som används för att paketera och köra applikationer i isolerade miljöer som kallas för containers. Docker paketerar applikationens kod och dess beroenden i en så kallad Docker-image för att sen köra en container. En Docker-container kan liknas vid en virtuell maskin, men är mer lättviktig eftersom den delar värddatorns operativsystemskärna i stället för att köra ett helt eget operativsystem. Detta leder till bättre resurseffektivitet. [7]

Eftersom Docker-containers paketerar applikationens kod och beroenden och körs i en isolerad miljö så minskar risken för problem som beror på skillnader i operativsystem, installerade bibliotek eller konfigurering. Till exempel så behövs bara en Docker-image för att köra en PostgreSQL-databas lokalt på datorn, och PostgreSQL behöver alltså inte vara installerat på datorn. Detta underlättar arbetet i projektgrupper då det kan hända att olika personer jobbar i olika miljöer.

För att skapa en Docker-container krävs en Docker-image. En Docker-image fungerar som en mall och innehåller kod, inställningar, beroenden och instruktioner för hur en container ska köras. En Docker-image skapas genom en Dockerfil.

På grund av att Blurt redan kördes i en containerbaserad miljö med Docker beslutades det tidigt i utvecklingsprocessen att även denna applikation skulle utvecklas som en egen container som kunde kommunicera med de befintliga tjänsterna. Docker användes i projektet för att starta och hantera de befintliga containrarna i Blurts miljö, samt ytterligare containers för visualiseringsapplikationen och dess databas.

2.9 TypeScript

TypeScript är ett programmeringsspråk som bygger vidare på JavaScript. Den största skillnaden mellan språken är att TypeScript har statisk typning, medan JavaScript har dynamisk typning. Det innebär att typer i JavaScript bestäms när programmet körs, medan TypeScript kan kontrollera typer redan under utvecklingen.

I TypeScript kan variabler, funktioner och objekt deklareras med bestämda typer. Detta gör att många typrelaterade fel kan upptäckas innan koden körs. Exempelvis kan TypeScript varna om en funktion förväntar sig ett numeriskt värde men får en textsträng.

TypeScript körs inte direkt i webbläsaren, utan kompileras först till JavaScript. På så sätt kan TypeScript användas för att skriva mer strukturerad och kontrollerad

kod, samtidigt som resultatet fortfarande kan köras i vanliga JavaScript-miljöer. [8]

Webbapplikationen utvecklades med TypeScript som huvudsakligt programmeringsspråk genom hela systemet. Valet av TypeScript gjordes eftersom språket är utvecklat med webbapplikationer som huvudsakligt användningsområde och underlättar hantering av vanliga problem i moderna webbapplikationer, exempelvis fel vid kommunikation mellan klient och server.

2.10 Jest

Jest är ett JavaScript-baserat testverktyg som används för att skriva och köra automatiserade tester. Ramverket används främst för enhetstester och integrationstester. I detta projekt användes Jest för att testa backend-funktionalitet och säkerställa korrekt hantering av loggdata. [9]

2.11 API

Ett API (Application Programming Interface) är ett gränssnitt som kan användas för att få olika system och programvarukomponenter att kommunicera med varandra. I webbapplikationer används API:er ofta för att låta frontend och backend kommunicera via HTTP-förfrågningar och svar. [10] I detta projekt användes ett REST-baserat API för att hämta och överföra loggdata mellan frontend-sidan och backend-sidan av applikationen.

3

Genomförande

I detta kapitel beskrivs hur utvecklingen av applikationen genomfördes samt de viktigaste stegen i processen.

3.1 Analys av loggdata

Arbetet inleddes med en analys av Blurts befintliga loggformat och integrering av en databas för effektiv hantering och lagring av data. Figuren nedan visar ett exempel på loggdatan som Blurt genererade, där information såsom tidsstämpel, namn på rum, användar-id och händelse loggades.

```
2025-09-08 05:52:13  [dat555] create room
2025-09-08 05:52:14  [dat555] et1uKKAyvx02OLs2AAAB student connect
2025-09-08 05:52:14  [dat555] et1uKKAyvx02OLs2AAAB respond:
```

Kodexempel 3.1: Exempel på loggdata

Loggdatan innehöll begränsad information, vilket påverkade flera beslut i designprocessen. Ett exempel är att användare i Blurt tilldelas ett nytt användar-id varje gång de ansluter till ett rum. Detta gjorde att statistiken över antal användare inte kunde beräknas helt exakt, eftersom samma person kunde förekomma med flera olika id.

För att hantera detta valdes i stället en lösning där systemet beräknar hur många användare som befann sig i rummet vid specifika tidpunkter. Detta gjordes genom att bearbeta händelser där användare gick med i eller lämnade rummet, och därefter koppla det beräknade antalet aktiva användare till respektive tidsstämpel.

Även rum och frågor saknade unika id i loggarna. Därför behövde logik utformas för att bygga upp ett händelseförlopp i programmet, så att olika rum, frågor och användarsessioner kunde särskiljas och kopplas samman på ett så tillförlitligt sätt som möjligt utifrån den tillgängliga datan. För att minska risken för att samma data lagrades flera gånger sparades även tidsstämplar för rum, användare, frågor och svar.

Loggarna innehöll inte heller frågeställningen för varje fråga. Det innebär att applikationen inte kan visa själva frågetexten, utan endast den information som finns

tillgänglig i loggdatan, till exempel frågetyp, svar och tidsstämplar. Anledningen till detta är att frågorna ställs muntligt under en Blurt-session efter att frågekategorin valts, och det är endast svaren som lämnas in skriftligt.

3.2 Implementering av grafer

Utifrån Blurts loggdata och de problem som uppstod implementerades två grafer för att visualisera ett rums historik. Den första grafen visar frågor på x-axeln i kronologisk ordning. För varje fråga användes staplar för att visa svarsfördelningen. I samma graf visas även en linje som representerar antalet användaranslutningar där användaren haft möjlighet att besvara frågan. Detta antal beräknades genom att utgå från antalet användare som befann sig i rummet när frågan presenterades och därefter lägga till det antal användare som anslöt innan nästa fråga ställdes, eller inom fem minuter om ingen ny fråga ställdes under detta tidsintervall.

Eftersom användar-id:n inte kunde användas för att identifiera en unik person kan linjen inte garantera det exakta antalet unika personer i rummet. Den bör därför tolkas som en uppskattning av antalet möjliga svarande användaranslutningar, snarare än ett exakt mått på antalet faktiska personer.

Den andra grafen visar rummets tidslinje från den tidpunkt då rummet skapades till den tidpunkt då det stängdes. I denna graf representerar x-axeln tid, medan frågorna markeras vid de tidpunkter då de inträffade. Grafen innehåller även en linje som visar antalet aktiva användare vid varje frågas tidpunkt samt vid femminutersintervall under rummets livstid.

3.3 Prototyp

En prototyp byggdes för användargränssnittet, där olika tekniker för visualisering testades. Inledningsvis övervägdes D3.js, eftersom biblioteket ger stor flexibilitet och kontroll över hur visualiseringar utformas. Under implementationen valdes dock Chart.js, eftersom det erbjöd tillräcklig funktionalitet för projektets behov samtidigt som det gjorde det snabbare och enklare att skapa tydliga diagram. Dessutom utforskades biblioteket bootstrap som används för att enkelt och snabbt bygga grafiskt gränssnitt.

I ett tidigt skede beslutades det även att datan skulle struktureras med hjälp av Data Transfer Objects (DTO). Detta innebar att rå data från backend-sidan omvandlades till tydliga och avgränsade dataobjekt innan den skickades vidare till frontend-sidan. På så sätt separerades systemets lager, vilket gjorde koden mer modulär, lättare att underhålla och enklare att vidareutveckla.

3.4 Backend-utveckling

Backend-arkitekturen utvecklades med separata lager för datamodeller, dataåtkomst, tjänstelogik samt hantering och bearbetning av loggdata, vilket bidrog till en tydligare struktur och ansvarsfördelning i systemet.

En viktig del av backendutvecklingen var hanteringen av loggdata. Eftersom loggfilerna innehöll begränsad information behövde den omstruktureras innan den kunde användas i visualiseringarna. Detta innefattade bland annat att identifiera rum, koppla samman frågor och svar, hantera användaranslutningar samt beräkna statistik. Under utvecklingen identifierades dock prestandaproblem vid hantering av stora datamängder. Dessa problem hanterades genom att bearbeta loggdata i grupper vid behov, för att minska antalet databasoperationer.

Backend-sidan ansvarade även för databashantering och lagring av bearbetad data. Databasen utformades med relationer mellan rum, frågor och svar. Dessutom implementerades logik för att lagra tidsstämpeln för när specifika rum, frågor och svar skapades. På så sätt kunde ett internt händelseförlopp skapas så att graferna kunde visualiseras korrekt.

För att skapa en tydligare struktur användes så kallade Data Transfer Objects (DTO) vid överföring av data. Detta innebar att backend kunde anpassa datan i strukturerade objekt innan den skickades vidare till frontend-sidan, vilket förenklade visualiseringarna.

3.5 Frontend-utveckling

Frontendutvecklingen omfattade flera delar av applikationen. En central del var att bygga upp applikationens vyer och komponentstruktur. Applikationen delades in i separata vyer, exempelvis `Overview`, `Room` och `About`, där varje vy ansvarade för en specifik del av användargränssnittet. Dessa vyer delades i sin tur upp i mindre komponenter, exempelvis diagramkomponenter, listor, navigationskomponenter, sidväljare och tooltip-komponenter. Detta gjorde gränssnittet mer överskådligt och underlättade vidareutveckling.

Frontend-delen hade även ansvar för navigering mellan applikationens olika sidor. Med hjälp av Vue Router kunde användaren växla mellan översiktssidan, enskilda rum och informationssidan. Detta gjorde det möjligt att strukturera applikationen som flera separata vyer, trots att den fungerade som en sammanhängande webbapplikation.

En annan viktig del av frontendutvecklingen var att hämta bearbetad loggdata från backend genom API-anrop. Den data som hämtades behövde därefter anpassas till de olika visualiseringarna i gränssnittet. För att implementera detta så behövde data omvandlas till ett format som kunde användas av Chart.js, relevant information behövde filtreras fram, svar behövde grupperas och frekvenslistor för textbaserade

svar behövde skapas.

Frontendutvecklingen bestod också av att implementera interaktivitet i gränssnittet. Användaren behövde exempelvis kunna välja månad, välja rum, växla mellan olika grafvyer, klicka på frågor för att visa mer detaljerad information och använda tooltips för att få förklaringar till olika visualiseringar. Denna interaktivitet var viktig för att göra applikationen mer användbar och användarvänlig, samt för att ge användaren möjlighet att själv utforska loggdatan.

Eftersom Blurt innehåller flera olika frågetyper behövde frontend-sidan även hantera olika sätt att presentera svar på. Flervalsfrågor och ja/nej-frågor kunde visualiseras som svarsfördelningar i diagram, medan textfrågor presenterades genom listor och ordmoln. Detta krävde att gränssnittet kunde anpassa presentationen beroende på vilken typ av data som skulle visas.

Utöver detta utfördes även arbete när det kommer till layout och visuell design. Detta innefattade bland annat placering av grafer, listor och kontroller, samt användning av färger, CSS-variabler, hover-effekter och tooltips. Syftet var att skapa ett konsekvent och lättförståeligt gränssnitt där visualiseringarna kunde tolkas utan onödig komplexitet.

3.6 Arbetsprocess och återkoppling

Under projektets gång hölls veckovisa möten med John J. Camilleri, där arbetets framsteg presenterades och diskuterades. Dessa möten möjliggjorde kontinuerlig återkoppling, vilket bidrog till att applikationen kunde utformas på ett sätt som bättre motsvarade behoven hos lärare vid Chalmers. Som ett resultat av denna återkoppling implementerades ett flertal funktioner, medan vissa existerande funktioner modifierades och vidareutvecklades.

I en tidig version av applikationen innehöll startsidan två separata stapeldiagram. Startsidan motsvarade den senare `Overview`-vyn, som beskrivs mer utförligt i avsnitt 4.2.4. Det ena diagrammet visade antalet rum per dag för en vald månad, medan det andra visade aktiviteten i Blurt, mätt som antalet användare per dag under samma period. Vid ett möte föreslog handledaren att dessa grafer skulle slås ihop, eftersom de utgick från samma x-axel. Utifrån denna återkoppling implementerades en ny kombinerad graf där antalet rum fortfarande visualiserades med staplar, medan aktiviteten i stället visualiserades med en linje.

3.7 Testning

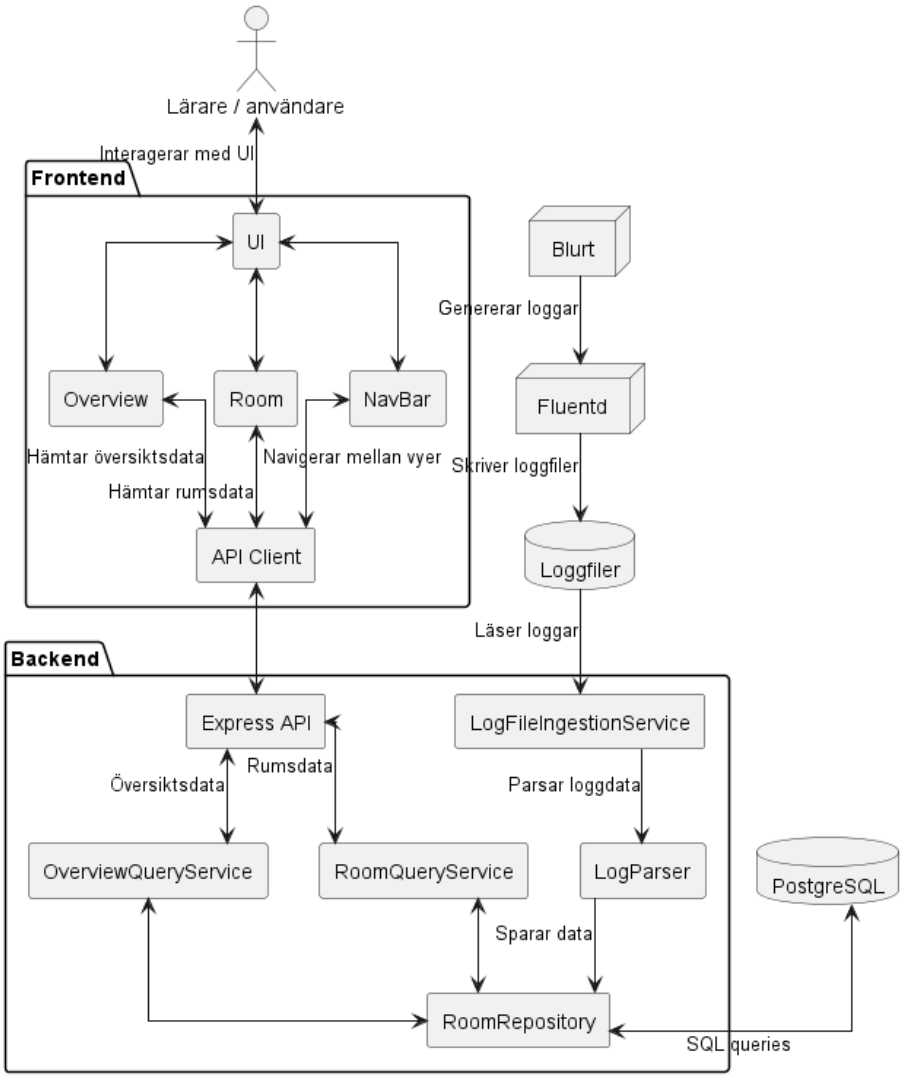
Ett antal tester skrevs också, med syftet att främst verifiera funktionaliteten i backend-delen av systemet. Fokus låg på att säkerställa att data lagrades och hämtades på ett förväntat sätt. Genom att testa centrala komponenter kunde buggar identifieras och åtgärdas.

I slutet av projektet fick ett antal lärare vid Chalmers testa applikationen och ge förslag på förbättringar. Utifrån denna återkoppling implementerades de förbättringar som ansågs vara högst prioriterade.

4

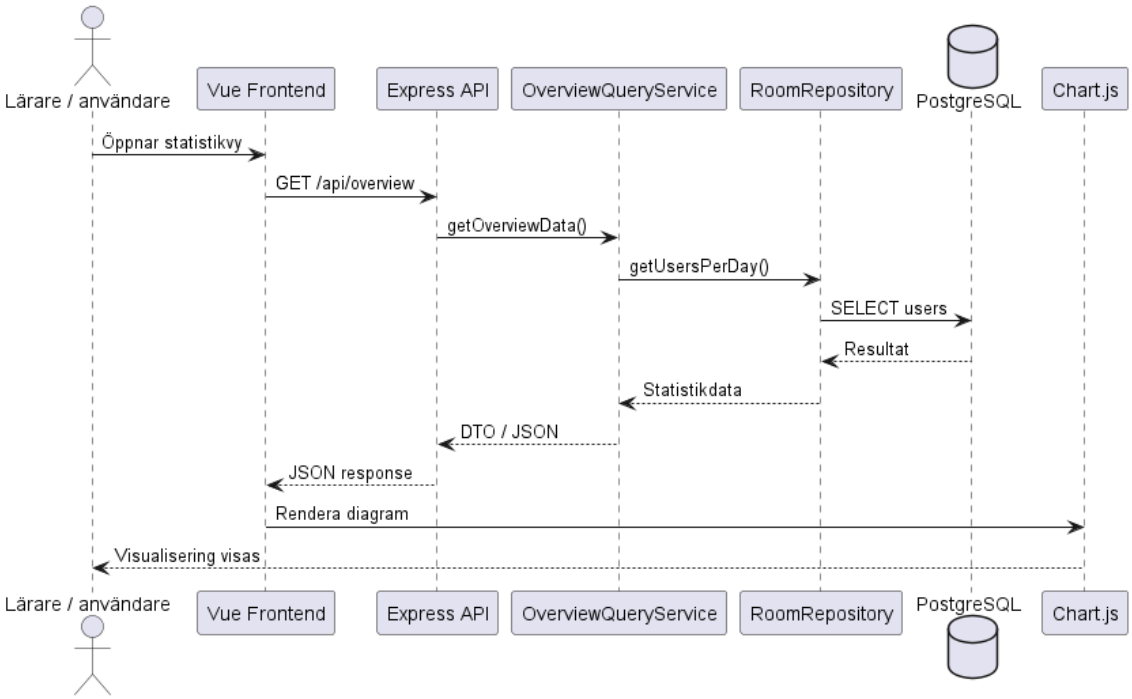
Systemdesign

Detta kapitel beskriver systemets övergripande arkitektur samt hur de olika delarna av applikationen samverkar. Backend-delen av systemet hade ansvar för inläsning, bearbetning och lagring av loggdata, medan frontend-delen hade ansvar för att hämta och visualisera denna data i ett webbaserat användargränssnitt.



Figur 4.1: Översikt över systemets arkitektur

Kommunikationen mellan frontend och backend utfördes via ett REST-baserat API implementerat med Express. Frontend skickade HTTP-förfrågningar till endpoints på backend för att hämta statistik. Exempel på endpoints som implementerades är `/api/overview`, som användes för att hämta övergripande statistik såsom antal användare och rum per dag, samt `/api/rooms/:name/:timestamp`, som användes för att hämta detaljerad information om ett specifikt rum. I backend hämtades sedan relevant data från databasen som sedan sammanställdes i form av dataöverföringsobjekt (DTO:er) och returnerade resultatet som JSON-data till frontend. Till slut visualiserades den mottagna datan i form av linje- och stapeldiagram.



Figur 4.2: Sekvensdiagram för kommunikation mellan frontend och backend

4.1 Backend

I backend användes designmönstret Data Transfer Object (DTO), vilket innebär att data som överförs mellan systemets olika lager struktureras i separata objekt. Detta bidrog till att tydliggöra ansvarsfördelningen mellan datamodeller, dataåtkomstlager och tjänstelager samt minskade beroenden mellan dessa delar av systemet.

De huvudsakliga komponenterna består av datamodeller, ett repository-lager för lagring och hantering av data, ett tjänstelager samt ett lager för att bearbeta loggdata. `Start.ts`- och `Index.ts`-klasserna ansvarar för att initiera och starta backend-sidan av applikationen. `Index.ts` startar servern och gör applikationen tillgänglig på en angiven port, medan `Start.ts` skapar en koppling till databasen och initierar nödvändiga tjänster och API-rutter för kommunikation med frontend.

4.1.1 Databas

För lagring av data användes PostgreSQL. Databasen består av tabeller för rum, frågor, svar och händelser såsom anslutningar till ett rum, och användes för att lagra den data som extraheras och tolkas av loggfilerna, så att applikationen på ett enkelt sätt skulle kunna hämta och sammanställa denna data för visualisering. Index skapades för att förbättra prestandan vid vanliga förfrågningar, och tidsstämpeln för den senast importerade loggdatan sparades och användes för att säkerställa att loggade händelser inte sparades i databasen mer än en gång var.

Även om det från början var tänkt att frågor och svar skulle sparas i databasen i realtid, så implementerades istället en lösning där applikationen bearbetar loggfilerna med jämna tidsintervall, baserat på en förbestämd variabel. Detta ansågs vara en tillräcklig lösning, eftersom den periodiska bearbetningen var tillräckligt snabb för att uppdateringarna fortfarande uppfattades som nästintill realtidsbaserade.

4.1.2 Datamodeller

Systemets datamodeller består huvudsakligen av klasserna `Room`, `RoomEvent` och `Response`, samt gränssnittet `Question` med tillhörande implementationer för olika frågetyper i Blurt. Modellerna användes för att strukturera den parsade loggdatan innan den sparades i databasen. På så sätt kunde information om rum, frågor, svar och användarhändelser hanteras på ett mer konsekvent och kontrollerat sätt.

Ett `Room` representerar ett rum i Blurt. Objektet innehåller bland annat rummets namn, tidsstämpeln för när rummet skapades, lärarens unika id, en lista över studenter som anslutit till rummet, en lista över frågor samt en lista över händelser kopplade till rummet.

```
1  type clientID = string;
2
3  export class Room {
4      public name: string;
5      public timestamp: string;
6      public students: clientID[];
7      public questions: Question[];
8      public teacherID: clientID;
9      public events: RoomEvent[];
10 }
```

Kodexempel 4.1: Room

En `Question` representerar en fråga i ett rum. Varje fråga innehåller en tidsstämpel som anger när frågan skapades, samt en lista över svaren på frågan. Eftersom Blurt stödjer flera olika typer av frågor användes ett gränssnitt, vilket gjorde att olika frågetyper kunde implementeras på olika sätt men fortfarande hanteras genom samma grundstruktur.

```
export interface Question {
    timestamp: string;
    responses: Response[];

    getType(): string;
}
```

Kodexempel 4.2: Question

En `Response` representerar ett svar från en användare. Varje svar innehåller en tidsstämpel, själva svarsvärdet och det tillfälliga id-numret för användaren som svarade. Detta gjorde det möjligt att koppla varje svar till både en specifik fråga och en specifik användare.

```
export class Response {
    public timestamp: string;
    public value: string;
    public id: string;
}
```

Kodexempel 4.3: Response

Klassen `RoomEvent` representerar en användarhändelse i ett rum. Varje händelse innehåller en tidsstämpel, användarens tillfälliga id-nummer samt en typ som angav om användaren anslöt till eller lämnade rummet. Dessa händelser användes för att kunna följa användaraktivitet över tid, exempelvis hur många studenter som var aktiva i rummet vid olika tidpunkter.

```
export class RoomEvent {
  public timestamp: string;
  public studentId: string;
  public type: "join" | "leave";
}
```

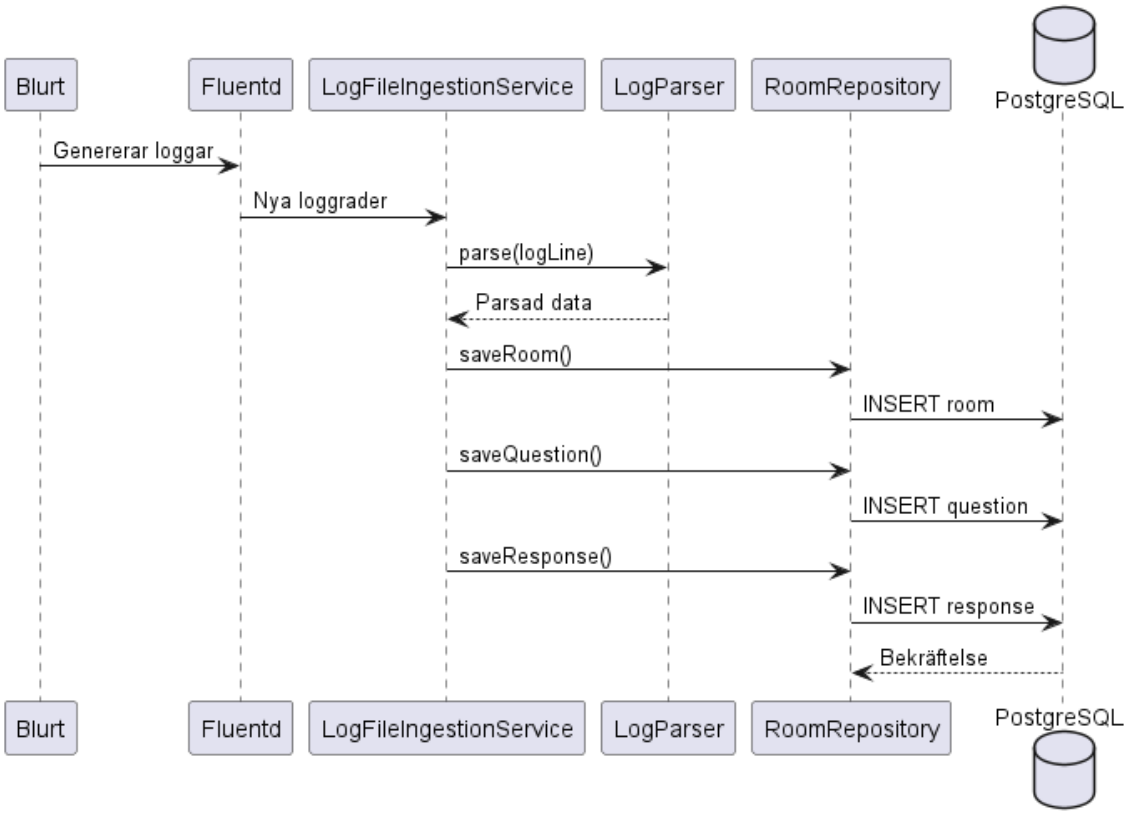
Kodexempel 4.4: RoomEvent

4.1.3 Dataåtkomstlager

Dataåtkomst hanterades genom klassen `Repository`, som fungerade som ett mellanlager mellan applikationen och databasen. Denna klass ansvarar för att hantera all interaktion med databasen. Detta innebär att övriga delar av systemet inte behöver hantera SQL-förfrågningar direkt, utan kan istället anropa metoder i `Repository`-klassen. Funktionaliteten i detta lager omfattar inte bara insättning av ny data, utan även hämtning av sammanställda datamängder som till exempel antal användare eller rum per dag.

4.1.4 Tjänstelager och hantering av loggdata

Inläsning och bearbetning av loggdata utförs genom ett antal klasser i tjänstelagret som hanterar parsning och ingestion. Processen inleds med att loggdata läses in rad för rad från loggfilerna. Varje rad bearbetas i flera steg, där data först extraheras och formateras på rätt sätt innan den omvandlas till interna datamodeller.



Figur 4.3: Sekvensdiagram för hantering av loggdata

Tolkningen av loggdata hanteras av klassen `LogParser`, som ansvarar för att identifiera relevanta händelser och skapa objekt som rum, frågor, svar och händelser utifrån denna data. Varje loggrad analyseras stegvis genom att först extrahera tidsstämpel, rumsnamn och typ av händelse. Därefter byggs ett internt händelseförlopp upp genom att markera de rum som för tillfället är aktiva, vilka frågor som är aktuella i respektive rum samt vilka användare som anslutit till eller lämnat ett rum.

Ett problem som uppstod under utvecklingsprocessen var att olika rum inte hade något globalt unikt identifieringsnummer i loggdatan. `LogParser`-klassen behövde därför själv skapa interna nycklar för att kunna särskilja olika rum med samma namn. Detta löstes genom att kombinera rummets namn med tidsstämpeln för när rummet skapades, så att varje rum hade en unik identifierare. På så sätt kunde flera olika rum hållas isär även om de hade samma namn.

4.1.4.1 Logghistorik och händelseförlopp

Eftersom loggdatan inte innehöll komplett information för varje typ av händelse behövde stora delar av händelseförloppet rekonstrueras utifrån ordningen på loggraderna. När en loggrad som exempelvis `create room` upptäcktes skapades ett nytt rum som markerades som aktivt. Senare händelser, såsom anslutningar, frånkopplingar samt skapande av frågor och svar behövde kopplas till rätt rum baserat på rummets namn och tidpunkten då rummet skapades. När en `close room`-händelse inträffade markerades rummet som inaktivt. Detta gjorde att systemet successivt kunde bygga upp en tidslinje över hur en Blurt-session hade använts och vilka användare och frågor som hörde till respektive rum.

4.1.4.2 Hantering av olika loggformat

Ett annat problem som uppstod var att loggdatan fanns i två olika format, nämligen det ursprungliga formatet och ett nyare format som inkapslar det gamla loggformatet i ett JSON-objekt. För att stödja det nya loggformatet utvecklades även klassen `LogLineUnwrapper`, som användes för att extrahera relevant information från det nyare loggformatet och formatera det precis som det gamla loggformatet. Detta var nödvändigt för att säkerställa att efterföljande hantering av loggdatan utfördes på ett korrekt sätt och att inga händelser tappades bort eller feltolkades.

4.1.4.3 Svarshantering

Ett centralt problem som uppstod var att avgöra vilken svarsdata som skulle visualiseras. Ett alternativ var att visa samtliga svar som hade registrerats för varje fråga. Ett annat alternativ var att endast visa det senaste svaret från varje enskild användare per fråga.

Detta blev särskilt relevant eftersom Blurt stödjer flera olika frågetyper, vilket innebär att svarsdatan kan se olika ut beroende på frågans format. För textfrågor uppstod ett särskilt problem om samtliga registrerade svar skulle visas. Om en användare skrev långsamt, ändrade sitt svar eller rättade stavfel kunde Blurt logga flera versioner av samma svar. I vissa fall kunde detta leda till att en enskild användare genererade ett stort antal svar på samma fråga, trots att dessa egentligen endast representerade justeringar av samma ursprungliga svar.

För att undvika att visualiseringen blev missvisande valdes därför en lösning där endast den senaste responsen från varje användare för en specifik fråga sparas. Om en användare ändrade sitt svar under en aktiv fråga uppdateras alltså den tidigare responsen istället för att flera separata svar skapas. Detta gjorde att visualiseringar och statistik inte påverkades av tillfälliga svar, samtidigt som mängden lagrad data reducerades betydligt.

4.1.4.4 Batchhantering

För att effektivisera hanteringen av stora mängder loggdata under en kort tid, vilket kan genereras till exempel om ett stort antal elever svarar på en fråga eller ansluter

och lämnar rum samtidigt, bearbetas flera loggrader i så kallade batches innan de sparas. Genom att behandla flera rader samtidigt minskar antalet databasoperationer, vilket ger betydliga prestandaförbättringar. För att ytterligare minska mängden onödiga operationer sparas även information om vilka rum som faktiskt förändrats sedan den senaste databasoperationen. Dessa objekt markeras som förändrade för att motsvarande rader i databasen ska kunna uppdateras.

4.1.4.5 Återuppspelning

Systemet stödjer även så kallad återuppspelning, vilket innebär att tidigare loggdata kan läsas in igen för att återskapa applikationens senaste tillstånd vid uppstart. Detta gör det möjligt att fortsätta bearbetning av data från en tidigare punkt utan att behöva läsa in all data från början. Vid uppstart så försöker systemet först läsa in loggar från den senaste dagen för att återskapa det interna händelseförloppet. Om den första raden i loggdatan för dagen inte är en `create room`-rad, eller om de efterföljande raderna inte har något rum att placeras i så försöker systemet återspela händelseförloppet från dagen innan, och så vidare. På så sätt kan ett händelseförlopp med relevant logghistorik byggas upp samtidigt som det är garanterat att inga händelser hör ihop med rum som skapats innan början på detta händelseförlopp. Detta gör också systemet betydligt snabbare vid uppstart.

4.1.4.6 Sammanställning av data

Klasserna OverviewQueryService och RoomQueryService fungerar som ett lager mellan API- och repository-lagren. OverviewQueryService ansvarar för att sammanställa övergripande information, såsom antal användare och rum per dag, medan RoomQueryService hanterar mer detaljerad data kopplad till enskilda rum, exempelvis svar per fråga. I RoomQueryService byggdes DTO:er upp, vilket innebar att data struktureras i ett format som är bättre anpassat för vidare användning, såsom visualisering i användargränssnittet.

4.1.5 API

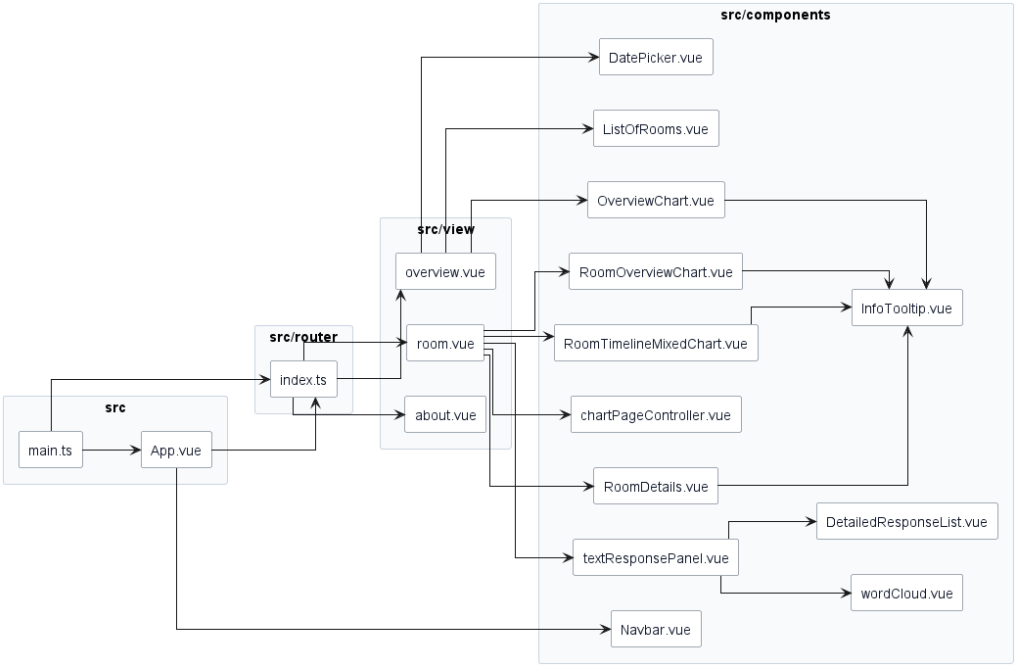
API-lagret implementerades med hjälp av ramverket Express och ansvarar för att låta frontend-delen av programmet kommunicera med backend via HTTP-endpoints. I detta lager kan inkommande förfrågningar tas emot och valideras för att sedan vidarebefordras till relevanta tjänster. API-lagret ansvarar även för grundläggande felhantering och validering, till exempel genom att kontrollera parametrar i förfrågningar.

4.2 Frontend

Systemets frontend-sida implementerades med ramverket Vue.js och Vue Router. Klienten ansvarar inte för den ursprungliga loggparsningen eller permanent datalagring, utan har främst som uppgift att hämta data från servern, bearbeta den till

vy-specifika datastrukturer och presentera resultatet genom ett interaktivt användargränssnitt.

Frontend-sidans systemdesign följde en komponentbaserad arkitektur där vyer, återanvändbara komponenter, visualiseringskonfiguration och datatransformering separerades i olika delar av kodbasen. Syftet med denna uppdelning var att minska kopplingen mellan olika delar av gränssnittet och göra systemet lättare att underhålla och vidareutveckla. I figuren nedan representerar de större blåmarkerade rutorna kataloger i projektets `src`-mapp, medan de mindre rutorna representerar filer eller Vue-komponenter. Pilarna visar huvudsakliga beroenden mellan filerna, exempelvis att en komponent importerar eller använder en annan komponent.



Figur 4.4: Översikt över frontendens fil- och komponentstruktur.

Klientens startpunkt finns i `main.ts`, där Vue-applikationen skapas, routern registreras och applikationen monteras i DOM:en. App.vue fungerar som applikationens övergripande komponent och renderar gemensamma gränssnittselement, såsom navigationsmenyn. Den aktuella sidan renderas därefter genom Vue Routers RouterView.

4.2.1 Routing och Sidstruktur

Klienten använder Vue Router med hash-baserad routing, och applikationen består av tre huvudsakliga routes, nämligen: `/overview`, `/room/:name/:timestamp` och `/about`. Startsidan omdirigerar till `/overview`, där användaren kan få en överblick över Blurt-aktivitet under en vald månad.

Från översiktssidan kan användaren navigera vidare till ett specifikt rum. Varje rum identifieras genom en kombination av namn och tidsstämpel, vilket gör det möjligt

att särskilja rum med samma namn som skapats vid olika tidspunkter. Utöver detta finns även en separat informationssida, `/about`, som beskriver applikationens syfte.

4.2.2 Komponentbaserad arkitektur

Frontend-delen av applikationen byggdes upp enligt en komponentbaserad arkitektur i Vue.js. Detta innebär att användargränssnittet delades upp i mindre, avgränsade komponenter, där varje komponent ansvarar för en specifik del av gränssnittet eller en viss typ av funktionalitet.

Exempel på komponenttyper i applikationen är vykomponenter, visualiseringskomponenter, navigationskomponenter, informationsrutor och kontrollkomponenter. Genom att dela upp funktionaliteten på detta sätt blev koden mer överskådlig och lättare att underhålla. Uppdelningen gjorde det även möjligt att återanvända komponenter, exempelvis informationsrutor och diagramkomponenter, på flera platser i applikationen.

4.2.3 Separation av ansvar

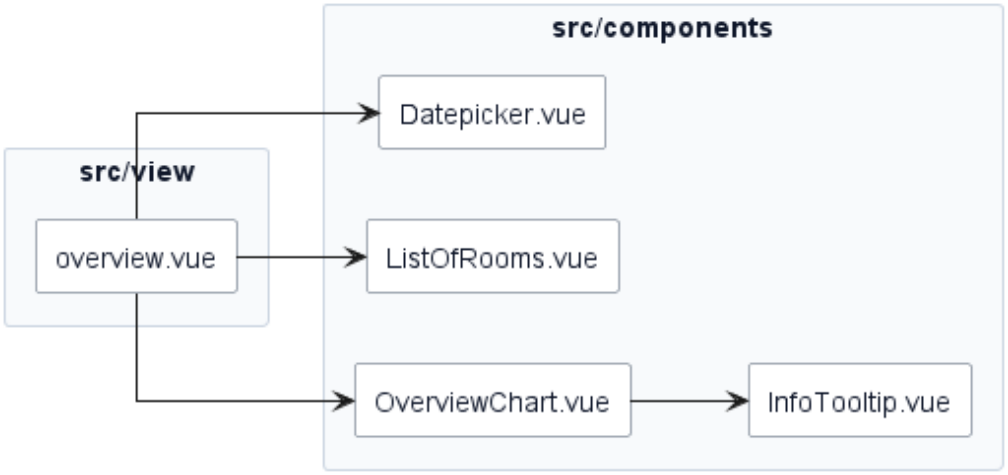
Ett centralt designval i frontend-delen var separation av ansvar, även kallat *Separation of Concerns*. Detta innebär att olika delar av systemet har tydligt avgränsade ansvarsområden. I applikationen användes Vue-komponenterna främst för presentation och användarinteraktion, medan bearbetning och omstrukturering av data huvudsakligen skedde i separata hjälpfunktioner.

Den data som hämtades från backend används därför inte direkt i visualiseringskomponenterna. I stället filtreras, aggregeras och transformeras datan till vy-specifika datastrukturer innan den skickas vidare till komponenterna. Dessa strukturer anpassas efter respektive visualisering, som till exempel svarsfördelning, tidslinjer och listor över text- och nummersvar.

Denna uppdelning minskade den direkta kopplingen mellan backend-sidans datastruktur och frontend-sidans presentationslogik. Det gjorde komponenterna enklare att förstå, testa och återanvända, samtidigt som förändringar i databehandlingen kunde göras utan att varje visualiseringskomponent behövde ändras.

4.2.4 Overview-vyn

Overview-vyn designades för att ge en översikt över Blurt-aktivitet under en vald månad. Vyn kan visa bland annat antal skapade rum och antal användare per dag. Komponenten fungerar som en container-komponent, vilket innebär att den ansvarar för att hämta data från API:et, lagra den lokalt i komponentens tillstånd och skicka relevant data vidare till underliggande presentationskomponenter.



Figur 4.5: Komponentstruktur i Overview-vyn

Overview-vyn bestod huvudsakligen av tre komponenter: `DatePicker`, `OverviewChart` och `ListOfRooms`. `DatePicker` låter användaren välja vilken månad som ska visas. När månaden ändras skickas ett event till Overview-vyn, som därefter hämtar ny data från servern och uppdaterar den data som skickas vidare till `OverviewChart` via props.

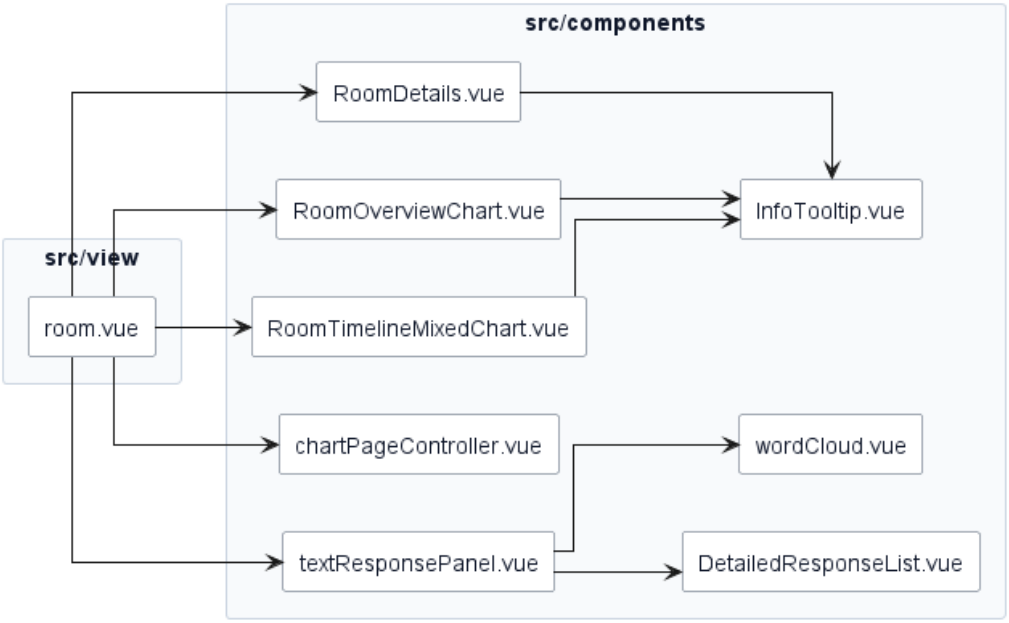
`OverviewChart` ansvarar för att visualisera den hämtade månadsdatan i ett diagram. När användaren interagerar med diagrammet, exempelvis genom att klicka på en specifik dag, skickas ett event tillbaka till Overview-vyn. Detta event används för att öppna `ListOfRooms`, som då kan visa de rum som skapades under den valda dagen.

Denna struktur gjorde att logiken för datahämtning, tillståndshantering och navigation hölls samlad i Overview-vyn, medan de underliggande komponenterna kunde utformas med fokus på presentation och användarinteraktion.

4.2.5 Room-vyn

Room-vyn ansvarar för att presentera och analysera data för ett enskilt Blurt-rum. Likt Overview fungerar komponenten som en container-komponent, men med ett mer detaljerat ansvar för datatransformering och visualisering av rumsdata.

Data som hämtas från servern var relativt nära den interna representationen av Blurt-loggarna. Detta minskade behovet av flera specialiserade API-anrop, men innebar samtidigt att viss bearbetning behövde utföras på frontend-sidan innan datan kunde visualiseras. Denna bearbetning skedde genom separata hjälpfunktioner i `roomData.ts`, vilket gjorde att datatransformeringen hölls separerad från komponentens presentationslogik.



Figur 4.6: Komponentstruktur i Room-vyn

Room-vyn består av flera barnkomponenter där varje komponent har ett tydligt avgränsat ansvar. Komponenterna `RoomOverviewChart` och `RoomTimelineMixedChart` tar emot bearbetad data via props och ansvarar för att rendera respektive diagram. När användaren interagerar med diagrammen skickas events tillbaka till Room-vyn. Dessa events används bland annat för att öppna `TextResponsePanel`, där mer detaljerad information om en vald fråga presenteras genom komponenterna `WordCloud` och `DetailedResponseList`.

4.2.6 RoomData

`roomData.ts` ansvarar för att omvandla den relativt råa rumsdatan som hämtas från backend till en vy-anpassad datastruktur som kan användas av klientens visualiseringar. `buildRoomViewData` är huvudfunktionen som tar emot ett `BackendRoom`-objekt samt en lista av `UserEvents` och returnerar ett `RoomViewData`-objekt.

```
export interface RoomViewData {
  room: {
    id: string
    name: string
    timestamp: string
  }

  summary: {
    totalUsers: number
    totalQuestions: number
    totalResponses: number
  }

  questions: QuestionViewData[]
  userTimeline: UserTimelinePoint[]
}
```

Kodexempel 4.5: RoomViewData

Funktionen tar emot information om ett rum, dess frågor och svar samt en lista över användarhändelser, såsom när studenter ansluter till eller lämnar rummet. Därefter filtreras ogiltiga eller tomma svar bort, och för varje fråga sparas endast det senaste giltiga svaret från varje student. Utifrån detta beräknas bland annat antal svar per svarstyp, totalt antal svar, antal frågor och hur många studenter som var aktiva i rummet vid varje frågetillfälle. För text- och nummersvar skapas även frekvensdata som används i responslistan och ordmolnet. På så sätt fungerar `roomData.ts` som ett mellanlager mellan backend-sidans datamodeller och frontend-sidans diagramkomponenter, där rå API-data struktureras och bearbetas till ett format som är direkt användbart för visualisering.

4.2.7 Visualiseringsarkitektur

Applikationens diagram implementerades med Chart.js. För att minska mängden upprepad kod och skapa en mer konsekvent visuell utformning separerades diagramkonfigurationen från komponenterna i egna filer:

- `mixedBarChartConfig.ts`
- `timelineMixedBarChartConfig.ts`
- `theme.ts`

Genom att separera diagramkonfigurationen från Vue-komponenterna kunde samma grundkonfiguration återanvändas i flera visualiseringar. Komponenterna behövde därmed endast skicka in relevanta parametrar som till exempel data, etiketter och diagramtyp, medan gemensamma inställningar för axlar, färger, tooltip och legend hanteras centralt.

Filen `theme.ts` används för att samla färg- och stilrelaterad logik. Detta bidrar till att diagrammen har ett enhetligt utseende och gjorde det enklare att ändra

applikationens visuella tema utan att behöva modifiera varje diagramkomponent separat.

Denna design gjorde visualiseringsdelen mer modulär och underlättade framtida vidareutveckling, exempelvis om nya diagramtyper eller ytterligare datavyer behöver läggas till i framtiden.

4.2.8 Variables.css

För att skapa ett konsekvent utseende genom hela applikationen användes en separat CSS-fil med globala CSS-variabler. I filen definierades bland annat färger, marginaler, padding och textstorlekar som sedan kunde återanvändas i flera komponenter.

Genom att samla dessa värden i `variables.css` blev det enklare att underhålla gränssnittets visuella design. Om exempelvis en färg eller ett avstånd behövde ändras kunde detta ändring utföras på en plats i koden, i stället för att samma värde skulle behöva ändras i flera olika komponenter. Detta bidrog till en mer enhetlig design och minskade risken för inkonsekvent styling.

5

Resultat

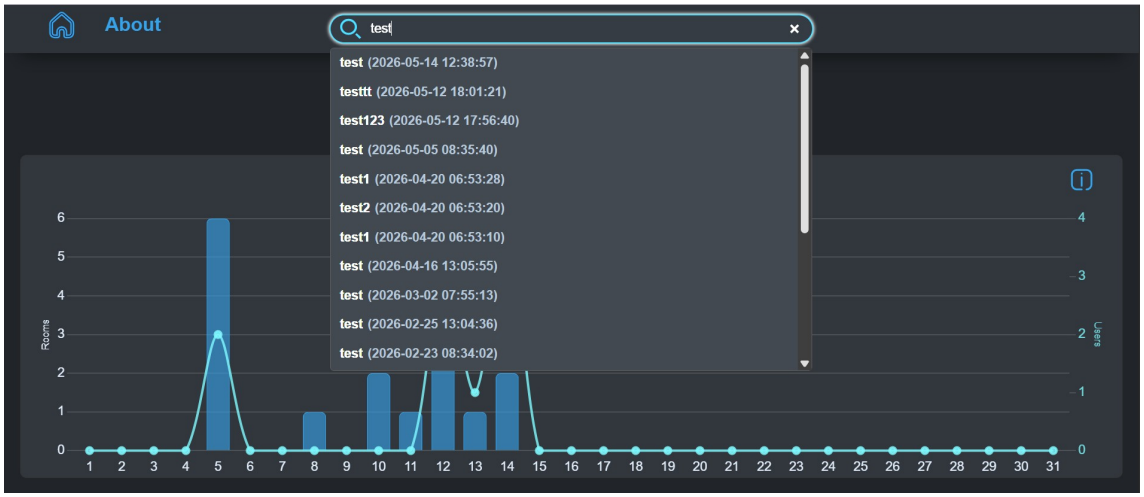
Arbetet resulterade i en fristående webbapplikation som kan läsa in, bearbeta och visualisera loggdata från Blurt. Applikationen gör det möjligt att analysera studenters aktivitet och engagemang genom olika typer av diagram. Resultatet är ett verktyg som förhoppningsvis kan ge lärare och kursansvariga bättre möjligheter att förstå hur Blurt används för att kunna dra slutsatser kring studenters deltagande.

5.1 Funktionalitet

Övergripande statistik presenteras på startsidan, såsom antal användare och rum per dag. På startsidan går det också att navigera mellan diagram för varje månad, och att klicka på staplarna i diagrammen för att direkt gå till ett rum. Det går också att söka efter ett specifikt rum.

Figur 5.1: Startsidan

Figur 5.2: Startsidan vid val av ett specifikt rum genom klick på stapeldiagrammet



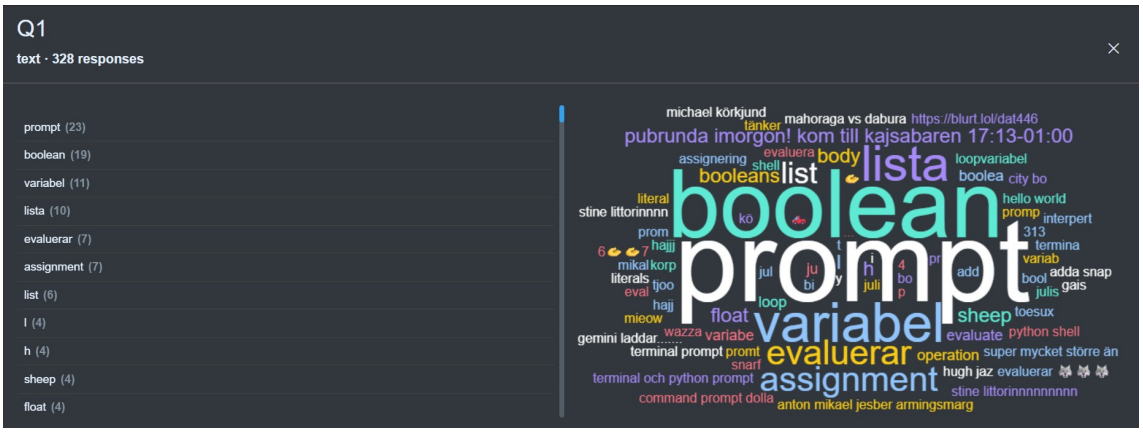
Figur 5.3: Startsidan vid sökning efter ett specifikt rum

Utöver detta kan mer detaljerad information visas för enskilda rum, som exempelvis antal användare under en föreläsning samt svarsfördelning för olika frågor. När det kommer till rumsstatistiken kan användaren växla mellan olika vyer, där x-axeln antingen representerar tid eller enskilda frågor. Användaren kan också filtrera bort delar av diagrammen för att tydligare se relevant data. Utöver det är det också möjligt att dynamiskt förstora och förminska diagrammen.

Figur 5.4: Sidan för ett specifikt rum där x-axeln representerar antal frågor

Figur 5.5: Sidan för ett specifikt rum där x-axeln representerar tid

Visualiseringarna implementerades i form av linjediagram för att visa antal användare samt stapeldiagram för antal rum och antal svar. För ja/nej-frågor och flervalsfrågor är stapeldiagrammen uppdelade i sektioner, vilket ger användaren en tydlig översikt över svarsfördelningen. Eftersom text- och nummerfrågor kan resultera i ett stort antal olika svar, har istället funktionalitet implementerats som gör det möjligt att klicka på dessa staplar för att visa en lista över svaren samt ett ordmoln.

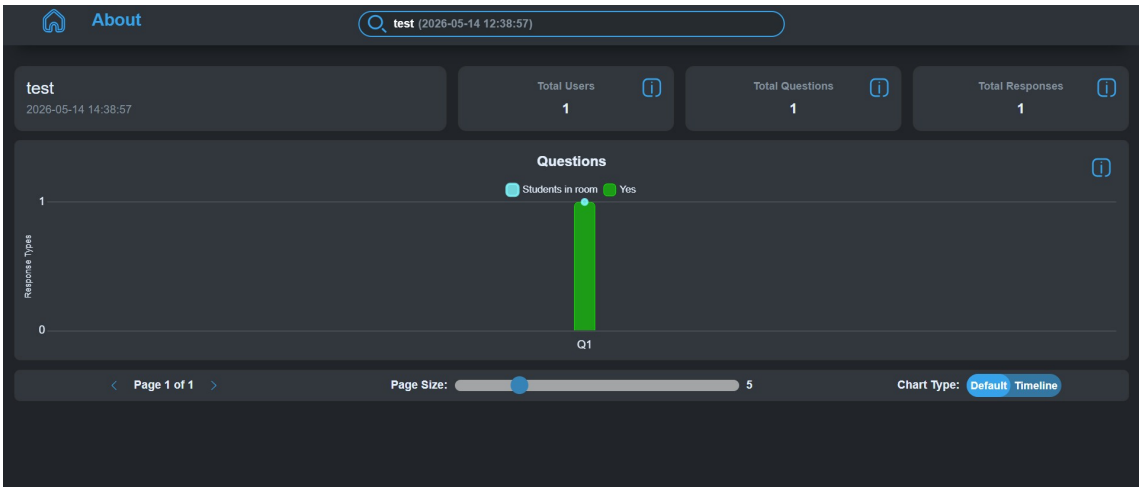


Figur 5.6: Ordmoln för en specifik fråga

Systemet testades med verklig loggdata, och kan hantera och visualisera data på ett korrekt och effektivt sätt. Nedan visas ett exempel på hur en specifik del av ett dokument med loggdata visualiseras, där en enstaka ja- eller nejfråga har skapats och fått ett svar av en student.

```
2026-05-14 12:38:57  [test] create room
2026-05-14 12:38:57  [test] uhqYqVjCECRNKnptAAAB teacher connect
2026-05-14 12:39:30  [test] CcRX1lTEA8HChIE7AAAD student connect
2026-05-14 12:50:21  [test] uhqYqVjCECRNKnptAAAB clear responses
2026-05-14 12:50:21  [test] uhqYqVjCECRNKnptAAAB set mode: yes-no-maybe
2026-05-14 12:50:53  [test] CcRX1lTEA8HChIE7AAAD respond: yes
```

Kodexempel 5.1: Exempel på loggdata



Figur 5.7: Exempel på visualiserad loggdata

5.2 Återkoppling från användare

Verktyget testades av lärare och kursansvariga på Chalmers som fick fylla i ett frågeformulär, och utav dessa var det två personer som gav återkoppling. I formuläret

fick personalen svara på frågor angående användbarhet och användarvänlighet, och komma med förslag på förbättringar. Formuläret bestod av följande frågor:

- Hur var din allmänna upplevelse av att använda applikationen?
- Var det något som var förvirrande eller svårt att förstå, och i så fall vad?
- Fanns det några buggar eller funktioner som inte fungerade som förväntat?
- Vilka funktioner skulle du vilja lägga till eller förbättra i applikationen?
- Upplevde du att användargränssnittet var intuitivt och enkelt att navigera?
- Var det tydligt vad varje knapp och funktion i applikationen användes till?
- Var statistiken och visualiseringarna i applikationen lätta att förstå?
- Finns det någon ytterligare feedback som du skulle vilja dela med dig av?

Generellt sett så ansåg personalen att användarupplevelsen var bra, och att applikationen är användbar när det kommer till att utvärdera föreläsningar i efterhand och att kunna se statistik för studentdeltagande. Det fanns dock förslag på vissa förbättringar.

I formuläret var det tydligt att knapparna för att ändra sidstorlek och sida orsakade viss förvirring. Eftersom det förmodligen inte finns någon anledning att ha dessa knappar i ett rum med ett litet antal frågor kan en möjlig lösning vara att helt enkelt inaktivera knapparna i rum där antalet frågor är tillräckligt litet för att få plats på en sida utan att diagrammet blir oöverskådligt.

Utöver det fanns det också ett behov av att, efter att användaren navigerat till ett specifikt rum, kunna gå tillbaka till den senast visade månaden på startsidan genom att klicka på tillbakapilen längst upp i webbläsaren. I nuläget navigeras användaren istället tillbaka till sidan för den nuvarande månaden.

Det fanns även ett behov av att göra navigeringen mellan olika rum med samma namn mer användarvänlig, exempelvis genom att behålla sökförslagen från den senaste sökningen varje gång användaren klickar i sökrutan.

Ett förslag som låg utanför projektets räckvidd var att visa innehållet i frågorna istället för endast svaren. För att implementera detta skulle programmet Blurt behöva modifieras, eftersom det i nuläget inte sparar information om frågorna i loggarna.

6

Slutsats

Projektet resulterade i en webbapplikation som innehåller de funktioner som bedömdes vara nödvändiga för att skapa ett användarvänligt och användbart verktyg.

6.1 Utvärdering av projektmål

Det huvudsakliga målet med projektet var att utveckla en fristående applikation som kunde läsa in, bearbeta och visualisera loggdata från Blurt på ett tydligt och lättöverskådligt sätt. Detta mål uppnåddes genom implementation av funktionalitet för hantering och lagring av loggdata samt visualisering av användarstatistik och svarsfördelning genom olika typer av diagram. Projektet avgränsades även till ett begränsat antal visualiseringar och stöd för moderna webbläsare. Mobilanpassning och mer avancerade analysfunktioner prioriterades bort på grund av projektets tidsram och omfattning.

Ett annat mål med projektet var att det skulle vara modulärt och lätt att vidareutveckla, vilket också är ett mål som anses ha uppnåts. Genom att följa Data Transfer Object (DTO)-strukturen på backend-sidan av programmet har en tydlig fördelning av ansvar mellan de olika klasserna gjorts. Frontend-sidan av programmet har byggts upp genom att placera olika komponenter i varandra enligt en komponentbaserad struktur, vilket gör gränssnittet modulärt och underlättar både vidareutveckling och återanvändning av kod.

6.2 Utvecklingsprocess och förändringar

Projektets specifikation förändrades också delvis under utvecklingens gång. Till exempel så var tanken från början att frågor och svar skulle sparas direkt i databasen i realtid, men detta ersattes senare av en lösning där loggfiler bearbetades periodiskt med jämna tidsintervall, vilket ansågs vara en tillräckligt bra lösning.

En aspekt av programmet som ändrades efter återkoppling från handledare var att de två ursprungliga graferna på startsidan över antal rum respektive antal användare kombinerades till en och samma graf med information om båda typerna av statistik, för att minska mängden visualiseringar på startsidan och göra gränssnittet mer överskådligt.

6.3 Begränsningar i loggdatan

En utmaning under projektets gång var att Blurts loggdata inte hade tillräckligt mycket information för att kunna visualisera den korrekt. För att hantera detta utvecklades funktionalitet för att strukturera datan och koppla relevanta frågor och svar till rätt rum innan de sparades i databasen.

6.4 Framtida arbete

Under projektets gång fanns det även förslag för möjliga vidareutvecklingar, som till exempel att implementera andra typer av diagram. Dessa prioriterades dock bort för att istället säkerställa att de mest centrala delarna av programmet fungerade korrekt.

Det finns alltså mycket förbättringspotential i framtiden för denna applikation. Av den återkoppling som gavs av lärare och kursansvariga framkom bland annat önskemål att kunna navigera tillbaka från ett rum till den senaste visade månaden på startsidan istället för den nuvarande månaden, att de senaste sökförlagen borde visas när användaren klickar på sökrutan och att knapparna för att ändra sida och sidstorlek borde avaktiveras i rum med ett begränsat antal frågor.

Utöver det kan applikationen i framtiden mobilanpassas och fler typer av visualiseringar kan också implementeras. Det finns också ett visst antal buggar, som till exempel att svartfiltreringsknapparna inte ör korrekt alfabetiserade eller kapitaliserade i vyn där x-axeln representerar tid. En annan bugg är att i rum som inte har några frågor så står det `Page 1 of 0` längst ner i vänstra hörnet. Dessa problem bör åtgärdas i framtida vidareutveckling av applikationen.

6.5 Avslutande reflektion

För att sammanfatta projektet så har det resulterat i ett fungerande verktyg som kan bidra till att ge bättre insikter i studenters engagemang och därmed skapa förutsättningar för att kontinuerligt förbättra undervisningen på Chalmers.

Källförteckning

[1] "Blurt README and Quick Start Guide", Accessed: May 14, 2026. [Online]. Available: https://github.com/johnjcamilleri/blurt/blob/main/README.md#quick-start

[2] "About Node.js", Accessed: May 14, 2026. [Online]. Available: https://nodejs.org/en/about

[3] "Vue.js Guide – Introduction", Accessed: May 14, 2026. [Online]. Available: https://vuejs.org/guide/introduction.html

[4] "Chart.js Documentation", Accessed: May 14, 2026. [Online]. Available: https://www.chartjs.org/docs/latest/

[5] "About PostgreSQL", Accessed: May 14, 2026. [Online]. Available: https://www.postgresql.org/about/

[6] "Node.js Express Framework", Accessed: May 14, 2026. [Online]. Available: https://www.w3schools.com/nodejs/nodejs_express.asp

[7] "Docker Overview", Accessed: May 14, 2026. [Online]. Available: https://docs.docker.com/get-started/docker-overview/

[8] "TypeScript Introduction", Accessed: May 14, 2026. [Online]. Available: https://www.w3schools.com/typescript/typescript_intro.php

[9] "Jest Documentation", Accessed: May 14, 2026. [Online]. Available: https://jestjs.io/docs/getting-started

[10] "What is a REST API?", Accessed: May 21, 2026. [Online]. Available: https://www.ibm.com/topics/rest-apis

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS