CHALMERS

Implementering av ett notifikationssystem och vidareutveckling av Chalmers On-boarding applikation

Utveckling av notifikationssystem och förbättrad kommunikation genom artificiell intelligens

Examensarbete inom högskoleingenjörsprogrammet Datateknik

ALI AZIZ
MHD SUHEIB SHAHIN

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2025
www.chalmers.se

Examensarbete 2025

Implementering av ett notifikationssystem och vidareutveckling av Chalmers Onboarding applikation

Utveckling av notifikationssystem och förbättrad kommunikation genom artificiell intelligens

ALI AZIZ
MHD SUHEIB SHAHIN

CHALMERS

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Göteborg 2025

Implementering av ett notifikationssystem och vidareutveckling av Chalmers Onboarding applikation
Utveckling av notifikationssystem och förbättrad kommunikation genom artificiell intelligens
ALI AZIZ, MHD SUHEIB SHAHIN,

Handledare: Sakib Sistek, Data- och Informationsteknik
Examinator: Nicholas Smallbone, Data- och Informationsteknik

Examensarbete 2025
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Skriven i LaTeX
Göteborg 2025

Implementering av ett notifikationssystem och vidareutveckling av Chalmers Onboarding applikation
Utveckling av notifikationssystem och förbättrad kommunikation genom artificiell intelligens.
Ali Aziz, Mhd Suheib Shahin
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Syftet med projektet var att utöka Chalmers Onboarding-applikationen med att implementera ett kommunikationssystem som består av följande funktioner; meddelandesystem, notifikationssystem, text-till-tal, artificiell intelligens baserad skrivassistent och förbättrade inmatningsmetoder. Projektet har lyckats att implementera meddelandesystem via mejl, text-till-tal med valbar röst och avatar, en artificiell intelligens skrivassistent för textformulering via Mistral AI från tjänsten Ollama, samt notiser för båda Android och iOS enheter via Firebase. Dessutom låg fokus på att förbättra användarvänligheten genom att simplifiera gränssnittet och relevanta valmöjligheter prioriterades för att underlätta användningen. Trotts vissa tekniska och ekonomiska begränsningar främst i form av omöjligheten att testa applikationen på iOS-enheter samt begränsning till kostnadsfria verktyg lyckades projektet leverera alla planerade funktioner. Slutligen diskuterades potentiella säkerhetsrisker och förbättringsmöjligheter bland annat molnbaserad artificiell intelligens lösningar och förbättrade urval av röster som genomförbara alternativ som framtida utvecklingsområden.

Nyckelord: Onboarding, Flutter, Dart, Android Studio, Text-till-tal, Notifikationer, AI, Plattformoberoende.

Abstract

The purpose of the project was to expand Chalmers Onboarding application by implementing a communication system consisting of the following features: a messaging system, notification system, text-to-speech, an artificial intelligence based writing assistant and improved input methods. The project successfully implemented email based messaging system, text-to-speech with selectable voices and avatars, an articiel intelligent writing assitant for text formulation using Mistral AI through Ollama service, as well as, push notifications for both Android and iOS devices through Firebase. In addition, emphasis was placed on improving user friendliness by simplifying the interface and relevant options were prioritized to decrease the difficulty of usage. Despite certain technical and financial limitations, primarily the inability to test the application on iOS devices and the constraint of using free of charge tools, the project successfully achieved all its planned features. Finally, the project discussed the potential security risks and areas for improvment, including cloud-based artificial intelligence solutions and an expansion of selectable voices.

Nyckelord: Onboarding, Flutter, Dart, Android Studio, Text-to-speech, Notifications, AI, Cross-platform.

Förord

Detta examensarbete har genomförts under våren 2025 inom Dataingenjörsprogrammet vid Chalmers tekniska högskola av Ali Aziz och Suheib Shahin. Vi vill rikta ett stort tack till alla som varit involverade, både nuvarande och tidigare i arbetets utveckling. Ett särskilt tack riktas till vår handledare vid Chalmers, Sakib Sistek för hans stöd och engagemang under hela projektets gång.

Ali Aziz och Suheib Shahin, Göteborg, Maj 2025

Beteckningar

Nedan följer en lista över akronymer som används i denna rapporten, presenterat i alfabetisk ordning

AI	Artificiell intelligens	(eng. Artificial Intelligence)
API	Applikationsprogrammeringsgränssnitt	(eng. Application Programming Interface)
AVD	Android virtuell enhet	(eng. Android Virtual Device)
FCM	*Firebase Cloud Messaging*	(Det saknas en motsvarighet på svenska)
HTTP	Hypertextöverföringsprotokoll	(eng. Hypertext Transfer Protocol)
IDE	Integrerad Utvecklingsmiljö	(eng Integrated Development Environment)
LLM	Stor språkmodell	(eng Large Language Model)

Figurer

Innehåll

1

Inledning

1.1 Bakgrund

Chalmers tekniska högskola är en internationell institution som grundades 1829 och har sedan dess etablerat sig som en av Sveriges främsta högskolor [1]. Detta har lett till att forskare och doktorander från hela världen söker sig till Chalmers för att arbeta och bli en del av institutionen. Denna mångfald av nyanställda med varierande bakgrund innebär dock en utmaning när det gäller integration och tillgång till viktig information.

För att underlätta denna process skapade Chalmers applikationen Onboarding som erbjuder smidigare tillgång till information och vägledning. En av de huvudsakliga utmaningar som applikationen ska lösa är att lokalisera kritisk information. I dagsläget är informationen utspridd över flera olika plattformar, vilket kan försvåra olika aspekter av integrationen bland annat navigation på campus eller tillgång till relevanta resurser.

Den tidigare versionen av applikationen lade en grund genom att implementera funktioner som en interaktiv karta, quiz och ett administrativt gränssnitt. Däremot identifierades flera utvecklingsmöjligheter, vilket gör att projektet kan vidareutvecklas för att utöka funktionaliteten och förbättra användarupplevelsen.

1.2 Mål

Målet med projektet är att vidareutveckla Chalmers Onboarding-applikation genom att bygga vidare på den tidigare versionen och samtidigt introducera ytterligare funktionalitet med fokus på att effektivisera kommunikationen mellan användare, öka tillgängligheten och förenkla hanteringen av applikationen för administratörer. Projektet har huvudsakligen fem mål:

- **Notifikationssystem**: Utveckla ett system som möjliggör för administratörer att skicka notiser till användare direkt i applikationen. Detta kan användas för att meddela om nyheter, viktiga uppdateringar och tidskänslig information på ett smidigt sätt.
- **Meddelandesystem via e-post**: Implementera ett Meddelandesystem som spelar rollen av backend-strukturen för notifikationssystemet. Det skulle även

vara möjligt för specifika användare, exempelvis avdelningschefer att skicka viktiga meddelanden till sina medarbetare baserat på deras avdelning och roll. Detta system är tänkt huvudsakligen att användas för tidskänslig och kritisk information, som ändrade mötesplatser, skiftscheman eller nödsituationer, inte som en informell chattfunktion.

- **Skrivassistent**: Integrera en artificiell intelligens baserad skrivassistent som stödjer nyanställda vid formulering av meddelanden. Assistenten kan ge förslag på grammatiska förbättringar samt klarspråk för att säkerställa att kommunikationen är professionell och lättförståelig och minska sannolikheten för masskommunikation som kan ske av orsaken att vissa användare som har andra modersmål kan ha svårigheter med att uttrycka sig på ett formellt sätt på engelska.

- **Text-till-tal funktionalitet**: Införa en artificiell intelligens driven text-till-tal-funktion som kan läsa upp innehållet i notifikationer och meddelanden direkt i appen. Detta förbättrar tillgängligheten, särskilt för användare med synnedsättning eller preferens för auditiv kommunikation.

- **Förbättringar till inmatningssätt i olika delar**: Införa ändringar till nuvarande sätt att mata in information som ökar användarvänlighet och förenklar processen. Trots att alla funktioner fungerar som de ska och nya info kan tilläggas utan behov av kodning förkunskaper så anser projektet att det finns utrymme för förbättringar som kan leda till en enklare och smidigare process för administratörer i flera olika delar av applikationen.

1.3 Syfte

Syftet med projektet är att vidareutveckla Chalmers Onboarding-applikation för att underlätta och stödja integrationen av nyanställda doktorander och forskare på Chalmers. I den tidigare versionen av applikationen identifierades flera utvecklingsområden som vid vidare utveckling skulle förbättra integrationen av doktorander och forskare på Chalmers betydligt. Vidareutvecklingen fokuserar på tre huvudområden.

- Notifikations- och meddelandesystem: Denna funktionalitet syftar till att effektivisera hur informationen sprids mellan doktorander och forskare. Genom att underlätta kommunikation kan applikationen bidra till förbättrad samarbete och interaktion inom forskningsmiljön på Chalmers.

- Tillgänglighet och stöd för användare med funktionsnedsättningar: Syftet är att säkerställa att alla användare har lika god tillgång till information. Många användare har svårigheter att ta del av information på grund av funktionsnedsättningar bland annat synnedsättning eller kognitiva nedsättningar. Applikationen kommer därför utökas med funktionalitet som text-till-tal och en artificiell intelligens baserad skrivassistent. Dessa funktioner kommer att stödja personer med synnedsättning, kognitiva utmaningar eller skrivsvårigheter, vilket förenklar åtkomst till information.

- Förbättrad användarvänlighet: Projektet kommer även vidareutveckla områden som identifierades att behöva finslipning för att öka användarvänligheten för båda administratörer och användare. Detta bidrar till mindre svårigheter att navigera och använda applikationen.

1.4 Avgränsningar

- Arbetet fokuserar på utvecklingen av själva appen utan tillägg av information och data i form av kartmarkörer, länkar, quiz frågor eller annat som kan förväntas vara befintliga vid användning av appen. Detta ska hanteras av Chalmers administratörer som senare kommer sköta appen.
- Projektet kommer inte heller ta hänsyn till Chalmers interna databas infrastruktur utan kommer endast utgå från att en godtycklig PostgreSQL server ska kunna anslutas.
- Implementationen av övrig funktionalitet som skrivassistent och text-till-tal kommer att baseras på befintliga verktyg och kodbibliotek. Projektet innefattar därmed inte utvecklingen av sådana lösning från grunden utan utgår från etablerade verktyg.
- Testning kommer utgå preliminärt utifrån Android emulatorn som inkluderas i den utvalda integrerad utvecklingsmiljö. Detta val motiveras av att iOS kräver tillgång till Mac eller liknande enheter för att kunna simulera applikationen.

2

Teknisk Bakgrund

Detta kapitel presenterar teknisk bakgrund kring de olika momenten som genomfördes under projektets gång. Detta kommer inkludera begrepp och teknologier som erbjuder en grundläggande förståelse för de olika verktyg och tekniker som används under projektets utveckling.

2.1 Versionshantering

Versionshantering handlar om att organisera, spåra och kontrollera flertal versioner av samma projekt. Detta innebär varenda ändring som orsakas av ett tillägg, modifikation eller borttagning av filer inom ett projekt spåras. Dessa ändringar sparas och öppnar möjligheten för att återställa filer till sin tidigare version. Denna teknik erbjuder förmågan att parallellt utveckla ett projekt inom olika områden med reducerad risk att skriva över andras arbete. För att detta ska drivas effektivt, versionshantering används i stor utsträckning i kombination med verktygen Versionshantering system. Detta system assisterar användarna för att enkelt navigera genom de olika versionerna och erbjuder ett simpelt sätt att jämföra olika versioner av projektet för att återställa till tidigare versioner vid behov. Varje version omfattar information om de genomförda ändringarna, en lista över alla ändrade filer, samt vid grupparbete specificeras vilken individ som har modifierat koden och vid vilken tidpunkt [2].

2.2 Databas

En databas är ett datorbaserad system för att underhålla data och gör den tillgänglig vid behov. Informationen som lagras i sådana system är ursprungligen rådata som mjukvarusystemet använder för att producera användbar information. Dessa data bearbetas och assimileras för att förmedla användaren av relevant information. Databassystem har som huvudsyfte med att bevara information och administrera över det på ett säkert och organiserat sätt utifrån fördefinierade egenskaper och relationer. Systemet används ofta i kombination med en uppsättning program för att möjliggöra hantering av databasen. Dessa system kan vara konstruerade utifrån ett objektorienterat, relationellt eller hierarkiskt syn [3]. Projektet använder ett relationsdatabassystem som använder en relationsmodell för att organisera data. Denna typ av modell lagrar data i tabeller där varje rad och kolumn representerar

ett attribut.

2.3 Stor språkmodell

En stor språkmodell (Large language model, LLM) är en modell designad för att tolka och bearbeta mänsklig skriven text samt att generera skrift som kan uppfattas av människor [4].

Träningsprocessen av språkmodellen uppdelas i två primära kategorier: finjustering och prompting. Det förstnämnda tränar språkmodellen på en stor mängd av oklassificerad data som är specificerad för textinmatning såsom mänsklig text eller kod. Efter denna process går språkmodellen genom flera finjusteringsstadier för att finslippa modellen med hjälp av ett klassificerat dataset. Den sistnämnda träningsprocessen involverar användning av promptar. Dessa promptar inkluderar språkmeningar som är ofullständiga och som har tomrum för språkmodellen att fylla in [5].

2.4 Applikationsprogrammeringsgränssnitt

Applikationsprogrammeringsgränssnitt (Application programming interface, API) är gränssnitt som möjliggör för flera sorters applikationer att kommunicera med varandra genom utbyte av information. API sätter reglerna för hur förfrågningar och respons ska utformas. Detta möjliggör för en applikation att tillämpa specifika funktionalitet från en annan applikation utan att behöva kunna de interna implementationerna. Detta innebär att auktoriserad användare kan förfråga om information och få svar från de ursprungliga applikationen.

Original applikationen kan modifiera signaturen av förfrågningarna och kontrollera åtkomsten till funktionalitet. Det finns två standardiserade typer av API; kodbibliotek och webbtjänster. Kodbibliotek bygger på att strukturera upp serier av förutbestämda funktionsanrop. Dessa anrop är konstruerade för att kunna skickas mellan flera applikationer och kan variera drastiskt i komplexitet. Webbtjänster är traditionellt designera omkring att skicka förfrågningarna över HTTP-kanaler [6].

2.5 Avatar

`Lottie` är ett kodbibliotek som tillåter utvecklare att återge Adobe After Effects-animationer direkt i Flutter-applikationer genom att exportera animationerna som JSON-filer. Lottie tillåter smidig integration över flera plattformar, inklusive Android, iOS och Windows samt erbjuder kontroll över samtliga egenskaper hos animationerna på ett enkelt sätt [7].

2.6 Text-till-tal

`Flutter_tts` är ett inbyggd Flutter kodbibliotek som erbjuder grundläggande kostnadsfri text-till-tal funktionalitet över samtliga plattformar bland annat Android,

iOS, webb, Windows och macOS. Bibliotek innehåller en samling av flera röster på olika språk samt tillåter kontroll över röstens egenskaper såsom tonhöjd, volym och talhastighet [8].

2.7 Firebase

`Firebase_core` och `firebase_messaging` är kodbibliotek som möjliggör användning av *Firebase*-funktionalitet [9]. Det förstnämnda är essentiellt för att använda funktioner som är associerade med Firebase. Firebase_core måste initieras innan några ytterligare anrop görs till relaterade kodbibliotek. Det sistnämnda används för att kommunicera med *Firebase Cloud Messaging* (FCM) [10]. Den har för uppgift att skicka och ta emot meddelanden i form av pushnotiser över flera plattformar såsom *Android*, *iOS* och webbplattformar.

2.8 Google

`Googleapi` ger tillgång till genererade *Dart* kodbibliotek som är relaterade till Googles egna tjänster. Genom dessa bibliotek öppnar möjligheten att interagera med olika Google tjänster i Flutter-applikation. Dessa kan vara bland annat *Google Drive API* , `Google Docs API` eller `Custom Search API` [11].

För att autentisera och auktorisera åtkomst till dessa tjänster används kodbiblioteket `googleapis_auth`. Den har som uppgift att hantera olika funktionalitet relaterade till OAuth2. Kodbiblioteket ger möjligheten att smidigt hämta en auktoriserad HTTP-Klient och automatiskt förnya OAuth2-uppgifter [12].

2.9 Ganttschema

Gantt-schema är ett verktyg för att analysera olika tidsintervall inom ett projekts tidslinje ochc bedöma om en uppgift kan genomföras. Den totala tiden för att utföra en arbetsuppgifter uppdelas i mindre tidsintervall där varje uppgift planeras och tilldelas en specifik tidsperiod för genomförandet. Detta innebär att projekt i sin helhet planeras och struktureras för att kunna genomföras inom en bestämd tidsperiod. Detta verktygen ofta används för att grafiskt presentera hur olika moment kommer se ut under tidsperioden för projektet [13].

2.10 Integrerad Utvecklingsmiljö

Integrerad utvecklingsmiljö (Integrated Development Environment, IDE) är ett program som består av flera avancerade verktyg såsom en kodredigerare, en kompilator och en debugger. Dessa verktyg är ofta utvecklade separat och därefter integreras i en IDE. En traditionell IDE ger möjligheten att utveckla program på flera programmeringsspråk via användning av plugins och underlättar för mjukvarautvecklare att

på ett effektivare sätt skriva, testa och debugga kod [14]. I detta projekt användes Android Studio som den primära integrerade utvecklingsmiljön för att utveckla program för både Android- och iOS-plattformarna. Android Studio inkluderar en emulator för Android-mobiler via Android Virtual Device (AVD) vilket ger möjligheten för körning av applikationen i en simulerad miljö [15].

2.11 Verktyg

2.11.1 Flutter

Flutter är ett open-source användargränssnitt ramverk. Designat för att tillåta utveckling av program på en gemensam kodbas samtidigt som det ger möjligheten att kunna köra på flertal plattformar bland annat: webbaserade, mobiltelefoner och datorsystem. En kodbas är en samling av olika kodfiler för att skapa mjukvaruprogram, hemsidor och applikationer. Varje format av mjukvara som nämndes tidigare har olika krav för att fungera på korrekt sätt. Flutter i sin grund är designat för att hindra dessa problem. Detta erbjuder en smidigare utveckling av mjukvara program på flera plattformar bland annat Android, iOS och Windows [16]. Flutter är konstruerat för att fungera på Dart-plattformen vilket innebär att den använder dart-programmeringsspråket som är ett objektorienterat språk samt har stöd för både statisk typning och asynkron programmering.

2.11.2 Kotlin

Kotlin är ett modernt programmeringsspråk utvecklat av JetBrains och introducerad vid 2011. Språket är designat för att vara koncist och säkert samt kompatibelt med Java med stöd för båda objektorienterad och funktionell programmering. År 2017 blev Kotlin erkänt av Google som ett officiellt språk för Android-utveckling. Utöver Android används Kotlin också för serverbaserad utveckling, webbapplikationer och plattformsoberoende mobilutveckling [17].

2.11.3 Firebase

Firebase är en omfattande plattform utvecklad av Google för utveckling av mobil- och webbapplikationer. Den erbjuder en mängd verktyg och tjänster som är utformade för att hjälpa utvecklare att bygga och förbättra sina appar på ett effektivt sätt. I detta projekt användes Firebase för att implementera pushnotiser för både Android och iOS [9]. Firebase Använder tjänsten `Firebase Cloud Messaging` (FCM) för att skicka meddelanden i form av pushnotiser till Android mobiler samt den motsvarande tjänsten `Apple Push Notification Service` (APNs) för iOS mobiler.

2.11.4 Ollama

Ollama är en open-source platform som erbjuder möjligheten att köra LLM:er lokalt på en personlig dator. Lokala modeller körs frånkopplat från internet och eliminerar

behovet av molnbaserade API:er eller betalda prenumerationer, vilket minskar beroende av externa tjänster [18]. Ollama är kompatibel med olika operativa system som Windows, Mac och Linux samt stöder flera modeller för lokal användning som bland annat, Mistral, Llama och DeepSeek.

2.11.5 Mistral AI

Mistral AI är en fransk startup inom artificiell intelligens som grundades i april 2023 av forskarna Arthur Mensch, Guillaume Lample och Timothée Lacroix [19]. Mistral utvecklar huvudsakligen open-sources LLM:er som kan integreras med andra tjänster och applikationer.

3

Metod

Detta kapitel redogör för de metoder som har använts för att navigera genom problem, hantera hinder och lösa utmaningar. Detta kommer bland annat att omfatta arbetssätt, datainsamlingsmetoder och problemlösning.

3.1 Arbetsprocess

Projektets arbetsinsats baserades på en planeringsrapport som skrevs under den första perioden. Huvudfokus i planeringsrapport var att formulera projektets inledning, metod och tidsplan. Gruppmedlemmar utgick ifrån Gantt-schemat som konstruerades i tidsplan avsnittet av planeringsrapporten. Den omfattade alla moment som projektet kunde genomföra under en termin och specificerade tidsintervallet för varje moment. Tidsplaneringen grundades på flera aspekter, bland annat hur lång tid det beräknas att alla moment kräver utifrån kollektiva erfarenheter, tillgången på resurser som är relativt anpassat till projektets syfte, samt hur mycket tid som projektet förväntas att satsa veckovis.

3.2 Datainsamling

Under projektets gång har medlemmarna samlat olika typer av information för att hjälpa med rapportskrivning samt aktuella kodning och applikation planering. Tack vare medlemmarnas tidigare erfarenhet med projektet samt Flutters extensiva dokumentation och resurser som spelade rollen av den primära källan för teknisk information som har varit nödvändig för utvecklingsprocessen. Utöver det har även andra resurser såsom diskussionstrådar på Stack Overflow, youtube filmer och publika repositories på Github använts som referensmaterial för att effektivisera utvecklingsprocessen. Dessa mer tekniska resurser uppfyller inte kraven som vetenskapliga källor och kommer dessutom inte citeras i rapporten. Istället jämfördes informationen med officiella dokumentation och användes därefter som en referens eller riktlinje vid implementation av nya funktioner.

Det fanns till och med behov för teoretiska källor som har använts för att stärka den akademiska grunden i rapporten och säkerställa att projektet vilar på en solid vetenskaplig bas samt att ge en grundlig förklaring till samtliga relevanta verktyg och koncept som nämndes i rapporten. Källorna består huvudsakligen av vetenskapliga artiklar hämtade från Chalmers bibliotek, samt andra vetenskapliga publikation

plattformar hittad via Google-scholar. Dessa externa källor har tydligt redovisats för att skilja dem från projektets egna slutsatser och implementationer.

Denna metod för informationsinsamling har säkerställt att både den teoretiska och tekniska grunden är väl underbyggd, vilket bidrar välstrukturerade och begriplig process.

3.3 Insamling av verktyg

Under projektets gång identifierades flera moment som krävde verktyg som var utanför gruppmedlemmars erfarenhet. Detta innebar att verktyg kontinuerligt inhämtades från externa resurser. Projektet primärt utgick från Flutter och Dart officiella paketförråd [20] för att samla ihop de olika verktyg. Under vissa moment upptäcktes ytterligare verktyg med potential att tillämpas för att konstruera applikationen. Denna moment var tidskrävande och involverade arbete för att kontrollera och säkerställa att de olika kodbiblioteken inte medförde problem vid installation. Dessa problem kunde involvera inkompatibla versioner av Kotlin, SDK versioner och IOS versioner. Projektet även undersökte om äldre versioner av paketet var kompatibla med den nuvarande versionen av applikationen. Ett annat problem kunde vara i form av olika verktyg som krävde varierande API-nycklar för att kunna användas. Detta medförde ofta att vissa verktyg involverade betalningsplaner, vilket låg utanför projektets tillgängliga resurser.

3.4 Användning av AI

Under projektets gång användes AI med syfte att förbättra rapporten. Detta inkluderade att hitta synonymer till ord för att formalisera eller hitta motsvarande begrepp på svenska, samt att fixa till grammatiska fel och under vissa instanser att korrigera dem. Användningen av AI i rapportskrivandet avgränsat till att det tidigare nämnda.

4

Genomförande

I följande kapitel kommer genomförande av projektets olika moment att diskuteras. Detta kommer att inkludera en beskrivning av arbetsgången samt motivationen bakom de olika besluten som har fattats. Denna sektion kommer att inledas i en kronologisk ordning baserat på hur projektets huvudmål har genomförts.

4.1 Text-till-tal

Utvecklingen av text-till-tal bygger på inkapslingen av data och funktionalitet som relaterade till det utvalda kodbiblioteket. Detta inkluderade deklarationen av *FlutterTts*, fast även relaterade variabler såsom volym, tonhöjd, talhastighet. Detta lade grunden för utvecklingen av interna implementationerna för att spela rösten till en specificerad text och stoppa den. Den första aspekten som behövdes ta hänsyn till vid implementationen av den förstnämnda funktionen var att välja mellan endast röster som tillhör det utvalda språket. Projektet löste det problemet genom att implementera en ny funktion som var ansvarig för att hämta alla möjliga röster och filtrerar ut efter det utvalda språket. Resultat sparades i en lista av röster. Den första av dessa röster lagrades och utvaldes för att köra som standardrösten vid initiationen av objektet. De resterande parametrarna initieras med standardvärden varje gång objektet skapas.

Ett tidigt problem som uppstod var att pausfunktionalitet för rösten originellt endast utvecklades för Android-plattformen. Detta ledde till ett flertal problem under utvecklingsprocessen, vilket resulterade i oförväntade konsekvenser där rösten slutade spela utan någon tydlig anledning. För att bemöta detta problem utvecklade projektet en intern implementation av pausfunktionaliteten som uppdelades i två funktioner. Den första var ansvarig för att spela rösten från stadiet den stoppades. Implementation utgick ifrån att kontinuerligt spara aktuella positionen i röstuppspelningen. Vid anrop till andra funktionen förkortades den aktuella texten till punkten där den tidigare avslutades och i sin tur startar rösten med den förkortade texten.

4.2 Avatar

Under denna process utvaldes att 3D-animationer skulle först införas för sin utmärkta potential. Det visade sig vara mindre lämpligt för projektets syfte då den utvalda kodbiblioteket var resurs- och tidskrävande. Valet av 2D-avataren baserades på kodbiblioteket *Lottie*. Den inkluderade flera animationer av båda 2D- och 3D-dimensioner. Genom att lista ut standardkaraktär av avataren blev det enkelt identifiera och inkludera grundläggande ansiktsdrag, proportionell kroppsstorlek och utan några kontroversiella aspekter. Implementation av biblioteket krävde minimal insats då det endast behövde ett gränssnittselement för animationen och en *Animationcontroller* för att hantera de olika aspekterna såsom start, stop och varaktighet. Något som uppmärksammades under arbetsgång var att vissa animationer inkluderade oönskade komponenter såsom en icke-transparent bakgrundsfärg. Detta kunde dock lösas effektivt och smidigt genom Lotties webb-baserade animationsverktyg.

4.3 Konstruktion av Text-till-tal och avatar sidan

Konstruktionen av *TTS & Avatar* sidan har i huvudsyfte med att ha en lokaliserad plats för att modifiera egenskaper tillhörande text-till-tal och avatar. Det krävde att skapa en ny sida som är kopplat till den existerande profilsidan. Den primära tanken är att alla inloggade användare oavsett kontobehörighet ska ha tillgång till den.

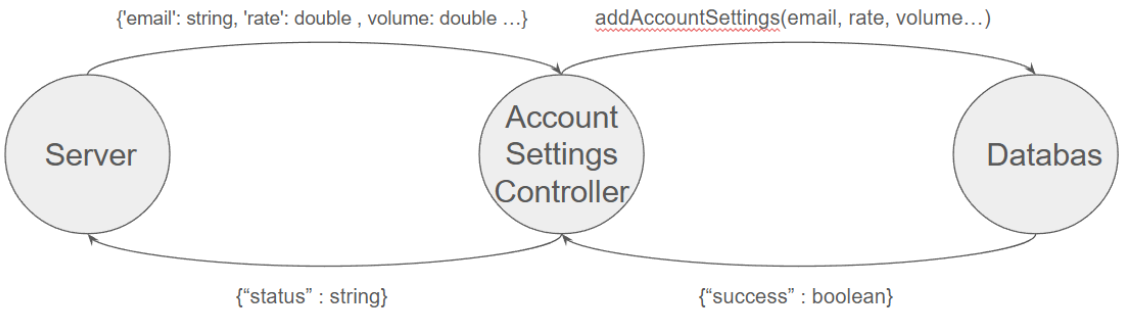
För text-till-tal insåg projektet att det finns aspekter som är viktigare att ha tillgång till än andra. Dessa egenskaper var rösten, volymen, tonhöjden och talhastighet. Det fanns några designval över hur användaren skulle interagera med dessa parametrar. En populär implementering för att välja mellan olika röst anses vara en rullgardinsmeny, vilket användes på flertal av sidorna som projektet inspekterade. Här fattade även projektet ett beslut vid varje instans av röst ändring avslutas rösten och börjar om. Vidare utvecklades gränssnittet för de resterande tre parametrarna. En lämplig design var att skjutreglage för varje individuellt parameter. För att testa dessa ändringarna implementerades en knapp som startar och pausar rösten på en fördefinierat text.

Vid integration av avatar funktionalitet var det viktig att kunna smidigt ändra mellan olika fördefinierade avatarer som är fördefinierade i en resurs mappen. Projektet utgick från sin implementation av röst rullgardinsmeny för att replikera det vid avatars valet. Detta medförde ett ytterligare problem med hämtning av avatars alternativen. Detta krävde en ny funktion som hämtar varje individuellt avatars filväg från tidigare nämnda mappen för att visa upp det i menyn.

Något som var viktigt vid denna stadiet är att spara de olika förändringar och associera det med ett specifikt konto. Denna implementation krävde en expansion av båda klientsidan och serversidan. Från klientsidan krävdes det ett sätt för användaren att interagera med lagring processen. Projekt implementerade det genom att utveckla en knapp ansvarig för att skicka till serversidan när användaren är nöjd med ändringarna.

Knapptrycket skickar en HTTP-förfrågan med e-postadressen, volym, tonhöjd, talhastighet och avatar filväg. Projektet beslutade att exkludera röstvalet från datainnehållet av HTTP-förfrågan. Detta baserades på att tillgängliga röster varierar mellan olika enheter, vilket gör att ett specifikt röstval inte alltid är möjlig för alla klienter.

Serversidan utökades för att möjliggöra mottagning och hantering av inkommande HTTP-förfrågningar riktat mot en specifik sökväg. Det sistnämnda processen sker i enlighet med Figur 4.1.



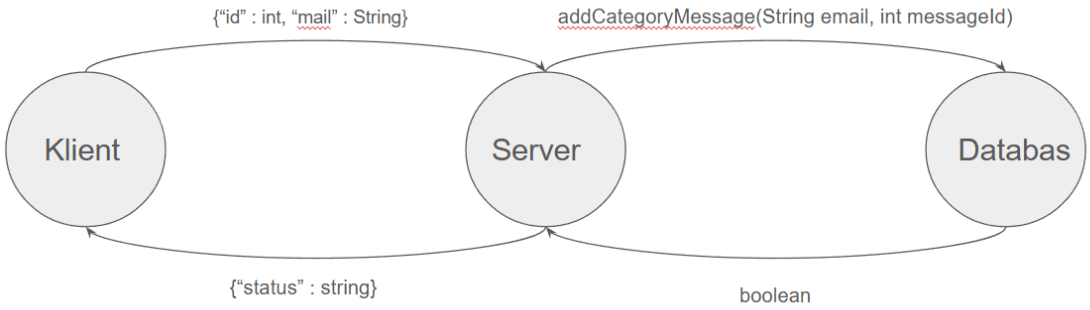
Figur 4.1: Tillägg av konto inställningar

4.4 Meddelandesystem

Implementeringen av meddelandesystem utgick från intern design. Den baserades på vad projektet kände till för möjliga verktyg samt hur traditionella tjänster är strukturerade. Detta ledde till följande preliminära design, se Figur 4.3. Projektet konstruerade ett tomt skelett som baserades på den tidigare figuren. Varje sektion ska inkludera meddelande utsedd för den aktuella kategorin *inbox*, *sent* och *favorite*. Vilket uppnåddes genom att implementera funktionalitet för att generera en listview med respektive meddelande för varje kategori. Detta gränssnitt möjliggör för användare att navigera genom innehållet med hjälp av rullningsfält. Visning av meddelande inkluderar funktionalitet att vid ett tryck av ett meddelande så visas den utvalda meddelandet i ett större format.

För att implementera funktionen att skicka e-postadress från klientsidan till serversidan skapades en knapp längst ner till höger för att navigera till sändningssidan. Gränssnittet inkluderade tre textfält: e-postadress, titel och beskrivningen. Vilket användes för att skicka en HTTP-förfråga till serversidan med all nödvändig information. Detta adderade ett meddelande i databasen under tabellen *messages*. Serversidan returnerade ett id av det nya meddelandet till klientsidan. Vilket användes i sin tur av klientsida för att skicka två nya HTTP-förfrågningar.

Nästa centrala process var att associera ett meddelande med sändarens e-postadress för att identifiera olika användare. Detta inkluderade metoder från klientsidan för att skicka sändarens e-postadress och meddelandets id för respektive kategori som lagras i databasen, vilket sker i enlighet med Figur 4.2.



Figur 4.2: Tillägg av meddelande under respektive kategori

De tidigare ändringarna krävde att nuvarande visning gränssnittet hämtar meddelande som associeras med kontots e-postadress. Detta ledda till bildandet av en metod som var ansvarig för att skicka en HTTP-förfråga för att hämta alla relevanta meddelande från respektive tabell. Dessa meddelande sparades senare separat i sin respektive lista. Vilket användes för att generera en listview med meddelande för varje kategori.

Projektet utökade visning sidan med ett ytterligare gränssnittselement: en favorit knapp. Funktionalitet av knappen liknar implementationen av tidigare nämnde sändning metod, vilket tar sändarens e-postadress och id relaterade till meddelande som parametrar. Sedan skickar det till serversidan genom en HTTP-förfrågan som inkluderar förenämnda parametrarna och lagras i databasen. För att öka användarvänligheten utökas favoritknappen med en flagga för att ändra färgen på från vit till gul vid aktivering.

Figur 4.3: Skiss för meddelande sidan

4.5 Notifikationssystem

Tidigt i projektet upptäcktes det att iOS-enheter hade annorlunda krav över hur HTTP-förfrågningar kan skickas till en applikation när den inte aktivt används. Projektet hittade ett populärt alternativ som använder en extern webbtjänst, Firebase cloud messaging och utgick från det för att skicka notifikationer till alla kopplade enheter.

Integrationen av Firebase med Flutter applikationen började med att konfigurera ett nytt projekt i Firebase console. Projektet valde det alternativa sättet med FlutterFire CLI, vilket rekommenderas när ett flutter projekt ska konfigureras över flera plattformar, samt installationen av tillhörande kodbibliotek *firebase_core* och *firebase_messaging* i Flutter.

Projektet delade upp funktionalitet i två delar; klientsidan för att hantera förfrågningar från Firebase och serversidan för att skicka till Firebase. Denna uppdelning var i syfte att behålla API-nycklar hemliga till klienten fast fortfarande behålla möjligheten att skicka HTTP-förfrågningar till Firebase.

Utvecklingen av klientsidan började med implementationen av funktionalitet för att hämta enhetens långvarig digital nyckel. Denna nyckeln tillsammans med användarens e-postadress sparas i databasen. Detta görs via en HTTP-förfråga till

serversidan med det tidigare nämnda innehållet för att tillägga den i databasen. För situationen där ny användare interagerar med applikationen utan att vara inloggad skapas en gäst e-postadress guest@gmail.com som konsekvent behåller alla användares digitala nycklar. För att konfigurera notifikationer som kommer ifrån Firebase ställdes nödvändiga behörigheter för notifikationer i bakgrunden samt förgrunden. Detta tillåter att meddelande tas emot i klientsidan när användaren aktivt interagerar med applikationen.

För att säkerställa att meddelande hanteras korrekt när applikationen aktivt används implementerades en händelselyssnare som inväntar meddelande från Firebase. Händelselyssnare anropar en metod för att skapa en lokal notifikation med titel och beskrivning av meddelandet. Projektet utökade ursprungliga implementationen för att begränsa längden av den synliga titeln och beskrivningen ifall den överstiger 50 karaktärer.

Sändning av notiser på klientsidan var implementerad i samma sekvens som meddelandesändningen Detta resulterade i att klientsidan hade enkelt åtkomst till titeln och beskrivning av det nya meddelandet. Dessa värden användes för att bilda notifikationens titel och beskrivning. Sändningen till specifika mottagare fortsatte med att hämta enhetsnycklarna, vilket extraherades från databasen baserat på kopplingen mellan e-postadressen och enhetsnyckeln. Detta möjliggjorde utsändningen av notifikationer till alla enheter associerat med användaren.

Utvecklingen av serversidan genomfördes för två huvudsakliga syften. Det första var att omvandla innehållet i HTTP-förfrågningar och korrelera dem till relevanta tabeller i databasen. Det andra syftet var att möjliggöra sändningen av notifikationer från klientsidan till Firebase med hjälp av API-nyckel.

Det första genomfördes genom implementationen av en klass som ansvarar för att bemöta HTTP-förfrågningar kopplat till tilläggningen och hämtning av enhetsnycklar.

Det andra syftet kretsade runt konstruktionen av klass som ansvarade för att hantera förfrågningar i seversidan riktat mot sändningen av notifikationer. Implementation i serversidan krävde konstruktionen av en stöd klass ansvarig av hantering av Firebase funktionalitet. Vilket inkluderade två nya funktionalitet. Det första är hämtning av OAuth 2.0 nycklar för att verifiera och autentisera sändaren. Eftersom nyckel är tidsbegränsat bildades en metod för att kontrollera om nyckeln är fortfarande giltig. Om nyckeln har löpt ut skickas en ny begäran med hjälp av den hemliga API-nycklen för att uppdatera OAuth 2.0 nyckel.

Den andra funktionalitet använder OAuth 2.0 för att sätta headerfältet *Authorization* med autentiseringsuppgifter. Denna nyckel används för att verifiera sändaren i Firebase server. Efter autentiseringen tolkas meddelandet i följande JSON-struktur,

```
{
  "message": {
    "token": "token",
    "notification": {
      "title": "title",
      "body": "description"
    }
  }
}
```

token representerar enhetsnyckeln som Firebase sänder notifikationen till.

4.6 Behörighetssidan

Implementation av behörighetssystemet utfördes primärt i serversidan av applikationen. Utvecklingen av system ledde till bildning av en ny klass ansvarig för behörigheterna. Klassen har två huvud funktionalitet; ena är att uppdatera nuvarande behörighetsnivån och den andra är att hämta denna nivå. Behörighetsnivån lagras lokalt i instansvariabeln som förenämnda funktionalitet har åtkomst till.

Vid initialiseringen av serversidan begränsas behörighetsnivå. Vilket kan föras vidare i de olika klasserna som tar hand om att bemöta HTTP-förfrågningar som kommer till serversidan. Detta användes för att utöka funktionalitet till notifikationssystemet, vilket resulterade i att sändningen av notifikationer endast genomförs om behörigheten tillåter det eller om användarens konto har administratörsbehörighet.

Serversidan krävde även en utökning för att kunna bemöta HTTP-förfrågningar relaterade till nuvarande behörighetsnivå. Det ledde till två nya metoder; ena är ansvarig för att indikera behörighetsnivån och den andra är ansvarig för att uppdatera behörigheten i serversidan. Den sistnämnda funktionalitet använder e-postadressen från den inkommande HTTP-förfråga för att lista ut om kontot är av administratörsbehörighet. Detta avgör om HTTP-förfrågan uppdaterar behörighetsnivån i serversidan.

Utvecklingen av klientsidan krävde minimal insats och omfattade två moment. Det första momentet bestod av konstruktionen av metoder för att hämta och uppdatera aktuella behörighets stadiet via HTTP-förfrågningar. Det andra momentet baserades på att använda resultat av HTTP-förfrågningar för att åtkomma önskat beteende i applikationen. Det sistnämnda omfattade två huvudsakliga syften; det ena var att begränsa möjligheten att skicka meddelanden och notifikationer till andra enheter om serversidan saknar korrekt behörighetsnivå. Det andra syftet var att möjliggöra uppdatering av behöriga stadiet i serversidan från klientsidan via sidan *Mail permission*.

4.7 Skrivassistent

Urvalet av LLM baserades på vissa grundläggande kriterier, såsom att köra modellen lokalt på en dator med standardkomponenter samt att licensvillkoren tillåter kommersiell användning utan några förbehåll. Detta resulterade i några möjliga alternativ som vissa modeller av Deepseek och en annan populär tjänst Ollama. Projektet valde det sistnämnda alternativet för möjligheten av att välja ifrån ett större urval av modeller med varierande mängder parametrar.

Implementering av Ollama gav möjligheten att antingen starta en lokal version av modellen eller att bilda en egen modell utifrån fördefinierade instruktioner. Det sistnämnda krävde ytterligare steg såsom att skapa en ny fil med namnet *Modelfile* och ange tre parametrar; modellens namn, instruktion och temperatur.

Utvecklingen av skrivassistenten krävde att välja och testa lämpliga modeller. Denna process utgick ifrån att hitta modeller som presterar inom rimliga gränser och under 10B parametrar, vilket resulterade i följande alternativ mistral:instruct, tinyllama och phi. Varje modell utsattes för samma förutsättningar där den instruerades med att endast returnera texten med korrekt grammatik, förbättrat tydlighet och bättre flyt. Modellerna testades på samma förutbestämda texter och den bästa presenterade modellen utvaldes. Projektet slutade med mistral:instruct.

För att projektet ska kunna kommunicera med lokala modellen skickar projektets HTTP-förfrågningar till ip-adressen http://127.0.0.1:11434. Denna adress sätt in förinställt och har flera slutpunkter för olika syften fast projektet valde */api/generate* för att generera en ström av respons med en given prompt skickat från projektet serversida.

Implementation av skrivassistenten är uppdelad i två delar; klientsidan och serversidan. Klientsidan börjar med att bilda en knapp med en lämplig ikon som symboliserar skrivassistenten vid meddelande sidan. Därefter implementerades en klass som associeras med sändningen av en prompt och mottagningen av respons. Båda dessa funktioner kan åstadkommas med en gemensam metod. Denna metod skickar en HTTP-förfrågan där prompten inkluderas i JSON-datan. När förfrågan når servern öppnas en ström från serversidan som skickar svar i form av JSON-objekt tillbaka. Klientsidan kan därmed stegvist ta emot textinnehållet och bilda en visuell respons på klientsidan. Något som identifierades tidigt på var att användaren hade behov av att kunna extrahera genererade texten, vilket resulterade i att en kopieringsfunktion integrerades i gränssnittet.

För att möjliggöra kommunikationen mellan serversidan och lokala Ollama-server konstruerades en klass som är ansvarig för att bemöta inkommande HTTP-förfrågningar från klientsidan och initiera en strömmad anslutning mot Ollamas slutpunkt, vilket möjliggör realtidsöverföring av generade text från Ollama-server till klientsidan.

Konstruktionen av metoden började med att avkoda inkommande JSON-data från klientsidan och extrahera prompten som användaren har inkluderat. Därefter skapas en HTTP-förfråga riktad till http://127.0.0.1:11434 /api/generat. Vilket innehåller

tre parametrar; modellen som ska användas, prompten som text generering ska baseras på och *stream: true* som indikerar för Ollama-server att bilda en strömmad anslutning.

Vid mottagningen av respons från Ollama-server delas responsen upp i flera delar. Varje enskild del avkodas successivt och därefter skickas omedelbart till klientsidan med hjälp av en metod som ser till att datan skickas direkt utan någon ytterligare fördröjning.

4.8 Förbättringar till inmatningssätt

Ett central problem som identifierades med tidigare implementationen av inmatningssystemet var inmatningen av longitud- och latitudvärden vid tillägg av nya markörer. Projektet bemötte det problemet genom att tillägga funktionalitet att extrahera longitud- och latitudvärden vid ett tryck på karta.

Denna lösning bygger på att aktivera *onTap*-funktionalitet som var inkluderat i kartinställningarna. När en klient tryckte på en knapp sparades koordinaterna i en instansvariabeln av typen ValueNotifier<LatLng?>. Denna variabel användes för att uppdatera dynamiskt en specificerad gränssnitt element positionerad längst ner till vänster som visualisera de aktuella latitud- och longitudvärden.

Den tidigare implementation orsakade vid varje klick på kartan en fullständig omladdning av sidan. Detta ledde till ökade fördröjningar varje gång en klient tryckte på en ny position. För att lösa detta problemet rekonstruerades uppdateringslogiken genom att omvandla typen av instansvariabeln till ValueNotifier vilket möjliggjorde en selektiv rendering av gränssnitt element som nämnes i förgående stycket.

Utöver det utvecklades en knapp med ett plustecken placerad längst ner till höger. Denna knappen var ansvarig för navigationen till AdminDashboard. Den nuvarande implementation av klassen AdminDashboard endast stödde navigation till huvudmenyn och saknade funktionalitet att föra vidare användaren till korrekt plats utan att de själva väljer korrekt överlägg. Därför vidareutvecklades denna klassen för att inkludera två nya parametrar i konstruktorn. Den ena ansvarar för att aktivera överlägget för tillägg av markörer medan den andra möjliggör överföring av användarens valda longitud- och latitudvärden. Dessa koordinater används för att fylla in motsvarande fält i textfältet för att lägga till nya markörer.

5

Resultat

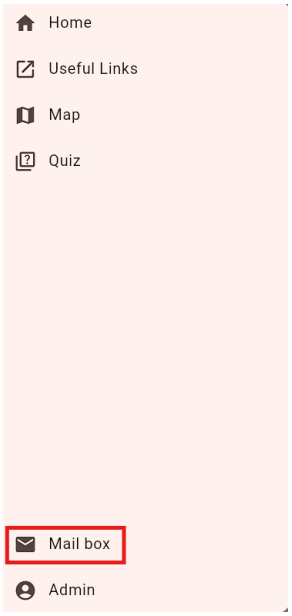
I detta kapitel presenteras resultatet av applikationens vidareutveckling. Allt som har implementerats under projektets gång redovisas, inklusive de uppsatta målen som omfattar ett notifikations- och meddelandesystem, en AI baserad skriv assistent samt en text-till-tall funktion bland andra förbättringar till existerande funktionalitet.

5.1 Meddelandesystem

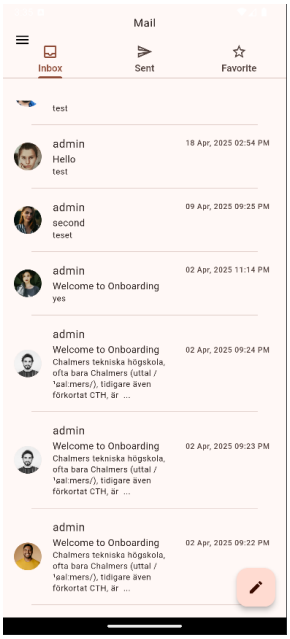
För att implementera meddelandesystemet har en ny dedikerad sida i form av en Flutter widget skapats från grunden och lagts till i navigationsmenyn som syns i Figur 5.1. Sidan är exklusivt tillgängligt för inloggade användare och har liknande beteende till profilsidan det vill säga att knappen i menyn är dold som standard och syns endast för användare som är inloggad. Själva meddelande sidan är tilldelad i tre delar bland annat `Inbox` där användare kan se inkommande meddelanden, `Sent` för meddelande som användaren har skickat och `Favorite` där användaren kan spara meddelande för enklare åtkomst, se Figurer 5.2, 5.3, 5.4. Ett nytt meddelande kan skapas genom att trycka på pennsymbolen i kanten till höger då flyttas användaren till en förenklad form med tre fält, en för mottagarens meddelande adress, en för meddelandets titel och en för innehållet, se Figur 5.5. För att läsa inkommande meddelande räcker det att trycka på önskade meddelandet då öppnas visningsläge för det valda meddelandet, som fyller hela sidan och visar sändarens namn och ikon överst följ med meddelandets titel och innehåll. Till och med har flera gränssnitt element implementerats vid detta läge som förenklar vanligt förekommande interaktioner för användare, mer specifikt så finns en tjärn symbol i den översta hörnan som byter färg till gul vid en klick och lägger till meddelandet i favorit listan, bredvid tjärn symbolen finns två pil symboler som har funktion att byta till nästa eller förgående meddelande i visning vid en klick.

Den nuvarande implementation av systemet tillåter alla användare att skicka meddelande till varandra utan restriktioner, detta är i kontrast till den preliminära designen som antyder att olika användare skulle har roller baserad på arbetsprestation eller grupp som används för bestämma vilka användare får skicka till vilka. Som konsekvens så blev båda implementation och användning av meddelandesystemet som byggdes från grunden märkligt enklare. Utöver här en ändring till profil sidan för administrativa konto skett i relation till meddelandesystemet, i form av sida för

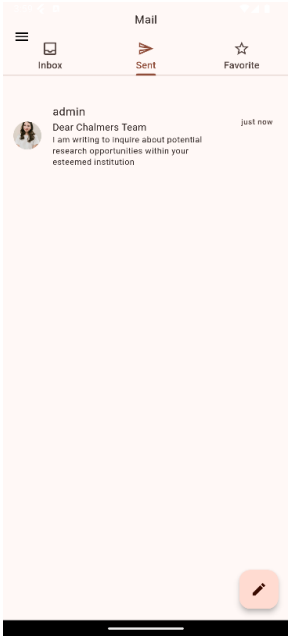
`Mail permissions` där administratören kan med en knapp välja att aktivera eller deaktivera hela meddelandesystemet för alla användare. Funktionen är menad att vara en beredskapsåtgärd vid missbruk av meddelandesystemet, se Figur 5.6.



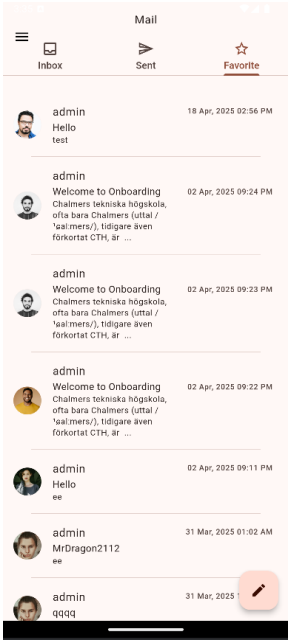
Figur 5.1: Ikon för e-post sidan



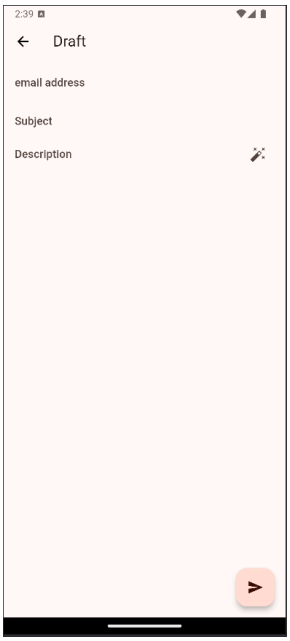
Figur 5.2: Inbox vy



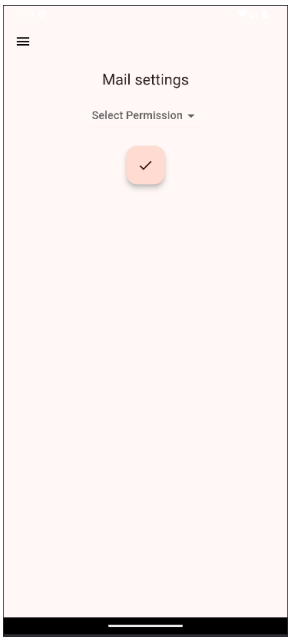
Figur 5.3: Skickade meddelanden



Figur 5.4: Sparade meddelanden



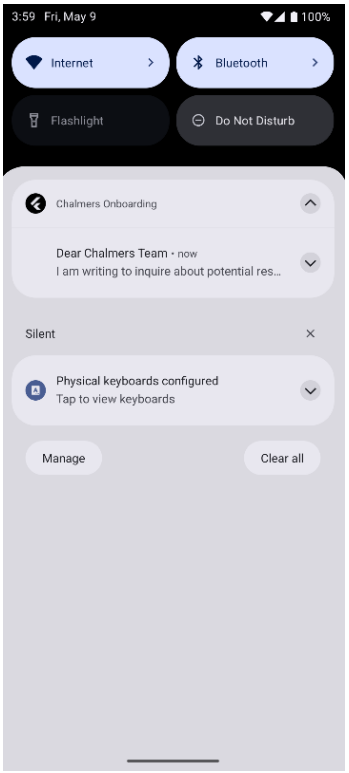
Figur 5.5: Att skriva ett meddelande



Figur 5.6: Sidan för tillstånd kontroll

5.2 Notifikationssystem

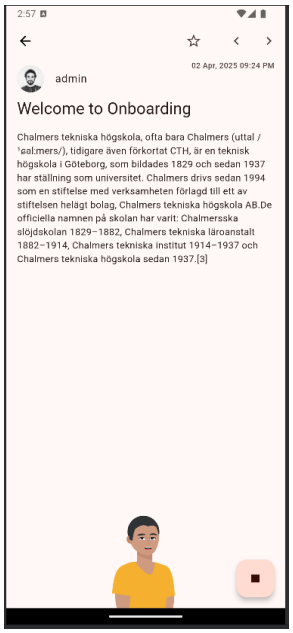
Ett notifikationssystem har implementerats sådant att användare som får inkommande meddelande via e-postsystemet kan se dessa meddelande i form av en avisering på mobiltelefonen. Implementeringen som använder sig av tjänsten Firebase tillåter att aviseringar fungerar för båda android och iOS på samma sätt samt att kunna få aviseringar även om applikationen är stängd. En standard avisering består av två fält för titel och beskrivning med en max längd av 50 tecken som tas automatiskt från meddelandets innehåll, med andra ord så visar en avisering titeln på meddelandet tillsammans med de första 50 bokstäver, se Figur 5.7. Genom att trycka på aviseringen så startas applikationen och tar användaren till visning sidan för det nämnda meddelandet.



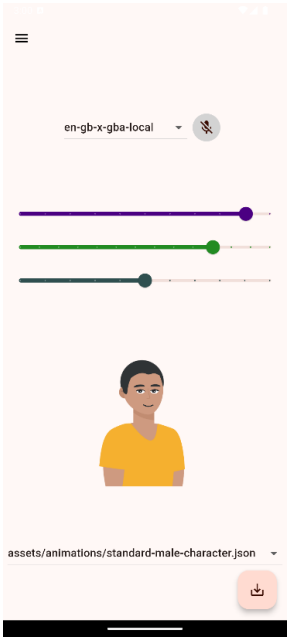
Figur 5.7: avisering för ett inkommande meddelande

5.3 Text-till-tal

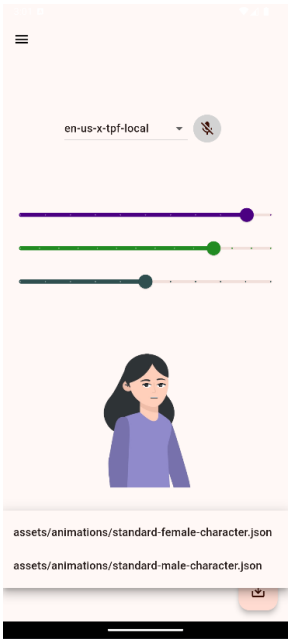
En text-till-tal funktion har lagts till i applikationen och integrerats i e-postsystemet. För att aktivera text-till-tal så räcker det att öppna upp en meddelande i e-postsidan och trycka på TTS-knappen i nedre hörnan som ersätter pennsymbolen i visningssidan för meddelande. Det ärr möjligt att pausa funktionen genom att trycka på TTS-knappen medan den är aktiv, och därefter kan den återupptas genom att klicka på knappen igen. Text-till-tal innefattar en röst som läser texten i sidan samt en 2D avatar som dyker upp under funktionens gång och läppsynkroniserar med rösten och stoppar när TTS rösten har pausats eller stoppats, se Figur 5.8. Utöver det så har en ny alternativ under profil sidan blivit inlagt som tillåter anpassning och redigering av båda rösten och avatar bilden, se Figurer 5.9, 5.10. Denna sida är tillgängligt för alla användare och består av en rullgardinsmeny för att välja en röst från en lista med flera inbyggda tillgängliga röster, bredvid rullgardinsmenyn finns en mikrofon symbol som aktiverar och pausar TTS funktionen vid en klick. Applikationen har en förutbestämd textbit vilken rösten alltid läser när den är aktiverad i denna sida för att ge användaren ett sätt att testa rösten efter en ändring. Under rullgardinsmenyn finns tre stycken horisontella skjutreglaget som är ansvariga för vissa av röstens egenskaper bland annat volym, tonhöjd och talhastighet. Till och med så finns en förhandsvisning av den för närvarande valda av avatar bilden tillsammans med en till rullgardinsmenyn med alternativ för olika avatarer att välja mellan vilket automatiskt ändrar bilden i förhandsvisningen till. För att spara ändringar behöver användaren trycka på `spara knappen` i nedre hörnan och för att kunna se ändringarna så räcker det ladda om sidan eller alternativt lämna till en annan sidan i applikation och senare komma tillbaka.



Figur 5.8: Pågående TTS i ett meddelande



Figur 5.9: inställningssida för TTS och Avatar



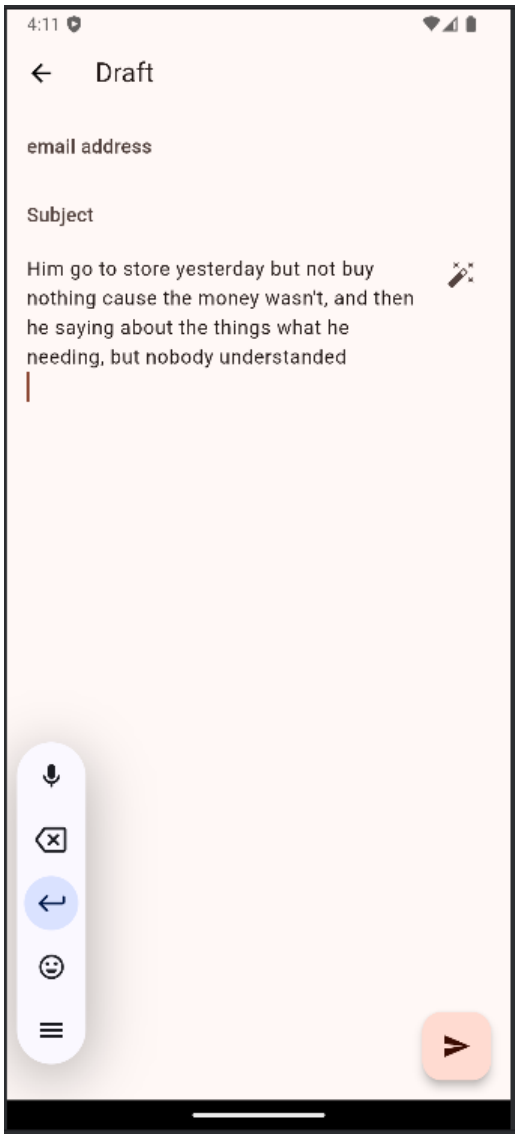
Figur 5.10: Val av en alternativ avatar

5.4 Skrivassistent

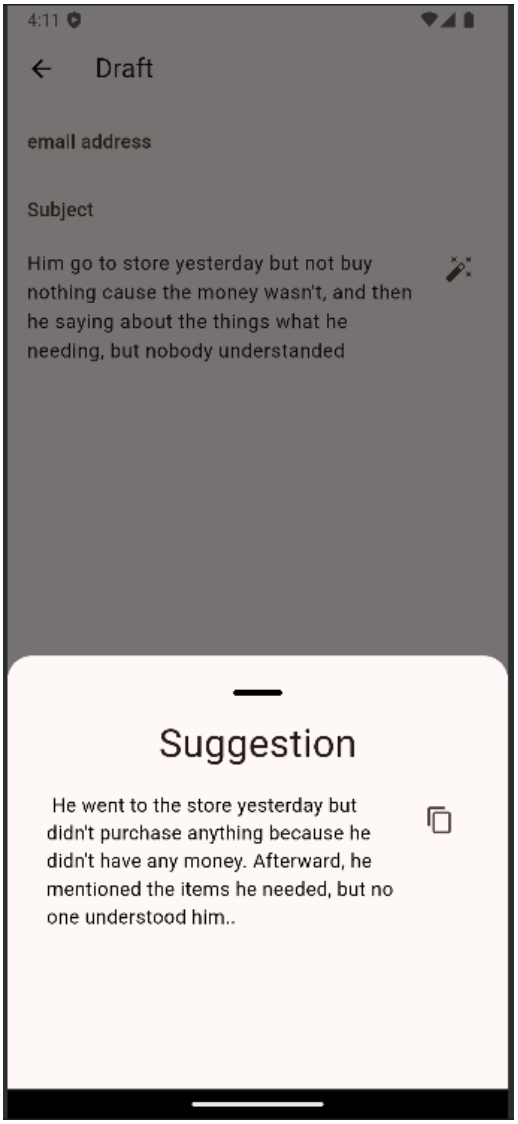
För att stödja användare med skrivande och formulering av meddelanden har en AI-baserad skrivassistent implementerats. Vid skrivande av ett meddelande finns det en dedikerad knapp till sidan om textfältet för meddelandets innehåll som aktiverar AI-assistenten och efter en klick tar innehållet av textfältet och med hjälp av AI:n skriver om det med förbättrad grammatik och på ett lättförståelig och formellt sätt. Dessutom istället för att direkt ersätta innehållet med den bearbetade texten så dyker upp ett nytt dragbar text fält som visar den nya versionen av texten tillsammans med en knapp för att kopierar den vid behov. Detta för att förenkla processen och samtidigt ge användaren valet att jämföra och välja mellan den originella och den bearbetade texten på egen hand, se Figurer 5.11, 5.12. För att driva skrivassistenten används en lokal version av LLM:n `Mistral AI` som i enlighet med en hårdkodad prompt skriver om texten på ett passande sätt. Prompten är följande:

```
You are a text improver.
When a user gives you text, your ONLY job is to rewrite it with:
  Correct grammar and spelling
  Improved clarity
  Better flow


Do NOT:
  Explain your changes
  Add anything else

Only return the improved version of the input text, and nothing
   more.
```

Förutom denna knapp som finns i sidan för skapande av nya meddelande så kan användaren inte interagera med skrivassistenten eller AI modellen på något sätt.



Figur 5.11: Original text av ett nytt meddelande

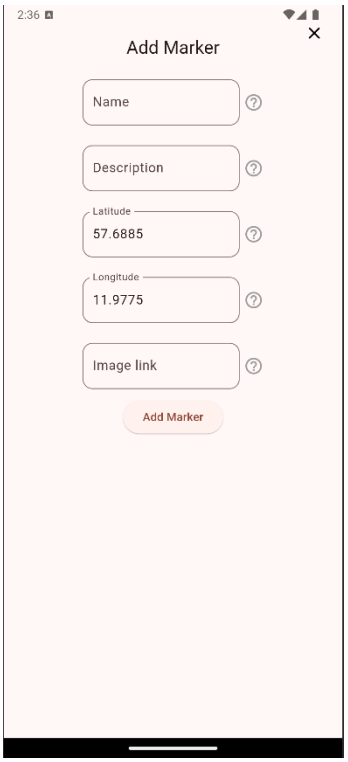


Figur 5.12: Bearbetade text med skrivassistenten

5.5 Övriga ändringar

Applikationen hade existerande funktioner som har utrymme för vidare optimering och det som var mest relevant är tillägg av nya kartmarkörer som tidigare krävde att gå in i sidan `Admin dashboard` följd av `add marker` och manuellt mata in siffrorna för koordinater med resterande information. För att förenkla processen har en ny knapp implementerats i karta-sidan tillsammans med en indikator i kanten som visar koordinaterna för en punkt på kartan efter att trycka på den, se Figur 5.13. Genom att trycka på den önskade platsen på kartan så går det att trycka på den nya knappen för att ta användaren eller administratören i detta fall direkt till `add marker` sidan. Den automatiskt fyller in fälten för longitud och latitud med värdera från den utvalda platsen. Dessa värden även indikeras i dem tidigare nämnde fälten så att administratören behöver endast fokusera på de andra mer relevanta inmatningsfälten som `Name` och `Description`, se Figur 5.14. Resten av processen utförs precis som tidigare utan några ändringar.

Figur 5.13: Uppdaterade karta sidan



Figur 5.14: Automatisk inmatning av koordinater

6

Diskussion & Slutsatser

I detta kapitel reflekteras över projektets resultat i förhållande till dess frågeställningar och syfte. Diskussionen tar upp centrala aspekter såsom användarvänlighet, begränsningar under utveckling samt etiska och miljömässiga betänkande för att ge en helhetsbild av projektets genomförande och dess konsekvenser. Sen avslutas kapitlet med möjliga förbättringsområden följd av en sammanfattande slutsats som belyser projektets framgångar.

6.1 Diskussion

6.1.1 Användarvänlighet

Det är lätt att bli överväldigad när en person presenteras för en stor mängd information på en gång, särskilt om man inte har en teknisk bakgrund eller inte är bekant med det aktuella ämnet. Därför kan det ofta vara klokt att begränsa antalet alternativ eller val som användaren bemötas och istället presentera en produkt som bra funkar. Med det i åtanke har vissa val gjordes under projektets gång med avsikt att minska antalet val som användaren ställs inför och istället begränsas till vad som anses vara mer relevant. Till exempel består `TTS & Avatar` sidan i profilen av endast en rullgardinsmeny för rösten och tre skjutreglage för röst trots att `flutter_TTS` erbjuder en mängd fler justerbara egenskaper, projektet har bestämts att behålla sig till de tre som verkar vara mest relevanta och bort sig från onödiga och förvirrande val. Ett liknande tillvägagångssätt användes vid design av meddelande systemet och skrivassistenten. Exempelvis så använder meddelandesystemet ett simpelt gränssnitt som använder sig av endast en knapp på skärmen åt gången. Till och med så har projektet valt inte inkludera några alternativ för textformatering som att ändra typsnitt stil eller färg vid skrivande av nya meddelande. När det kommer till skrivassistenten har det diskuterats att lägga till justerbara reglage för hur strikt bearbetande av texten är eller vad det är som prompten säger till modellen. Projektet bestämde att hålla sig till en enkel knapp som funkar utan behov för justering eller uppsättning. Under projektets gång har prioritet varit vid att skapa ett system som känns intuitivt att använda för flesta användare utan behöv för en manual eller specifika bakgrund information, målet har varit att ska kunna starta upp applikation och kunna använda den nya funktionen på egen hjälp och vara bekvämt med det. Det är viktigt dock att inte överdriva denna tankesätt så att det förhindrar användbarhet av applikationen, det är inte meningen att ta bort

funktioner från applikationen utan istället så ska designen hämna i balans mellan enkelhet och funktionalitet och det är det som projektet försökte uppnå.

6.1.2 Sociala aspekter och miljö

I framtiden kommer personer med olika bakgrunder använda applikationen och då är det viktigt att designen under projektet inte resulterar i en applikation som riktar sig mot en specifik folkgrupp samtidigt som den struntar i andra gruppen. Flera aspekter togs i beaktande under planeringsfasen, såsom användare med olika etnicitet och kön. Gruppen ville skapa en upplevelse som i största möjliga mån är inkluderande och tillgänglig för flesta antal personer, även om detta koncept kan framstå som något otydligt när det gäller funktioner som notifikationer eller mejl, så är det tydligare i text-till-tal funktionen tack vare dess mer mänsklig och relaterbara egenskaper. På sidan `TTS & Avatar` under profilen, där användare kan konfigurera och anpassa sin önskade röst och avatar, strävade gruppen efter att erbjuda ett urval av alternativ för att passa så många olika personer som möjligt. Samlingen av röster tillgängliga i `Flutter_TTS` inkluderar olika engelska dialekter samt både manliga och kvinnliga röster. Trots att urvalet erbjuder viss variation är det dock inte så omfattande som gruppen hade önskat, och alla röster låter inte lika naturliga. Därför undersöktes även andra alternativ, men i slutändan var `Flutter_TTS` det mest lämpliga alternativet då det inte medför några kostnader. Som en åtgärd för att motverka detta problem implementerades skjutreglagen för visa röst egenskaper så att användaren kan själv justera rösten på det sättet som är mest lämpligt.

Förutom detta så finns inga flera relevanta aspekter att ta upp eftersom projektet har satsathuvudsakligen på att skapa en medium för kommunikationen. Däremot är det så att överseende över hur systemet används är ansvaret för administratörer som kommer hantera applikationen. Det vill säga att detta faller utanför projektets omfattning.

6.2 Begränsningar

6.2.1 Begränsningar i testning för iOS enheter

Testning på samtliga funktioner begränsade sig till Android och webb versioner av applikationen på grund av att ingen i gruppen äger eller har tillgång till en Iphone eller andra iOS enheter. Detta togs i hänsyn till vid planering så att alla funktioner blev implementerad så att de ska funka utan problem på iOS enheter trots att det inte går att testa. Detta är däremot en teoretisk bedömning och det finns fortfarande risk att applikationen kan leda av buggar eller oväntat beteende vid körning på en riktigt iOS enhet.

6.2.2 Begränsningar i relation till Kostnader

En huvud begränsning i projektet var val av verktyg som uppfyller kraven för samtliga funktioner och är dessutom kostnadsfria. Detta gällde särskilt för text-till-tal funktionalitet samt AI skrivassistenten med anledning att verktyg som gör dessa funktioner är relativt komplicerade och kostnadskrävande för utvecklaren att behålla och utveckla. Konsekvent brukar det kräva något sorts betald prenumeration för att få tillgång till dessa verktyg. Efter en förläng undersökning av flera alternativ hittade gruppen kostnadsfria varianter i form av `Flutter_TTS` för text-till-tal och `Mistral Ai` via plattformen `Ollama` för skrivassistenten och trots att det gick att implementera önskade funktionalitet på detta sätt så fanns det också kompromisser som behövde göra. I fallet med `Flutter_TTS` kan vissa röster låta ganska onaturliga och ha en obehaglig röst, det kan även sällan hända att TTS funktionen laggar eller kraschar. När det gäller skrivassistenten så är prestandan ganska tillfredsställande men eftersom modellen körs lokalt så ta den upp en viss mängd av enhetens resurser för att kunna köra, valet av LLM tog hansyn till detta och prioriterade modeller som inte är lika resurskrävande vilket gäller för `Mistral AI` som är relativt lättkörd. Trots detta anser gruppen att det något som är värt att påpeka och möjligtvis undersöka vidare i framtiden vid behov.

6.2.3 Riskfaktorer

Under utförande av arbetet och vid implementering av samtliga funktioner har flera potentiella riskfaktorer identifierats i form av designval som har vissa nackdelar eller även kod som har förmågan att vara problematisk. Ett exempel är vid implementering av notifikation systemet där anropet till *getAccessTokenServer()* originellt skedde på klientsidan vilket medförde flera säkerhetsrisker. Om OAuth 2.0 genereras på klientsidan kan den bli tillgänglig för externa användare. Detta innebär att någon obehörig har åtkomst till Firebase konsol, vilket kan leda till utförande av skadliga operationer för datastöld, manipulation eller exploatering av tokens. Detta problem åtgärdas genom att flytta skapande process av tokens till serversidan så att användaren inte har tillgång till privata information eller obehöriga funktioner.

Ett annat exempel på en designval som kunde ha blivit en riskfaktor är designen för skrivassistenten. Under tidigare steg av projektet så diskuterades flera möjliga sätt för hur AI modellen ska implementeras och vilken form skrivassistent funktionen ska existera. Ett av förslag som senare blev avslagen är att skapa en meny med flera skjutreglagen liknande till de i *TTS & Avatar* inställning sida samt en textbox som tillåter användaren att kommunicera med AI modellen. Tanken bakom denna design är att ge användaren större kontroll över hur skrivassistenten bearbetar texten så att det blir möjligt att byta mellan en prompt som gör texten mer formellt till en som är mer avslappnad och från en som förlänger texten till en som förkortar istället och så vidare. Gruppen antog dock att en stor utrymme för interaktion mellan användaren och modellen kan potentiellt leda till att så kallade "illvillig aktörss-om kan utnyttja detta för att påverka AI modellen på oförutsebara sätt genom manipulering av prompten eller andra okända svagheter vilket kan leda till farliga konsekvenser för serversidan. Därför bestämde gruppen att ta bort sig från denna

design och begränsa interaktionen mellan användare och AI modellen till bara det nivå som krävs för funktionen för att undvika risken.

Som sagt så har gruppen identifierat och åtgärdad dessa riskfaktorer men trots det så finns möjligheten att andra liknande riskfaktorer existerar men blev inte upptäckta. Därför lade gruppen stor vikt vid att säkerställa hantering av användare information och hålla implementerade funktioner bara till nivå av funktionalitet och tillstånd som krävs och inte mycket mer än det.

6.3 Möjliga förbättringar och alternativa tillvägagångssätt

Verktyg som utnyttjas och designval som har tagits inom projektet är tillämpat för att motverka de utmaningar som kan förhindra arbetet. Trots att gruppen anser att valda verktyg är för det mesta passande så finns också nackdelar som öppnar möjligheten för vidareutveckling och alternativa lösningar. Följande är några förslag på var som skulle kunna utvecklas vidare i framtiden eller annars göra på ett annat sätt än det som valdes för projektet:

- Istället för att köra en lokal AI modell så går det alternativt att använda en molnbaserad tjänst som via API. Detta skulle förmodligen vara mer mindre kostnadseffektivt men fördelen är att det blir ingen behöv att ställa upp den lokala versionen av modellen på serversidan och avsätta resurser till den. Detta är dock inte strikt en förbättring utan snarare ett möjligt alternativ där utvecklaren kan välja baserad på behovet. Båda alternativ i detta fall har meriter till sig och passar för olika skäl.
- Gällande valet av avatar i inställningsidan så är antalet förgjorda alternativ som *Lottie* erbjuder begränsad, då finns det utrymme att hitta annat verktyg som ger möjligheten att konfigurera avataren på flera sätt eller även hitta en resurs som erbjuder flera kompatibla avatarer.
- Ett liknande förslag kan ges i förhållande till valet av röst för text-till-tall funktionen. *Flutter_TTS* erbjuder en bred samling av röster med olika egenskaper dock finns stor variation i prestanda mellan rösterna, detta tillsammans med faktum att rösterna saknar konkret beskrivning för egenskaperna på olika alternativ gör det svårt att hitta den rösten som passar bäst utan att testa var enda alternativ för hand. Det skulle ha varit möjligt att komma på något system att filtrera och dela rösterna i olika kategorier baserad på vissa egenskaper som tillåter valet av den önskade rösten på ett enklare sätt.

6.4 Slutsats

Syftet med projektet var att vidareutveckla Chalmers onboarding-applikation genom att införa ett omfattande system för meddelanden och notifikationer, tillsammans med flera sammankopplade funktioner. Målet var att skapa ett medium för enkel och inkluderande kommunikation mellan användare med olika bakgrunder, med särskild fokus på användarvänlighet och tillgänglighet både för de nyintroducerade och befintliga funktionerna i applikationen.

Projektet anses ha lyckats med att uppnå detta mål genom att implementera ett flertal funktioner i följande ordning:

Det första steget var att införa en text-till-tal-funktion (TTS) i form av en röst med tillhörande avatar. Första version av funktionen implementerades som en fristående sida i applikationen med hjälp av Flutters inbyggda Text-till-tal bibliotek (`Flutter_TTS`) och därefter integrerades en till sida under användare profilen där inställningar för röst och avatar kunde justeras. En TTS-knapp infördes även i meddelandevy för att möjliggöra uppspelning av meddelanden.

Därefter utvecklades ett meddelandesystem som gör det möjligt för inloggade användare att kommunicera med varandra. Systemet bygger på nya databasstrukturer och en separat `Mail` sida med flera undersidor, inklusive vyer för inkommande, skickade och sparade meddelanden. I visningsläget för meddelanden integrerades TTS-knappen, och en dedikerad sida för att skriva nya meddelanden skapades. I denna sida integrerades även en knapp för att använda AI-skrivassistenten för att omformulera innehållet av texten i beskrivningsfältet.

Notifikationssystemet utvecklades näst med hjälp av Firebase, vilket möjliggjorde visning av inkommande meddelanden i form av notifikationer på både Android och iOS enheter. Detta gjordes för att undvika komplexa, plattformsberoende lösningar och säkerställa en stabil användarupplevelse för båda bakgrunds- och förgrundsaviseringar.

Efter det implementerade projektets en AI-skrivassistent som använder `Mistral AI`, som valdes framför den ursprungligen planerade modellen `DeepSeek` på grund av bättre dokumentation och mindre resurskrav på hårdvaran. Modellen kördes lokalt via plattformen `Ollama` och interaktionen begränsades till en enkel knapptryck för att bearbeta text enligt en hårdkodad prompt. Detta var ett designval med målet att minimera både implementeringstid och potentiella säkerhetsrisker.

Slutligen implementerades förbättringar i inmatningsmetoder. Det identifierades att tillägget av kartmarkörer var det mest kritiska förbättringsområdet och då lades till en ny knapp på karta sidan där administratörer kan klicka på en plats i kartan för att välja koordinater och direkt trycka på den nya knappen. Detta öppnar sidan `Add Markers` i `Admin Dashboard` och fyller koordinaterna automatiskt i rätt fält , vilket gör att administratören kan fokusera på att fylla i de andra mer relevanta fält. Trots att detta var den enda förbättringen inom detta område, bedöms målet som uppfyllt då det hanterade ett betydande användarproblem i den tidigare versionen.

Sammanfattningsvis har projektet levererat alla planerade funktioner, trots vissa

justeringar i implementeringen jämfört med den ursprungliga planen. Dessa förändringar har dock inte påverkat användarupplevelsen negativt.

Projektgruppen anser att projektet, trots tekniska, tidsmässiga och resursrelaterade utmaningar, har lyckats skapa ett effektivt, lättanvänt och pedagogiskt medium som främjar kommunikation mellan Chalmers personal med olika bakgrunder. Resultatet är ett intuitivt gränssnitt som är enkelt att lära sig och bekanta sig med, och som bidrar till en mer inkluderande digital arbetsmiljö för båda nyanställda och befintliga medarbetare.

Litteraturförteckning

[1] Chalmers. Chalmers historia. Hämtad: Apr 1, 2025. URL: https://www.chalmers.se/om-chalmers/tradition-och-hogtider/chalmers-historia/.

[2] Mariot Tsitoara. *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Apress, Berkeley, CA, 1 edition, 2020.

[3] Elvis C. Foster and Shripad Godbole. *Database Systems: A Pragmatic Approach*. Springer, 3 edition, 2016. Available via Chalmers Library E-book Collection.

[4] Sudhir K. Routray, Abhishek Javali, K. P. Sharmila, Mahesh Kumar Jha, Maria Pappa, and Monika Singh. Large language models (LLMs): Hypes and realities. *2023 International Conference on Computer Science and Emerging Technologies (CSET)*, pages 1–6, 2023. doi:10.1109/CSET58993.2023.10346621.

[5] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for large language models: A survey, 2023. arXiv preprint, hämtad: Feb 12, 2025. URL: https://arxiv.org/abs/2309.01029.

[6] T. Biscontini. Application programming interface (API), November 2024. Hämtad: Apr 4, 2025. URL: https://research.ebsco.com/linkprocessor/plink?id=3ef3574f-c975-3565-8026-b8024a623b63.

[7] Lottie for Flutter. Lottie - Flutter package, 2025. Hämtad: Apr 4, 2025. URL: https://pub.dev/packages/lottie.

[8] Flutter TTS Developers. Flutter TTS - text-to-speech package, 2025. Hämtad: Apr 4, 2025. URL: https://pub.dev/packages/flutter_tts.

[9] Google. Firebase, 2025. Hämtad: Maj 8, 2025. URL: https://firebase.google.com/.

[10] Google. Firebase cloud messaging documentation, 2025. Hämtad: Maj 8, 2025. URL: https://firebase.google.com/docs/cloud-messaging/.

[11] Google. googleapis: Auto-generated Dart client libraries for Google APIs, 2023. Hämtad: Maj 8, 2025. URL: https://pub.dev/packages/googleapis/versions/13.1.0.

[12] Google. googleapis_auth: Obtain access credentials for Google services using OAuth 2.0, 2023. Hämtad: Maj 8, 2025. URL: https://pub.dev/packages/googleapis_auth/versions/1.5.1.

[13] Sanjay Bhattacharya and Vaskar Saha. How to write a research grant proposal. *Indian Journal of Medical Microbiology*, 49:100482, 2024. URL: https://www.sciencedirect.com/science/article/pii/S0255085723001986?via%3Dihub#sec14, doi:10.1016/j.ijmmb.2023.100482.

[14] Chen Kuang Piao, Yonni Ezzati-jivan, Naser Dagenais, and Michel R. Dagenais. Distributed architecture for an integrated development environment, large trace analysis, and visualization. *Sensors*, 21(16):5560, 2021. URL: https://www.mdpi.com/1424-8220/21/16/5560, doi:10.3390/s21165560.

[15] R. K. Deshmukh, S. Markandey, and P. Sahu. Mobile application development with Android. *International Journal of Advances in Applied Sciences*, 7(4):317, 2018. doi:10.11591/ijaas.v7.i4.pp317-321.

[16] J. Ungvarsky. Flutter (software), March 2025. Hämtad: Apr 4, 2025. URL: https://research.ebsco.com/linkprocessor/plink?id=5114df9b-4364-3834-b22d-80a426185533.

[17] GeeksforGeeks. Introduction to Kotlin, 2025. Hämtad: Maj 8, 2025. URL: https://www.geeksforgeeks.org/introduction-to-kotlin/.

[18] Ollama. Mistral model — Ollama, 2025. Hämtad: Maj 8, 2025. URL: https://ollama.com/.

[19] Mistral AI. Mistral AI, 2025. Hämtad: Maj 8, 2025. URL: https://mistral.ai.

[20] Google. pub.dev – the official repository for Dart and Flutter packages, 2023. Hämtad: Maj 8, 2025. URL: https://pub.dev/.

A

Appendix 1

I denna sektion bifogas bilder och illustrationer som inte har inkluderats i huvudrapporten, men som bedöms vara viktiga för förståelsen av arbetet. Dessa material kompletterar rapportens innehåll och ger läsaren bättre inblick i hur arbetet har utförs.

A.1 Nya tabeller i databasen

```
CREATE TABLE account_settings (
 id SERIAL PRIMARY KEY,                    -- Auto-incrementing unique identifier
 email VARCHAR(255) UNIQUE NOT NULL,       -- Email of the user
 rate DOUBLE PRECISION NOT NULL,           -- rate of speech
 volume DOUBLE PRECISION NOT NULL,         -- volume of speech
 pitch DOUBLE PRECISION NOT NULL,          -- pitch of speech
 avatar TEXT NOT NULL,                     -- avatar route
 FOREIGN KEY (email)
  REFERENCES accounts(email)               -- Must be an existing email in the 'account' table
  ON DELETE CASCADE
);
```

Figur A.1: Tabell som lagrar inställningar för text-till-tal och Avatar

```
CREATE TABLE device_info (
 id SERIAL PRIMARY KEY,                    -- Auto-incrementing unique identifier
 email VARCHAR(255) NOT NULL,              -- Email of the user
 device_token TEXT NOT NULL,               -- Token for push notifications
 FOREIGN KEY (email)
  REFERENCES accounts(email)
  ON DELETE CASCADE,
 unique(email,device_token)
);
```

Figur A.2: Tabell som lagrar Firebase tokens för alla unika användare för aviseringar

```
CREATE TABLE messages (
 id SERIAL PRIMARY KEY,
 profileImage VARCHAR(255),                -- URL of the sender's profile picture
 username VARCHAR(255),                    -- Name of the sender's account
 subject VARCHAR(255),                     -- Subject / title of the message
 body TEXT,                                -- Main content of the message (AES encrypted)
 dateTime TEXT                             -- Timestamp when the message was created
);
```

Figur A.3: Tabell som lagrar innehållet av aviseringar

```
CREATE TABLE inbox_messages(
 id SERIAL PRIMARY KEY,                    -- Unique identifier for each entry in the inbox_messages table
 message_id INT NOT NULL,                  -- ID of the message being favorite
 email VARCHAR(255) NOT NULL,              -- Email of the account sending the message
 FOREIGN KEY (email)
  REFERENCES accounts(email)               -- Must be an existing account in the accounts table
  ON DELETE CASCADE,                       -- Delete inbox_messages when the corresponding account is deleted
 FOREIGN KEY (message_id)
  REFERENCES messages(id)                  -- Must be an existing message in the messages table
  ON DELETE CASCADE,                       -- Delete inbox_messages when the corresponding account is message
 UNIQUE(email, message_id)                 -- Ensure that each message can only be sent once by each account
);
```

Figur A.4: Tabell som lagrar information om meddelande i inbox listan

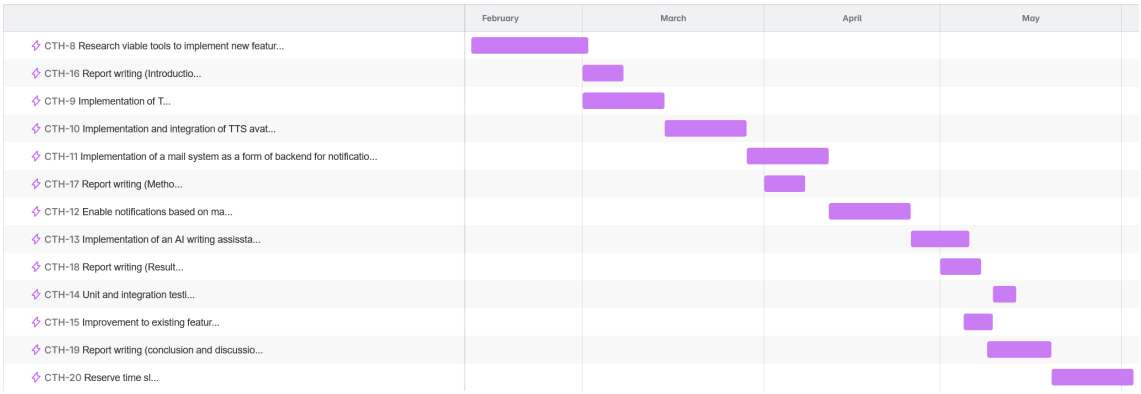
```
CREATE TABLE sent_messages (
 id SERIAL PRIMARY KEY,                    -- Unique identifier for each entry in the sent_messages table
 message_id INT NOT NULL,                  -- ID of the message
 email VARCHAR(255) NOT NULL,              -- Email of the account
 FOREIGN KEY (email)
  REFERENCES accounts(email)               -- Must be an existing account in the accounts table
  ON DELETE CASCADE,                       -- Delete sent_messages when the corresponding account is deleted
 FOREIGN KEY (message_id)
  REFERENCES messages(id)                  -- Must be an existing account in the accounts table
  ON DELETE CASCADE,                       -- Delete sent_message when the corresponding account is message
 UNIQUE(email, message_id)                 -- Ensure that each message can only be sent once by each account
);
```

Figur A.5: Tabell som lagrar information om meddelande i skickade listan

```
CREATE TABLE favorite_messages(
 id SERIAL PRIMARY KEY,                   -- Unique identifier for each entry in the favorite_messages table
 message_id INT NOT NULL,                 -- ID of the message being favorite
 email VARCHAR(255) NOT  NULL,            -- Email of the account sending the message
 FOREIGN KEY (email)
  REFERENCES accounts(email)              -- Must be an existing account in the accounts table
  ON DELETE CASCADE,                      -- Delete favorite_message when the corresponding account is deleted
 FOREIGN KEY (message_id)
  REFERENCES messages(id)                 -- Must be an existing message in the messages table
  ON DELETE CASCADE,                      -- Delete favorite_message when the corresponding account is message
 UNIQUE(email, message_id)                -- Ensure that each message can only be sent once by each account
);
```

Figur A.6: Tabell som lagrar information om meddelande i favorit listan

A.2 Gantt schema för tidplanen enligt planerings-rapport



Figur A.7: Tidplanen för att utföra projekt arbetet

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

CHALMERS