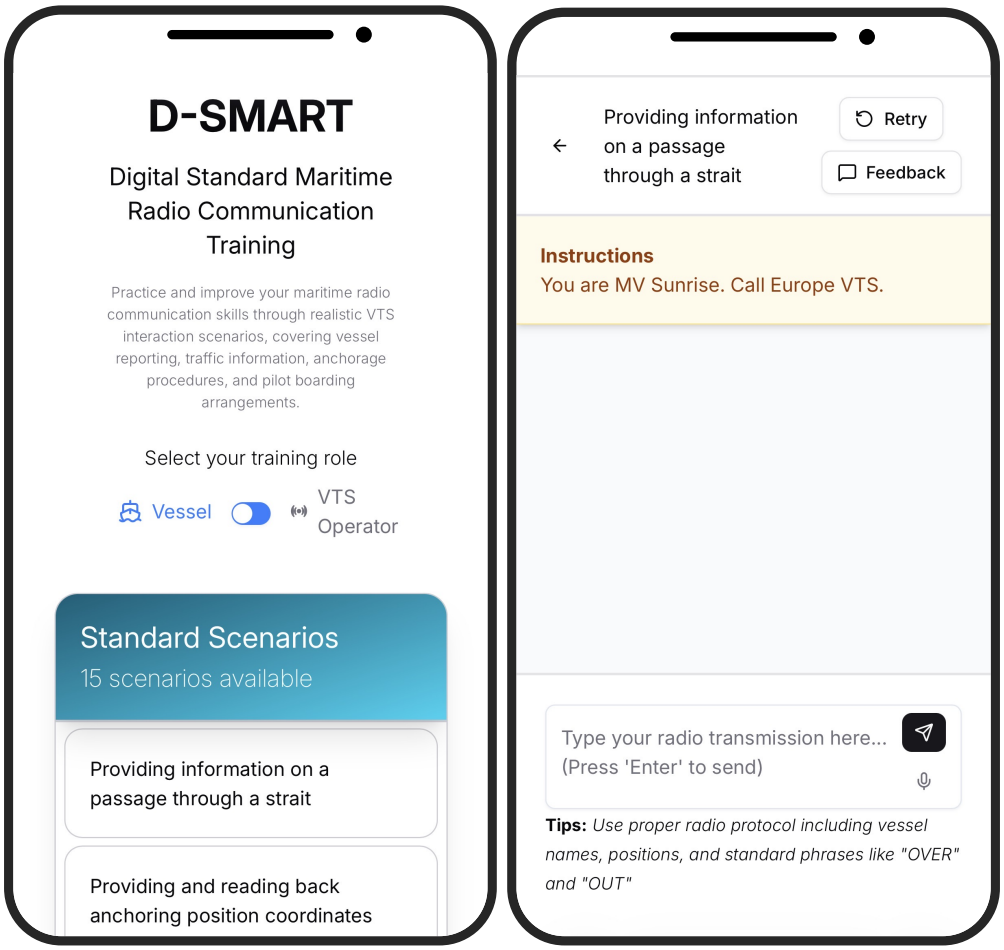


Interaktiv chatbot för maritim kommunikation

Examensarbete inom högskoleprogrammet Datateknik

Jesper Nyberg & Fred Egger

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2026
www.chalmers.se

Examensarbete 2026

Interaktiv chatbot för maritim kommunikation

Jesper Nyberg
Fred Egger

CHALMERS

GÖTEBORGS UNIVERSITET

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Göteborg 2026

Interaktiv chatbot för maritim kommunikation
Jesper Nyberg
Fred Egger

Handledare: Felicia Surdu
Examinator: Johannes Åman Pohjola

Examensarbete 2026
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Skärmdump från mobilvyn av webbapplikationen

Skriven i LaTeX
Göteborg 2026

Interaktiv chatbot för maritim kommunikation
Jesper Nyberg & Fred Egger
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Rapporten behandlar implementationen av en chatbot för maritim kommunikation genom en webbapplikation. Omkring 80% av olyckor som involverar kommersiella fartyg orsakas av mänskliga fel, varav uppemot hälften av dessa beror på undermålig kommunikation. Därmed finns det ett behov av att tillhandahålla ett lättillgängligt och interaktivt utbildningsverktyg för maritim kommunikation. Webbapplikationen erbjuder förutbestämda scenarier med en chatbot som kontinuerligt genererar feedback på användares meddelanden.

Användare kan agera som VTS operatör eller fartygrepresentant i de existerande scenarierna och chatboten antar den återstående rollen. Under scenariots gång får användaren instruktioner om vad det nästa meddelandet ska innehålla för information. För varje skickat meddelande genereras feedback med kommentarer om vad som är korrekt eller inte i meddelandet. Den genereras med hjälp av en hybridlösning av algoritmer och promptning mot en AI-modell. Algoritmerna undersöker meddelandets struktur och användningen av Phonetic Alphabet, Message Markers samt Ambiguous Words. AI-modellen används för att både korrigera stavfel och för att kontrollera betydelseinnebörden i ett meddelande. Hybridlösningen bedömer meddelandets korrekthet för att bestämma huruvida scenariot ska gå vidare till nästa instruktion.

Den resulterande webbapplikationen är en interaktiv och användarvänlig chatbot för maritim kommunikation, anpassad för både datorer samt mobila enheter. Den genererade feedbacken visas för användaren i en sidopanel, färgkodad med grön, gul eller röd för att visa ett meddelandes korrekthet. Chatboten har en tydlig förbättringspotential genom att exempelvis bedöma fler sorters tekniska begrepp samt tillhandahålla interaktiv feedback för att fortsatt förbättra dess användbarhet.

Nyckelord: webbapplikation, chatbot, maritim, kommunikation, feedback, algoritm

Förord

Detta examensarbete har genomförts vid Chalmers Tekniska Högskola under vårterminen 2026 inom ramen för Chalmers-projektet D-SMART.

Vi vill tacka vår handledare Felicia Surdu för vägledning under arbetets gång, samt Annamaria Gabrielli & Daniel Ernstsson för ett engagerat samarbete och värdefull återkoppling. Vi vill även tacka John J. Camilleri för hjälp med API-åtkomst till AI-modellen samt en VPS-server för hosting av applikationen. Slutligen vill vi även tacka de personer som deltagit i användartestningen.

Jesper Nyberg & Fred Egger, Göteborg, Maj 2026

Akronymer

Nedan listas akronymer som har använts i denna rapport:

AI	Artificiell Intelligens
CSS	Cascading Style Sheets
HOC	Higher-Order Component
HTML	HyperText Markup Language
LLM	Large Language Model
MV	Motor Vessel
SMCP	Standard Maritime Communication Protocol
UI	User Interface
VTS	Vessel Traffic Service

Innehåll

Figurer

Tabeller

1
Inledning

Detta kapitel presenterar bakgrunden till arbetet, dess syfte och mål samt de avgränsningar som bedömts varit nödvändiga.

1.1 Bakgrund

Enligt International Maritime Organization (IMO) sker 80% av olyckor som involverar kommersiella fartyg på grund av mänskliga fel. Av dessa olyckor orsakas uppemot hälften av bristfällig kommunikation[1]. Detta trots teknisk utveckling samt kontinuerligt arbete för att förbättra hur kommunikationen sker mellan fartyg och fartygstrafikservice. Därav existerar det ett fortsatt behov av att förenkla lärandet och öka spridningen av utbildning av denna kommunikation och det språkbruk som behövs för att göra kommunikationen effektiv. Standard Maritime Communication Protocol (SMCP) är ett centralt begrepp inom maritim kommunikation. Det är ett standardiserat ordförråd framtaget för att minimera språkliga missförstånd och öka säkerheten till sjöss[2].

Med detta som bakgrund har EU finansierat Erasmus-projektet DigiMar, som har till uppgift att utveckla digitala läromedel ämnad för maritim kommunikation. Detta examensarbete är beställt av Chalmers Tekniska Högskola och bygger vidare på DigiMar-projektet under namnet D-SMART. Avsikten med detta digitala verktyg är att tillhandahålla en webbsida med utbildningsmaterial för maritim kommunikation samt chatbot med scenarier för att träna användares kunskaper. D-SMART innefattar utöver detta ytterligare två projekt, för utveckling av webbsida med interaktivt utbildningsmaterial respektive utveckling av databas för detta material.

1.2 Syfte

Syftet med detta projekt är att utveckla ett interaktivt och lättillgängligt hjälpmedel som kan användas i utbildningssyfte för att underlätta inlärningen av maritim kommunikation. Genom att tillhandahålla en plattform för att öva vanligt förekommande maritima scenarion syftar projektet till att öka studenters behärskning av maritima språkkonventioner och därigenom bidra till att säkerheten förbättras inom sjöfarten.

1.3 Mål

Chatboten ska ha följande funktionaliteter:

- Tillhandahålla relevanta scenarion av maritim kommunikation till användare.
- Ge konstruktiv kritik under pågående samt efter avklarat scenario.
- Kunna användas genom både skriftlig samt muntlig kommunikation av användare.
- Kunna identifiera när användare väljer felaktiga eller saknar korrekta ämnesspecifika begrepp.
- Vara användarvänlig genom att, trots stavfel eller otydligt uttal, förstå vad användaren vill förmedla.
- Vara tillgänglig till användare genom en webbapplikation.
- Integreras mot extern databas via API för att säkerställa att korrekt information ges till användare.

1.4 Avgränsningar

Projektet kommer genomföras med följande avgränsningar:

- **Fördefinierade scenarion:** projektets beställare tillhandahåller ett färdigt urval av vanligt förekommande scenarion som är framtagna för att öva maritim kommunikation.
- **Språkstöd:** eftersom maritim kommunikation sker på engelska, så utformas applikationen med ett engelskspråkigt användargränssnitt.

2
Metod

Detta kapitel beskriver arbetets tillvägagångssätt. Först beskrivs valet av att ta fram en grafisk prototyp, följt av val av teknikstack för implementation. Därefter beskrivs det konstruktionsflöde som legat till grund för implementationen, samt hur användartestning planeras att utföras.

2.1 Grafisk prototyp

Som första steg planeras en prototyp av användargränssnittet att tas fram i designplattformen Figma. Prototypen fungerar som en tidig version av webbapplikationen där den utseendemässiga designen fastställs. Detta syftar till att förenkla processen med implementation av användargränssnittet, då designmässiga förändringar i ett senare skede riskerar att bli tidskrävande att genomföra.

2.2 Teknikstack

För att utveckla applikationen har en teknikstack valts med fokus på modern webbutveckling, tydlig separation mellan frontend och backend samt god utvecklarupplevelse. I följande avsnitt beskrivs de centrala ramverk och verktyg som kommer utgöra grunden för utvecklingsarbetet.

2.2.1 React

Eftersom applikationen är tänkt att vara interaktiv kommer ett komponentbaserat UI-bibliotek användas. Biblioteket som valts är React, ett modernt bibliotek baserat på webbstandarder som HTML, CSS och JavaScript, vilket möjliggör effektiv hantering av dynamiskt innehåll[3].

React tillhandahåller en komponentbaserad utvecklingsmodell utformad för att främja inkapsling och återanvändning av kod. Användargränssnittet delas upp i mindre, självständiga komponenter som enkelt kan återanvändas, vilket gör kodbasen lättare att överblicka och underlättar testning.

2.2.2 TypeScript

TypeScript kommer användas för att förenkla utvecklingsarbete. TypeScript är ett programmeringsspråk som bygger vidare på JavaScript med, bland annat, statisk

typkontroll. TypeScript konverteras till JavaScript som sedan kan köras i webbläsaren [4]. Med typkontroll kan fel som annars är svåra att upptäcka med JavaScript därmed identifieras direkt i utvecklingsmiljön och åtgärdas.

2.2.3 Styling och UI-komponenter

För att förenkla stylingprocessen kommer CSS-ramverket TailwindCSS användas. Det är ett utility-first-ramverk där styling tillämpas genom färdigdefinierade hjälpklasser som kan skrivas direkt i komponenternas markup, istället för i separata CSS-klasser [5]. Detta effektiviserar stylingarbetet genom att göra koden mer överskådlig, lättarbetad och enkel att underhålla.

Utöver Tailwind CSS används shadcn, en samling förbyggda och anpassningsbara UI-komponenter baserade på Tailwind CSS. Komponenterna möjliggör ett enhetligt användargränssnitt, som är enkelt att anpassa utifrån olika designval [6].

2.2.4 Express.js

För backend kommer Express.js användas, ett ramverk för Node.js som möjliggör enkel uppsättning av webbservrar och API:er [7]. Applikationens backend kommer exponera ett REST API som frontenden kommunicerar med via HTTP-anrop för att hämta och skicka data. Express.js kommer även hantera sessionshantering, vilket möjliggör att användartillstånd bevaras mellan förfrågningar utan att klienten behöver skicka autentiseringsinformation vid varje anrop.

2.3 Konstruktionsflöde

Nedan listas det logiska flödet som ska ligga till grund för hur applikationen konstrueras.

- Användare går in på chatboten och väljer ett scenario.
- Scenariot börjar genom att användaren får nödvändig information samt om chatboten ska börja så skickar den ett meddelande.
- Användaren skickar ett meddelande, antingen genom text- eller röstinmatning.
  - AI-modell kontrollerar stavning mot engelska med info om SMCP och namn i scenariot.
  - Algoritmer kategoriserar delar av meddelandet.
    - Namn på fartyg samt VTS
    - Identifiera *Phonetic Alphabet and Figures*
    - Identifiera *Action Verbs* (ex proceed, report, contact)
    - Identifiera *Message Markers* (ex question, request)
    - Identifiera *Ambiguous Words* (ex may, might, should, can)
    - Identifiera *Procedure Words* (ex over, out, repeat, correction)

  - Algoritmer och LLM kontrollerar meddelandets innehåll.
    * Rätt ordning på kategoriserade delar
    * Bedömning om ordantalet är rimligt
    * Korrekta ordval och begrepp används
    * Identifierar om något saknas i meddelandet
    * Vid röstinmatning undersöks pauser och tydlighet
- Chatboten skickar ett svar till användaren.
  - Går vidare till nästa del av pågående scenario.
  - Ber användaren att förtydliga eller skriva om svar (baserat på existerande fel).
- Fortsätter tills scenariot är avklarat.
- Efter avklarat scenario skickas konstruktiv kritik om återkommande fel.

2.4 Användartestning

För att säkerställa att den färdiga produkten uppfyller uppsatta mål, kommer användartestning vara central. Testning av applikationen planeras att genomföras med hjälp av studenter och lärare på Chalmers sjöfartshögskola. Ett utvärderingsformulär kommer tas fram som speglar projektets mål. Efter genomförd testning får användaren bedöma hur väl applikationen uppfyller målen genom att fylla i utvärderingsformuläret.

3
Teori

Denna del kommer innehålla relevant information om maritim kommunikation. Detta för att tillhandahålla en grundläggande förståelse om dess struktur och innehåll.

3.1 Terminologi inom maritim kommunikation

Maritim kommunikation behöver vara precis, enkel och otvetydig för att undvika förvirring och feltolkningar [8, s. 4]. Detta medför behovet av ett strukturerat språk och användningen av specifik termer vid denna sorts kommunikation.

3.1.1 Phonetic Alphabet

Vid bokstavering skall ett särskilt fonetiskt alfabet användas [8, s. 14]. Detta för att säkerställa att korrekt information uppfattas och det gäller för både bokstäver samt siffror. Tal sägs en siffra i taget [8, s. 17]. Maritim använder standarden *NATO Phonetic Alphabet* för bokstaveringar.

3.1.2 Message Markers

Message Markers är särskilda ord som bör användas för att förmedla information. Följande är en lista av Message Markers som bör användas. [8, s. 15].

- Instruction
- Advice
- Warning
- Information
- Question
- Answer
- Request
- Intention

Under förmedlandet av information bör dessa nyttjas för att öka sannolikheten att meddelandet förstås korrekt [8, s. 46]. Genom att använda dessa innan själva innehållet påbörjas förstår mottagaren vad för sorts information som ska ges och förstår därmed även meddelandet bättre.

3.1.3 Ambiguous Words

Ambiguous Words syftar på tvetydiga ord som har olika betydelser beroende på kontexten som de används i [8, s. 18]. Följande är en lista av dessa ord.

- May
- Might
- Should
- Could
- Can

Användning av dessa tvetydiga ord riskerar att leda till missförstånd och potentiellt att olyckor inträffar [8, s. 18]. Därmed bör de undvikas vid maritim kommunikation.

3.1.4 Allmänt språkbruk

Mer än specifika termer så bör maritim kommunikation även följa vissa principer för att förbättra säkerheten [8, s. 13]. Detta inkluderar undvikandet av kontraktioner, förenklad grammatik och att minimera idiomatiska uttryck. Att även begränsa antalet använda ord är en av dessa principer.

3.2 Meddelandets struktur

Meddelanden inom maritim kommunikation bör även följa en viss struktur för att enklare förmedla information mellan två parter. Ett meddelande bör därmed ha tre delar. Först ett anrop, därefter meddelandeinnehållet och slutligen en avslutssignal [9]. Det exakta innehållet i dessa ingående delar varierar då det saknas ett konkret internationellt regelverk för detta. Chatboten kommer anpassas för upplärningen inom Chalmers Sjöfartshögskola.

3.2.1 Anrop

Anropet är till för att etablera och förmedla parterna av kommunikationen. Anropet ska vara på formen "*Mottagare* this is *Avsändare*"[9]. Detta för att säkerställa att båda parterna är säkra på att mottagna meddelanden är en del av deras konversation. I det första meddelandet som initierar kommunikationen bör dock mottagaren upprepas två till tre gånger samt avsändaren tre gånger. Att repetera namnen flera gånger underlättar för mottagaren att höra att de anropas samt vilka det är som sänder meddelandet. Därefter bör de båda namnen sägas en gång vardera under kommunikationens gång.

3.2.2 Meddelandeinnehåll

Denna del utgör förmedlingen av den information som kommuniceras i meddelandet. De regler och principer som framgick under terminologin ska följas vid förmedlandet av meddelandeinnehållet. Vid mottagen information bör även kvittens genomföras

för att säkerställa att informationen har förståtts korrekt. Detta ska göras innan ny information kommuniceras samt påbörjas med *Received*[9].

3.2.3 Avslutssignal

Ett meddelande ska avslutas med antingen *Over* eller *Out* för att förmedla att meddelandet är klart och den andra parter kan börja tala [9]. *Over* används när ett meddelande från motparten förväntas och till sist används *Out* för att stänga kommunikationen.

4

Genomförande

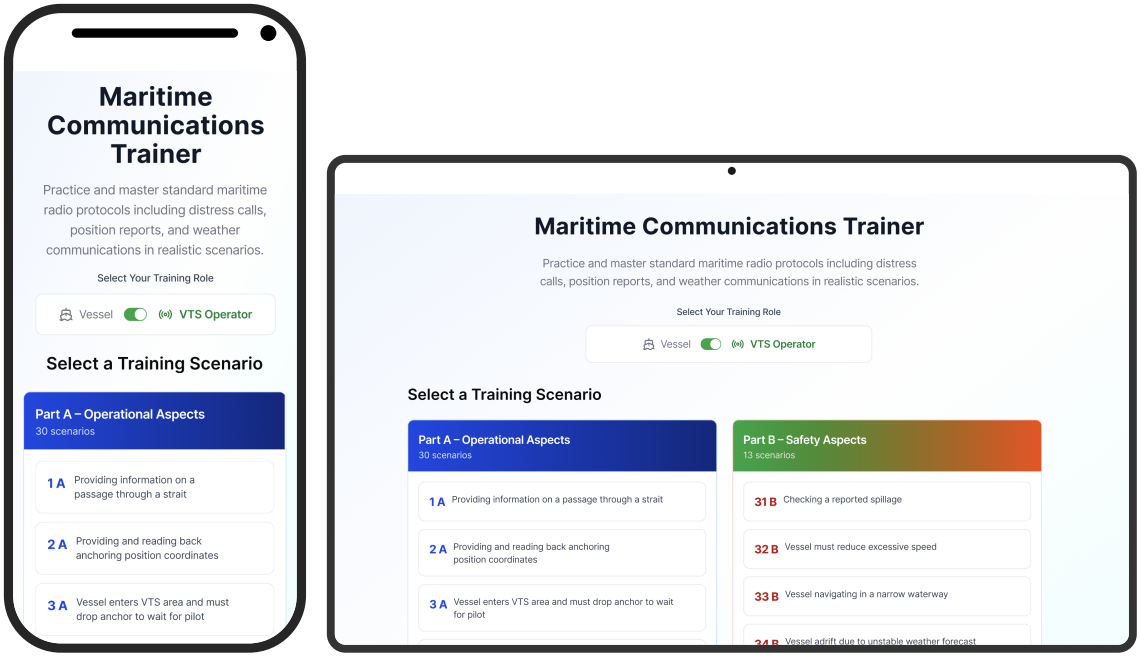
Detta kapitel redogör för skapandet av chatboten. Inledningsvis beskrivs framtagningen av den grafiska prototypen i Figma, följt av implementationen av applikationens frontend och slutligen dess backend.

4.1 Figma-prototyp

Designarbetet har utgått från att skapa ett responsivt gränssnitt anpassat för både mobila enheter och datorer. Meddelande- och feedbackinnehåll i prototypen återspeglar inte chatbotens tänkta funktionalitet, utan är snarare en visuell mall för hur applikationens gränssnitt ska utformas. Prototypen redovisas i följande delavsnitt.

4.1.1 Startvy och scenarioval

Startsidan som visas i Figur 1 utformades med syftet att ge en tydlig orientering för användaren. Överst presenteras en titel och en kort beskrivning av applikationen följt av en toggle-knapp där användaren kan välja rollen som antingen *Vessel* (fartyg) eller *VTS Operator*. Valet av roll styr sedan vilket kommunikationsroll som används under övningen. Toggle-knappen placerades ovanför listan av valbara scenarier för att framhäva och uppmana användaren till att välja roll som första steg i användarflödet.



(a) Mobilvy **(b)** Datorvy

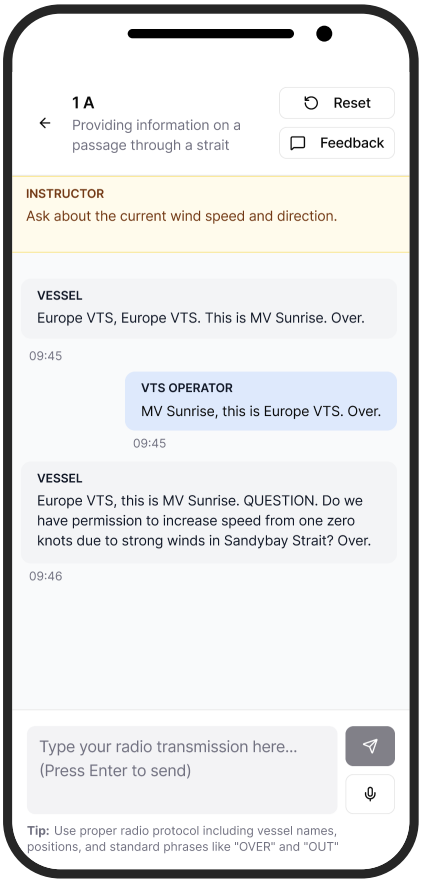
Figur 1: Prototyp av startsida för mobil- och datoranvändare

Designen stödjer tilläggande av moduler för att strukturera scenarier i tematiska block. Prototypen är utformad efter denna princip och fylldes med tillfällig data för att visualisera detta.

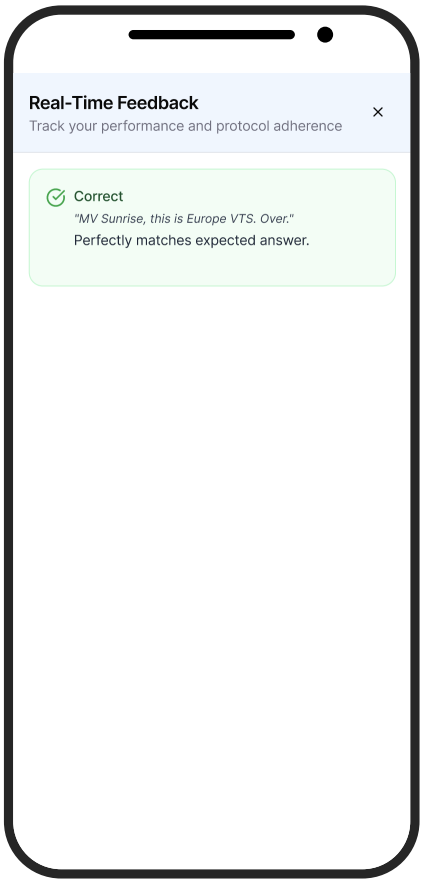
Eftersom layouten är responsivt designad anpassas startsidan efter enhetens skärmstorlek. På datorvyn arrangeras scenarioblocken i ett rutnät som nyttjar den tillgängliga bredden. På mobila enheter arrangeras scenarioblocken istället vertikalt för att förenkla läsbarhet och främja touch-baserade interaktioner.

4.1.2 Chattgränssnitt

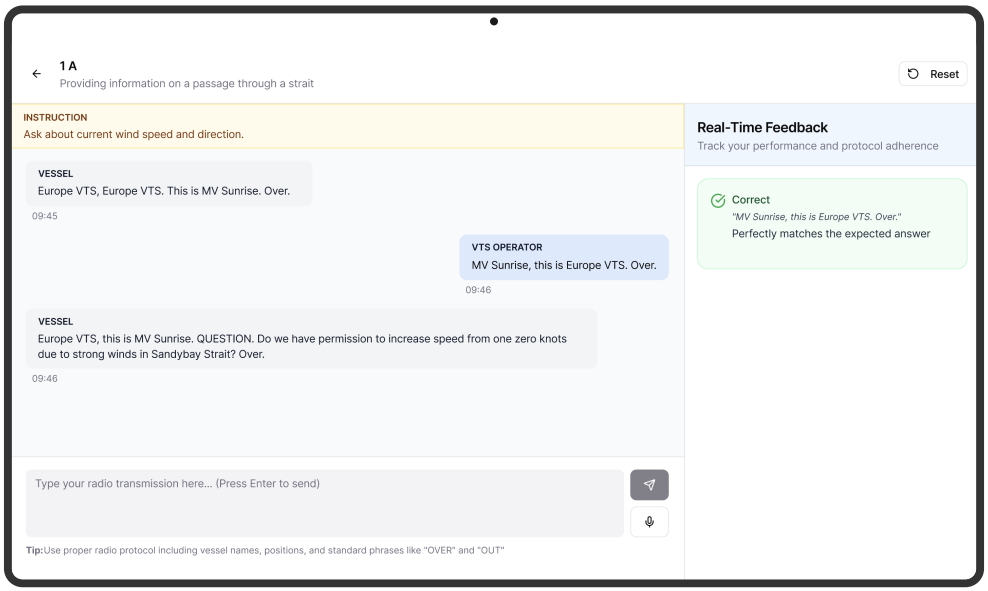
Chattgränssnittet som framgår i Figur 2a och 2c baseras på ett chattflöde där meddelanden visas i pratbubblor, varierade med färg, namn och placering beroende på avsändare och användarens roll. Varje meddelande kompletteras med en tidsstämpel undertill.



(a) Mobilvy för chatt



(b) Mobilvy för samling av genererad feedback



(c) Datorvy med kombinerad sida för chatt samt genererad feedback

Figur 2: Prototyp av chattgränssnitt för mobil- och datoranvändare

Ovanför chattgränssnittet finns en instruktionspanel utformad för att ge kontextuell vägledning till användaren i realtid. Panelen informerar användaren om vilket kommunikationssteg som förväntas härnäst samt nödvändig information och uppdateras i takt med att scenariot fortskrider.

Längre ner i vyn finns ett inmatningsfält som stöder både textbaserad och röstbaserad inmatning, där en mikrofonikon möjliggör röstinmatning av meddelandet. Pappersplans-symbolen används sedan för att skicka användarens meddelande och således fortskrider scenariot. Under inmatningsfältet visas en påminnelse *Tip* i syfte att kontinuerligt förstärka korrekt kommunikationsbeteende.

4.1.3 Realtidsfeedback

I anslutning till chattgränssnittet integrerades en feedbackmodul som ger användaren omedelbar återkoppling efter varje meddelande. I modulen bekräftas huruvida meddelandet var korrekt och vid behov ges konstruktiv feedback på förbättringsområden. Designen bygger på att återkopplingen sker kontinuerligt under övningen snarare än samlad feedback på slutet.

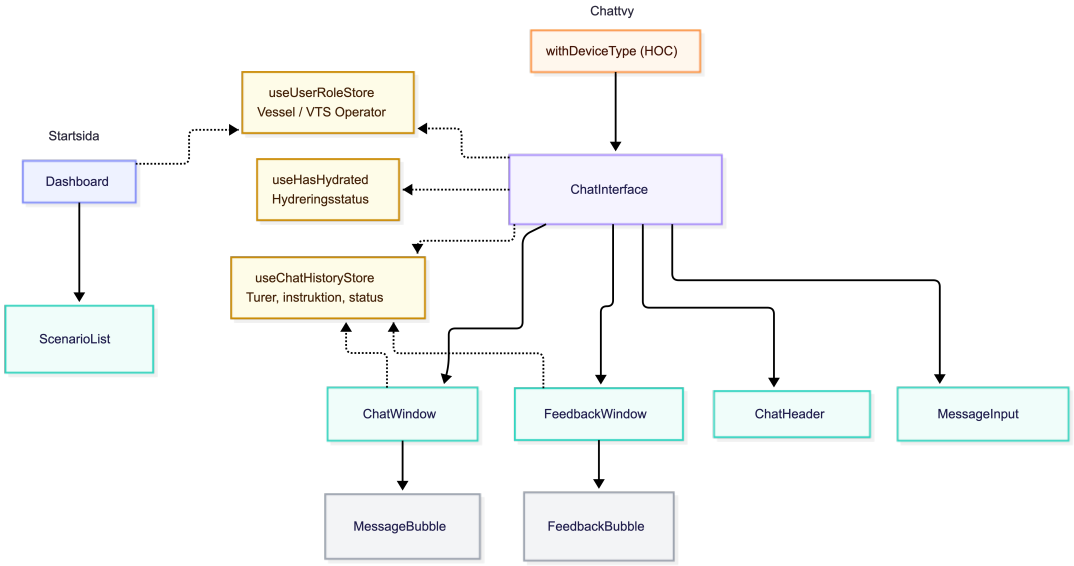
Feedbackmodulens synlighet anpassas beroende på enhet, vilket framgår i Figur 2b och 2c. I datorvyn placeras feedbackpanelen som ett fast sidoelement till höger om chattflödet, vilket möjliggör en omedelbar överblick utan att användaren behöver navigera bort från den aktiva konversationen. På mobila enheter, där skärmytan är begränsad, döljs feedbackelementet. Vid tryck på feedback-knappen öppnas det som en överlagd vy (*overlay*).

4.2 Frontend-utveckling

I följande avsnitt beskrivs hur frontendens komponenter är utformade, tillämpning av responsivitet, hur data och tillstånd hanteras, användning av animeringar och lösning för röstinmatning.

4.2.1 Komponentöversikt

Under utvecklingen av frontend har ett flertal komponenter tagits fram tillsammans med anpassade Shadcn-komponenter. Samtliga komponenter visas i Figur 3 och beskrivs mer ingående nedan.



Figur 3: Översikt över frontend-komponenter, stores och HOC (*Higher-order component*)

4.2.1.1 Startsida

- **Dashboard**: Huvudkomponenten i applikationens startsida. Härifrån görs ett API-anrop för att hämta samtliga scenarier och dess tillhörande beskrivningar. Komponenten ansvarar även för att skriva ut statisk text såsom titel och beskrivning av applikationen. Utöver statisk text har den även en toggle-switch från Shadcn som låter användaren välja träningsroll samt tillgängliga listor av scenarier i form av `ScenarioList`-komponenter. Vid klick på ett tillgängligt scenario navigeras användaren vidare till Chattvyn via React Router.
- **ScenarioList**: Renderar en enskild scenariolista bestående av en rubriksektion samt en scrollbar lista av tillgängliga scenarier. Listans färgtema styrs via ett `headerColor`-prop som kopplas till fördefinierade gradient- och hover-stilar i sju färgvarianter. Komponenten har även en `onClick`-prop som används vid klick på ett scenario i listan.

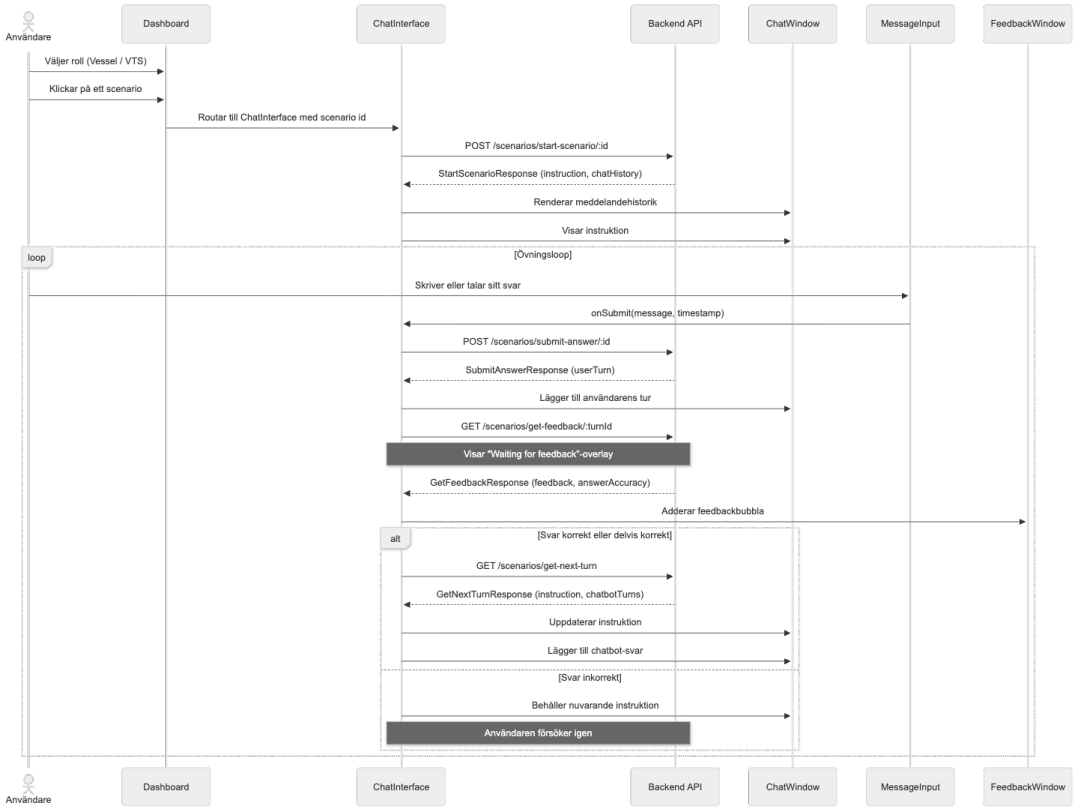
4.2.1.2 Chattvy

- **ChatInterface**: Huvudkomponent för chattvyn. Hanterar övningsscenariot från start till slut. När komponenten laddas in anropas en endpoint för att starta scenariot (samma endpoint används för att återuppta scenariot vid omladdning av webbsidan). Om användaren väljer att starta om ett scenario anropas tillhörande endpoint. Dessutom koordineras tre steg av API-anrop i samband med att användaren skickar ett meddelande. Först skickas meddelandet, sedan hämtas feedback och slutligen hämtas nästa chatbot-tur.
- **ChatHeader**: Sidhuvud med navigering och scenarioåtgärder. Innehåller en tillbaka-knapp till startsidan, scenariobeskrivning i mitten samt en *Retry*-knapp som anropar en `onRetry`-prop för att hantera omstart av scenario. På

mobila enheter visas även en *Feedback*-knapp som öppnar feedbackpanelen som en `Drawer`-komponent.

- **ChatWindow**: Renderar konversationens meddelandeflöde och en dynamisk instruktionspanel. Panelen visas i ett gult amber-tema under pågående scenario och växlar till rött med texten *Scenario ended* när scenariot avslutas. Meddelandelistan scrollar automatiskt till det senaste meddelandet.
- **MessageBubble**: Komponent för enskilt chattmeddelande. Användarmeddelanden visas som blå bubblor i högra delen av meddelandeflödet, medan chatbot-meddelanden är grå bubblor i vänstra delen av flödet. Ovanför respektive bubbla visas en avsändaretikett (*Vessel* eller *VTS Operator*) och under visas en tidsstämpel.
- **MessageInput**: Hanterar användarens meddelande-inmatning med stöd för både text och röst (via Web Speech API). Vid klick på skicka-knappen används en `onSubmit`-prop för att hantera skickande av meddelandet.
- **FeedbackWindow**: Panel som listar feedback för samtliga användarmeddelanden under det aktuella scenariot.
- **FeedbackBubble**: Komponent för visualisering av enskilt användarsvar med tillhörande feedback i form av en meddelandebubbla. Bubblorna är färgkodade utifrån svarets korrekthet, där grönt är för *Correct*, gult är för *Partially Correct* och rött är för *Incorrect*. Inuti bubblar visas användarsvaret i kursiv stil följt av feedbacktexten och en tidsstämpel.

Sekvensdiagrammet i Figur 4 visar hur komponenterna samverkar med varandra och med applikationens backend API vid en vanlig användarsekvens. Diagrammet visar användarflödet från att användaren väljer roll och scenario tills dess att scenariot är avslutat. När användaren skickar ett meddelande sker det tre sekventiella HTTP-anrop från huvudkomponenten `ChatInterface`. Först för att skicka in användarens meddelande, sen för att hämta feedback och avslutningsvis för att hämta nästa tur i scenariot.



Figur 4: Sekvensdiagram på användarflödet för frontend-komponenter

4.2.2 Responsivitet

Eftersom applikationen är utformad för att fungera på både dator och mobilenheter har en HOC `withDeviceType` tagits fram för att hantera anpassning beroende på enhet. En HOC är en funktion som tar en komponent som argument och returnerar en ny komponent [10]. `withDeviceType` läsar av webbläsarens bredd vid montering och skickar en `isMobile`-prop till `ChatInterface` som sedan styr vilken layout som ska renderas. `isMobile` sätts till *true* om bredden är mindre än 768 pixlar och till *false* annars.

En mer omfattande layoutskillnad gäller feedbackpanelen. I datorvyn renderas `FeedbackWindow` som en fast sidopanel till höger om chattflödet, vilket ger användaren en parallell överblick över feedback utan att behöva navigera bort från konversationen. På mobil ersätts sidopanelen av en `Drawer`-komponent från Shadcn och aktiveras via *Feedback*-knappen i headern. Knappen döljs i datorvyn med Tailwinds `md:hidden`-klass, vilket betyder att den responsiva layouten hanteras på CSS-nivå parallellt med den komponentbaserade logiken i `withDeviceType`.

4.2.3 Tillståndshantering

För att förenkla tillståndshantering och datalagring användes Zustand, ett state management-bibliotek för React. Genom Zustand kan man skapa globala tillstånd som lagras i webbläsarens cacheminne med hjälp av React hooken *useStore* [11]. Tre

separata datamoduler användes för olika ansvarsområden.

`useUserRoleStore` lagrar användarens valda träningsroll (*Vessel* eller *VTS Operator*) och en `switchUserRole`-funktion för att växla mellan rollerna. Eftersom rollvalet påverkar både startsidan och chattvyn används denna datamodul i både `Dashboard` och `ChatInterface`.

`useChatHistoryStore` är applikationens mest centrala datamodul och lagrar fullständig konversationshistorik i form av samtliga meddelanden med tillhörande feedback, aktuell instruktionstext och en flagga för om scenariot är avslutat. Datamodulen används av `ChatInterface`, `ChatWindow` och `FeedbackWindow`, vilket möjliggör att dessa komponenter alltid använder samma underliggande konversationsdata utan att props behöver passera genom komponentträdet.

`useHasHydrated` löser ett specifikt problem med *server-side rendering*. Eftersom Zustand-datamodulen lagras i webbläsarens cacheminne kan ett hydreringsfel uppstå om komponenter renderas på servern innan klientens lagrade tillstånd hunnit läsas in. Modulen har en boolesk flagga som sätts till *true* först när klientens tillstånd är inläst och komponenter som `ChatInterface` och `Dashboard` returnerar *null* tills dess.

4.2.4 Web Speech API

Röstinmatning till applikationen implementeras i `MessageInput` via webbläsarens inbyggda Web Speech API. Genom API:et används gränssnittet `SpeechRecognition` för taligenkänning [12]. Vid interaktion med mikrofonikonen skapas en ny instans av `SpeechRecognition` (eller `webkitSpeechRecognition` för Safari) konfigurerad för engelska (`en-US`)[13].

Eftersom Web Speech API inte stöds i samtliga webbläsare visas ett felmeddelande via `toast` om API:et saknas. Under aktiv röstinspelning inaktiveras inmatningsfältet och skicka-knappen och en animerad ljudvågsvisualisering visas som overlay.

Ett problem som upptäcktes under utveckling var att transkribering av siffror alltid tolkas som numeriska tecken och att vissa termer ofta feltolkas. För att motverka detta tillämpades en `correctSpeechResult`-funktion på transkriptionsresultatet innan det placeras i inmatningsfältet. Funktionen konverterar numeriska tecken till motsvarande ordform (exempelvis *3* till *three*) och vanliga feltranskriptioner som *BTS* och *MB* till de korrekta maritima förkortningarna *VTS* respektive *MV (Motor Vessel*.

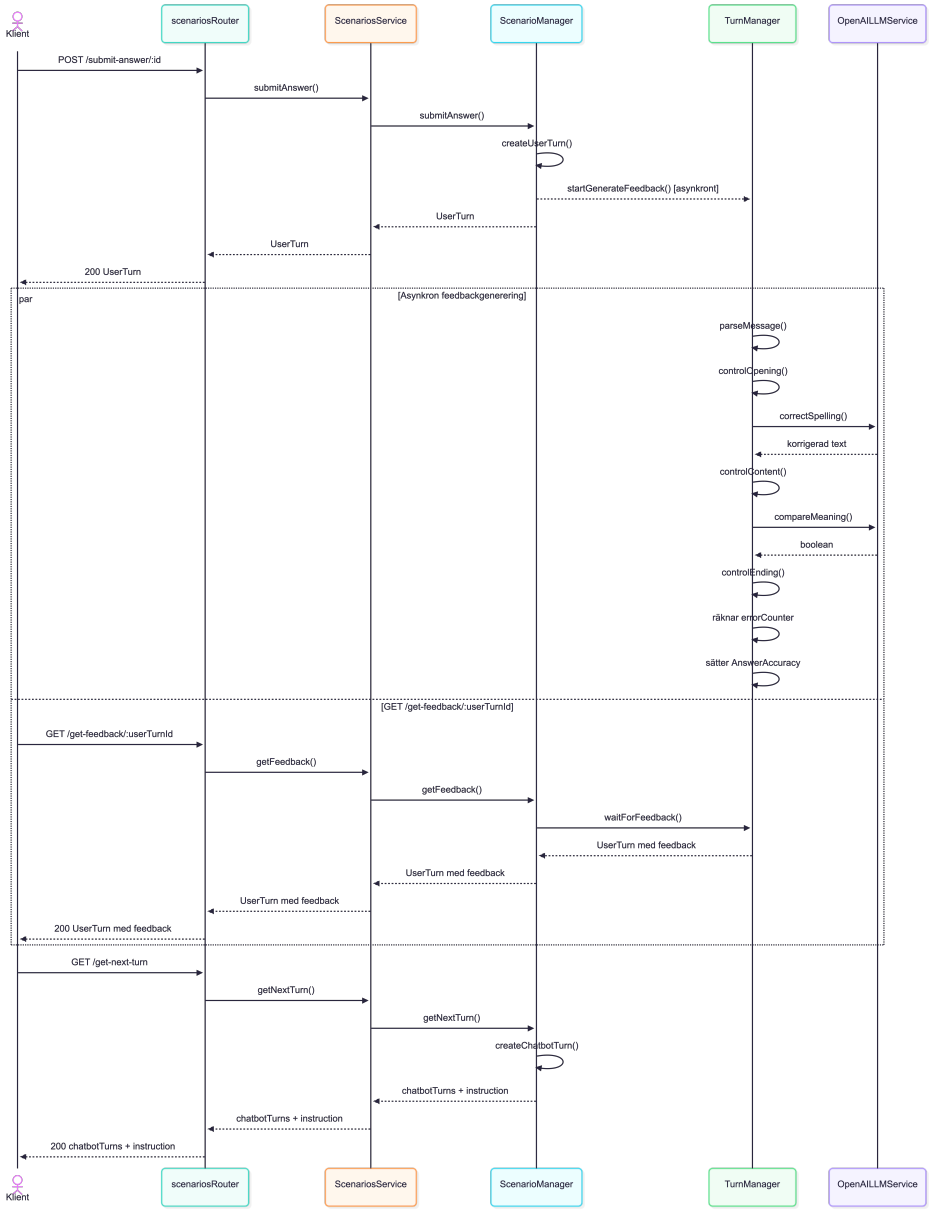
4.2.5 Animeringar

Animeringar hanteras genomgående med Motion, ett animationsbibliotek för React [14]. Biblioteket nyttjas på flera ställen i applikationen för att ge mjukare visuell upplevelse på tillståndsförändringar. Exempelvis används de när nya meddelanden

och feedback animeras in i chattflödet, när instruktionspanelen växlar mellan steg samt när overlays tonas in och ut.

4.3 Backend-utveckling och logik

Backend är implementerad som ett REST-API med Express.js och hanterar all tillståndshantering, scenarielogik och feedbackgenerering. Eftersom att användardata inte behöver vara persistent över tid lagras allt tillstånd *in-memory*. Figur 5 visar det fullständiga flödet vid sekvensen där användaren skickar sitt svar. Flödet är centralt för att förstå hur de olika klasserna interagerar med varandra och kommer förklaras detaljerat i detta avsnitt.

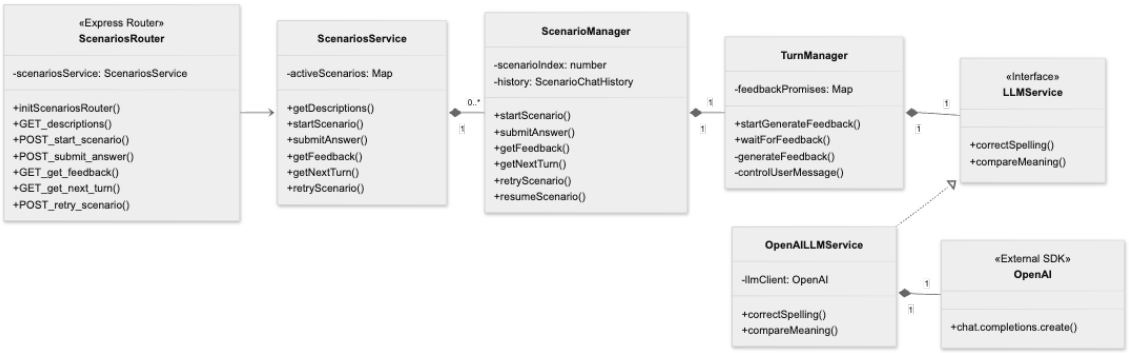


Figur 5: Sekvensdiagram på hanteringen av ett meddelande

4.3.1 Arkitektur och ansvarsfördelning

Backend är uppbyggd kring en arkitektur där varje klass har ett tydligt avgränsat ansvarsområde. Den övergripande klasstrukturen ses i Figur 6. Flödet mellan klasserna följer kedjar `ScenariosRouter` → `ScenariosService` → `ScenarioManager` → `TurnManager` → `OpenAILLMService` där varje klass i kedjan delegerar ansvaret nedåt. Ett `LLMService` interface används för att, vid behov, enkelt kunna byta ut det nuvarande lagret som är knutet till en OpenAI-modell.

`ScenariosRouter` hanterar inkommande HTTP-förfrågningar och vidarebefordrar dem till `ScenariosService`. `ScenariosService` utgör det lager som ansvarar för samtliga aktiva scenarier via en `Map` som kopplar varje användarsession mot en egen instans av `ScenarioManager`. `ScenarioManager` i sin tur ansvarar för tillståndshantering per användare. Detta inkluderar konversationshistorik samt turordning. Analys av användarmeddelande och feedbackgenerering delegeras vidare till `TurnManager`. `TurnManager` delegerar sedan, vid behov, vidare ansvar för AI-baserad stavningskontroll och betydelsejämförelse till `OpenAILLMService`. En fullständig översikt över klasserna på backend återfinns i Appendix A Figur A.1.



Figur 6: Övergripande arkitektur av backend.

4.3.2 Middlewares

Två mellanprogramvara (middlewares) har implementerats för att hantera autentisering samt felhanteringslogik. Båda dessa middlewares används för att minimera kod-duplicering.

Till att börja används `requireUser` för att validera om en aktiv användarsession existerar innan samtliga routes kan nås. Om ingen session existerar returneras direkt ett svar med statuskod 400, utan att förfrågan når servicelagret.

Felhanteringen hanterades genom `ErrorHandler`, som fångar upp alla fel som kastas. Fel av typen `HttpError`(egen typ som ärver den generiska typen Error) fångas upp och konverteras till ett Http-respons med tillhörande statuskod och felmeddelande, medan övriga fel resulterar i ett respons med statuskod 500. I samtliga lager av applikationens backend kan `HttpError` användas och kastas. Detta leder till att route-funktionerna inte behöver innehålla egen felhanteringslogik och gör koden

lättöverskådlig.

4.3.3 Scenariehantering

Datamodellen för scenarier är utformad så att användaren kan välja mellan de olika rollerna i scenariot. Dessutom tillhandahålls nödvändig information genom de instruktioner som ingår i scenariot. Ett scenario innehåller ett id, en beskrivning som visas för användaren, de olika rollerna med namn, samt fördefinierade instruktioner och meddelanden för båda roller. Strukturen för ett scenario och dess JSON-schema framgår i Figur 7.

```
{
  "id": "string",
  "name": "string",
  "participants": {
    "starter":   { "role": "vessel|vts", "name": "string" },
    "responder": { "role": "vessel|vts", "name": "string" }
  },
  "turns": [
    {
      "vessel_instruction": "string",
      "vessel_message":     "string",
      "vts_instruction":    "string",
      "vts_message":        "string"
    },
    ...
  ]
}
```

Figur 7: JSON-schema för ett scenario

Initieringen av ett scenario samt dess tillstånd hanteras i klassen `ScenarioManager`. Detta görs med scenariots id samt användarens valda roll, som visas i Tabell 1. Tillståndet hanteras med hjälp av ett index som inkrementeras i samband med att användarens svar accepteras för att gå vidare till scenariots nästa del. Med användarens roll bestäms även chatbotens och från scenariot framgår det huruvida användaren eller chatboten skall påbörja kommunikationen.

Chatbotens tur består av att dess meddelande skickas till klienten varefter användarens tur påbörjas. Den inleds med att instruktionen för meddelandet skickas till klienten och visas för användaren. Ett mottaget meddelande från användaren behandlas i `TurnManager` som genererar feedback samt bedömer meddelandets korrekthet som sedan visas i klienten. Om meddelandet bedöms som inkorrekt kommer `ScenarioManager` invänta ett nytt meddelande från användaren som behandlas likt det föregående. Vid accepterat svar går turen återigen över till chatboten.

4.3.4 Meddelandehantering

`TurnManager` har en metod för att påbörja generering av feedback som anropas när ett meddelande från användaren mottas. Detta sker med ett `Promise` som tillhandahåller att genereringen exekveras asynkront och metoden inväntar därmed inte att feedbacken ska bli klar. Genom att spara `Promise` i en `Map` kan applikationen fortsätta genomföra övriga uppgifter för att sedan invänta eller hämta feedbacken vid behov.

Vid potentiellt fel under genereringen av feedback kommer det fångas upp och radera det `Promise` som sparats. Dessutom kommer metoden som hämtar feedbacken skicka ett reserv-svar till klienten för att informera användaren om att ett fel har inträffat och att försöka igen. Det kontrolleras även om ett `Promise` redan finns för ett meddelande innan generering påbörjas som säkerställer att applikationen inte skapar duplicerad feedback för samma meddelande.

4.3.5 Generering av feedback

För att generera feedback används användarens meddelande samt det korrekta svaret som finns i scenariots datamodell. Inledningsvis normaliseras båda meddelandena för att säkerställa att de är på samma format och inte innehåller oväntade tecken. Dessutom sätts en boolesk flagga för att bestämma huruvida meddelandets syfte är att initiera kommunikationen mellan parterna. Denna bestäms genom att undersöka det korrekta svaret eftersom ett meddelande som inleder kommunikationen förväntas vara på ett annat format än övriga meddelanden. Vid behov kontrolleras stavningen i meddelandet för att minimera risken att en felstavningar påverkar feedbackens utfall.

4.3.5.1 Uppdelning av meddelandet

Därefter görs båda meddelandena om till objekt där deras anrop, meddelandeinnehåll, samt avslut har bestämts och delats upp. Anropet bestäms med hjälp av namnen på mottagaren samt avsändaren och nyttjar den ovan nämnda flaggan. Det görs genom att undersöka namnens positioner i meddelandet där anropet bör vara fram tills båda namnen har nämnts. En kontroll genomförs även med att undersöka var meddelandets första Message Marker är. Detta för att vakta mot att ett namn saknas i själva anropet men används senare i meddelandet. Om en Message Marker förekommer tidigare än ett namn, kommer positionen av det namnet att bortses ifrån.

Om flaggen däremot är sann så betraktas anropet fortsätta tills sista gången någon av namnen nämns. Detta för att ett meddelande som initierar kommunikationen ska enbart bestå av anrop samt avslut. Ett meddelandes avslut bedöms vara när det slutar med *Over and out*, *Over* eller *Out*. Meddelandeinnehållet sätts till meddelandets eventuellt återstående del efter att anropet samt avslutat har bestämts.

4.3.5.2 Felräknare till feedbacken

Härnäst kontrolleras meddelandets ingående delar och feedback genereras på dessa. Kontrolleringen av anropet samt innehållet returnerar en lista av kommenterar i form av strängar samt en felräknare. För avslutet returneras istället enbart en sträng samt en flagga som visar om avslutet är korrekt eller inte. Alla strängar läggs ihop till en gemensam sträng med radbyten för att kunna visas tydligare för användaren.

Dessutom adderas felräknarna ihop för att jämföras med en konstant, i nuläget är denna konstant fyra. Detta för att bedöma huruvida en användares meddelande ska accepteras. Om det inte finns några fel är meddelandet *Correct*, är felräknaren på intervallet från ett till tre (konstanten minus ett) är meddelandet *Partially Correct* och om den är högre eller lika med konstanten bedöms meddelandet vara *Incorrect*.

4.3.5.3 Feedback på anrop

Kontrollen av anropet nyttjar flaggan för om kommunikationen initieras. Anropet av ett meddelande anses vara korrekt om det är på formen "*Mottagare* this is *Avsändare*". Om kommunikationen initieras förväntas mottagaren nämnas två eller tre gånger samt att avsändaren nämns tre gånger. Felräknaren till kontrollen av anropet är begränsad till att inte överstiga två.

Följande kontroller genomförs på anropet och feedback läggs till vid avvikelser.

- Att både mottagare samt avsändare nämns.
- Att mottagaren nämns innan avsändaren.
- Antalet gånger som namnen förekommer.
- Att *this is* är med om båda namnen förekommer.
- Att anropet inte innehåller fler ord än vad den förväntas göra.

4.3.5.4 Feedback på meddelandeinnehåll

Bedömningen av innehållet i användarens meddelande baseras främst på jämförelser med det korrekta svaret. Dessutom undersöks hur många och vilka ord som används för båda texterna. Metoden som genomför detta anropas om antingen en användares meddelande eller svaret har ett innehåll.

Följande detaljer undersöks.

- Att om svaret har ett innehåll så har även meddelandet ett innehåll.
- Att innehållens betydelser är snarlika och det jämförs med hjälp av en AI-modell som returnerar sant eller falskt.
- Att svarets första Message Marker också förekommer i meddelandet.
- Att meddelandet inte innehåller Ambiguous Words som inte förekommer i svaret.
- Att meddelandet inte saknar Phonetic Alphabet som förekommer i svaret.

- Att meddelandet inte innehåller betydligt fler ord än svaret.

Felräknaren till feedbacken på innehållet ökas med ett för varje avvikelse ovan med undantag för jämförelsen av betydelse samt Phonetic Alphabet. Dessa fel medför att meddelandet kommer bedömas som inkorrekt och behöver därmed alltid skrivas om av användaren.

4.3.5.5 Feedback på avslutssignal

Utformningen av feedbacken för meddelandets avslut skiljer sig från den som används för de tidigare delarna. Detta till följd av att avslutet är enklare att undersöka samt att färre fel kan begås. Vid denna kontroll jämförs avslutssignalen från användaren med det korrekta svarets och om de är olika så bedöms avslutet vara inkorrekt och ökar felräknaren med två. Notera att *Over and out* alltid bedöms som inkorrekt.

4.3.6 AI-integration

Applikationens `TurnManager` nyttjar `OpenAILLMService` för att utföra stavningskontroll och betydelsejämförelse med hjälp av modellen *gpt-5.4-nano*.

Stavningskontrollen genomförs på användarmeddelandet när något av villkoren nedan är uppfyllt. På så sätt kan LLM-anrop undvikas om de kan bedömas utan hjälp av AI.

- Anropet är inkorrekt.
- Avslutet är inkorrekt.
- Användarens meddelande och det förväntade svaret har båda ett innehåll som då ska jämföras.

Betydelsejämförelsen används för att avgöra om meddelandeinnehållet är betydelsemässigt ekvivalent med det förväntade svaret och anropas enbart när både användarens svar och det fördefinierade svaret har ett innehåll.

Varje LLM-anrop använder ett `zodResponseFormat` för att garantera att modellen returnerar ett konsekvent JSON-format och således undviks oväntade fel, svårtolkade svar samt *prompt injections*. Modellens beteende styrs av följande systempromptar nedan.

Systemprompt för stavningskontroll:

```
 You correct spelling in maritime communication messages.
Rules:
- Fix only obvious spelling mistakes.
- Correct NATO phonetic alphabet words if slightly misspelled.
- If phonetic alphabet words are spelled correctly, do not change
them.
- Do not modify call signs, coordinates, or numbers.
- Return only the original text with corrected spelling.
- Do not add any comments or notes.
```

```
- Do not alter the meaning of the message.
```

Systemprompt för betydelsejämförelse:

```
 You compare a maritime communication message to determine if
it is similar and return only true or false.
Rules:
- Do not return any comments or notes.
- If the messages have similar meaning, return true.
- If the messages have different meaning, return false.
Compare the message to: förväntat meddelandeinnehåll
```

4.3.7 API Endpoints

Applikationens HTTP-endpoints exponeras via Express-routern `ScenariosRouter`. Routern ansvara enbart för att hantera förfrågningar och skicka dessa vidare till `ScenariosService`. All affärslogik sköts i underliggande lager. I Tabell 1 listas samtliga HTTP-endpoints.

Tabell 1: HTTP-endpoints för scenarios-API

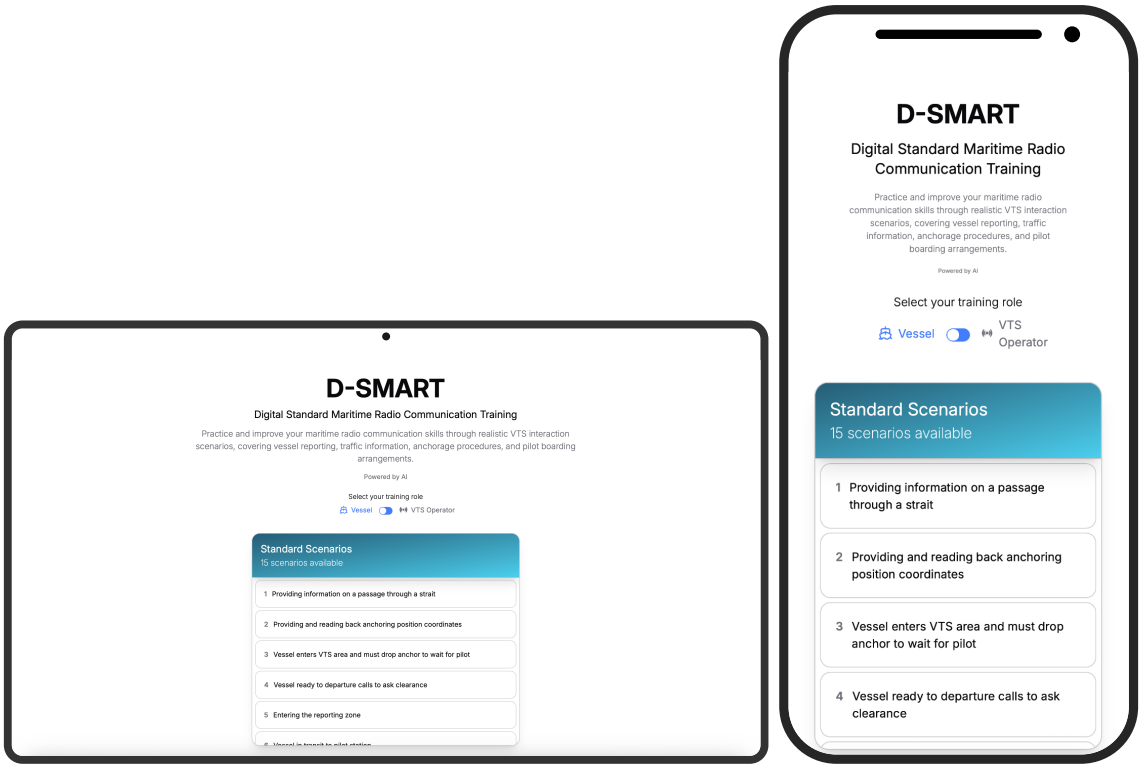
Metod	**Endpoint**	**Beskrivning**
GET	/scenarios/descriptions	Hämtar beskrivningar för alla tillgängliga scenarier
POST	/scenarios/start-scenario/:id	Startar ett scenario med givet ID och användarroll. Om användaren redan påbörjat scenariot återupptas det
POST	/scenarios/submit-answer/:id	Skickar in ett svar för aktuell tur i scenariot. Returnerar användar-turen utan feedback.
GET	/scenarios/get-feedback/:userTurnId	Hämtar feedback för en specifik användarturs ID. Returnerar användar-turen med feedback när den är redo
GET	/scenarios/get-next-turn	Hämtar nästa chatbot-tur och instruktion baserat på scenariots aktuella tillstånd
POST	/scenarios/retry-scenario	Återstartar nuvarande scenario från början och returnerar initial chatthistorik

5
Resultat

Detta avsnitt kommer hantera hur applikationen fungerar samt sammanställning av inkomna svar på enkäter för utvärdering.

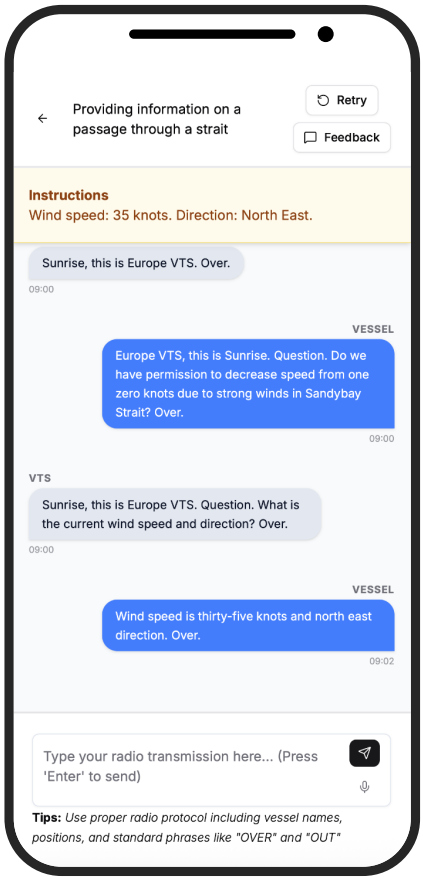
5.1 Gränssnitt

Det färdiga gränssnittet består av två huvudvyer: startsidan och chattvyn likt prototypen som redovisades i avsnitt 4.1. Figurerna 8 och 9 visar det implementerade systemet på både dator och mobil.

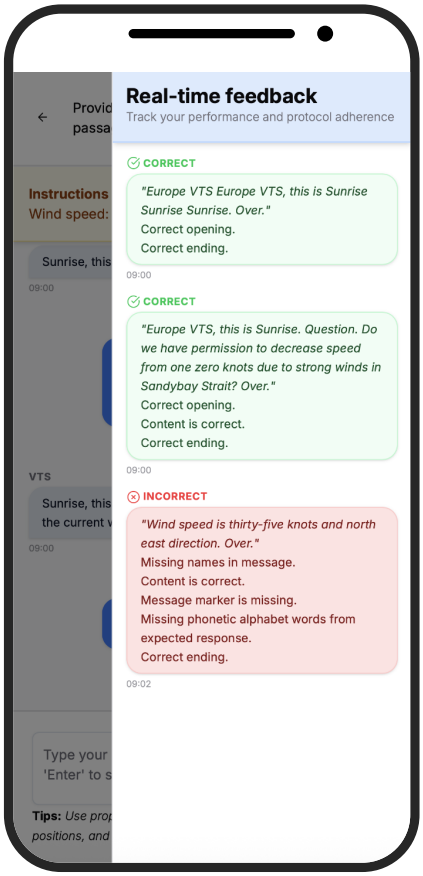


(a) Dator **(b)** Mobil

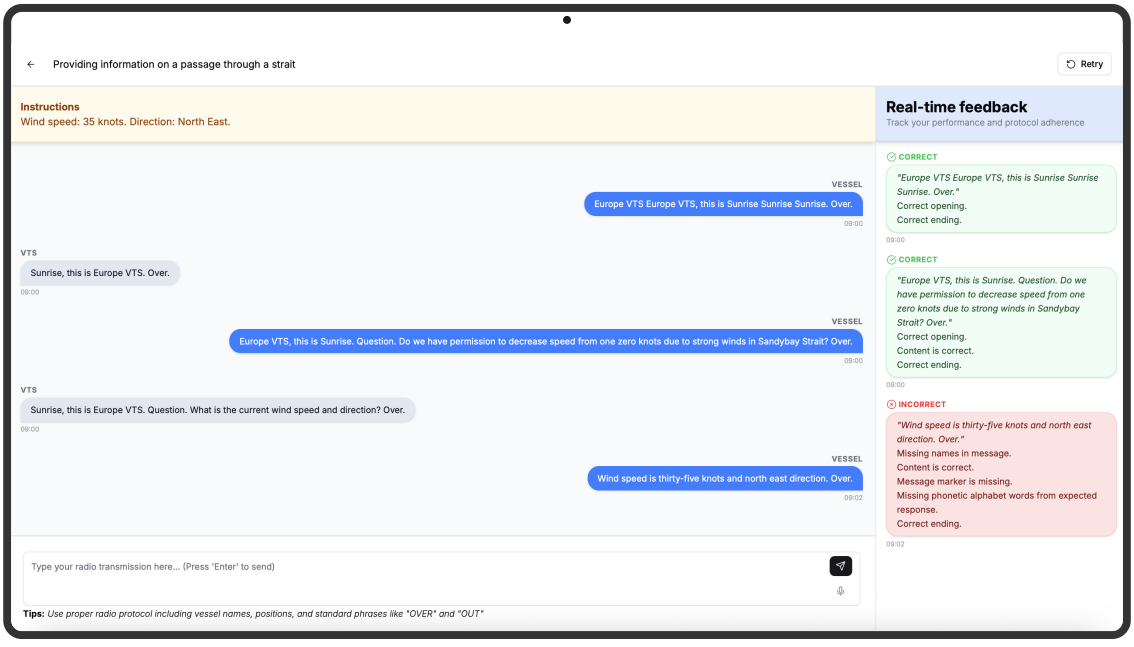
Figur 8: Startsidan med rollval och scenariolista



(a) Mobil chattvy



(b) Feedbackpanel öppnad på mobil enhet



(c) Datorvy med feedbackpanel till höger

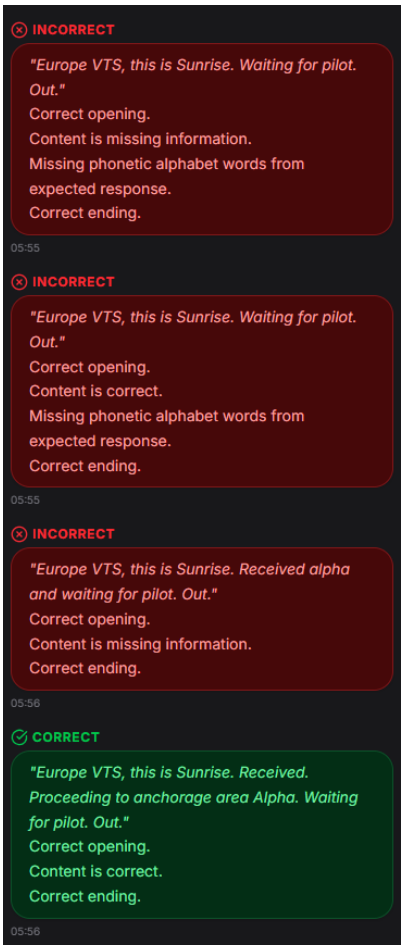
Figur 9: Chattvy under ett aktivt scenario

5.2 Meddelandehantering

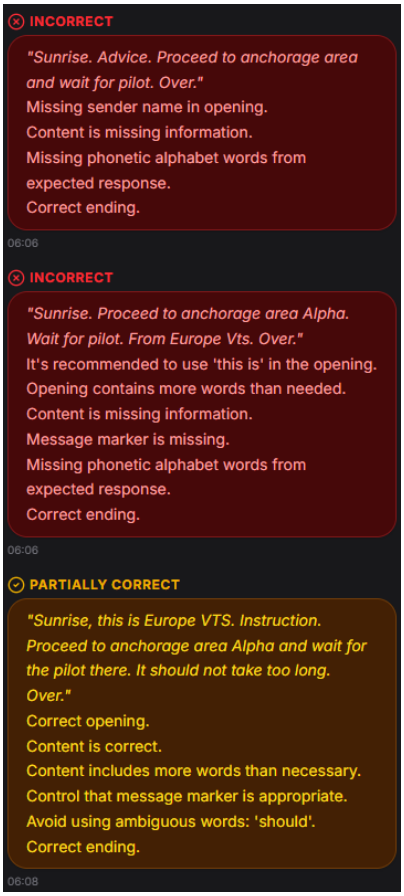
Detta delkapitel kommer handla om hur meddelanden hanteras av applikationen och hur feedbacken är konstruerad och visas för användare.

5.2.1 Feedbackens format

Visuellt så har feedbacken huvudsakligen två komponenter. Dels är det färgerna på feedbackrutan samt dess text som indikerar meddelandets korrekthet men även utskrivna rader med vilka av meddelandets ingående delar som är korrekta eller skulle kunna förbättras. Bilder i Figur 10 visar hur feedbackens format är konstruerat. När ett meddelande har bedömts som felaktigt behöver det skrivas om tills det accepteras. Överst inom citattecken är användarens meddelande, som även visas i själva chattrutan. Därefter följer feedback från meddelandets ingående delar från början till slut med att varje enskild kommentar börjar på ny rad. Exempelvis är kommentaren ”Content is missing information.” från jämförelsen mellan betydelsen i användarens och svarets meddelandeinnehåll som i nuläget enbart returnerar sant eller falskt.



(a) Inkonsekvent modell



(b) Uppbyggnad av feedback

Figur 10: Exempel på genererad feedback.

5.2.2 Algoritmer och felräknare

Framförallt är figur 10b relevant för hur algoritmerna är uppbyggda. Det korrekta svaret för det meddelandet är "*Sunrise, this is Europe VTS. Advice. Proceed to anchorage area Alpha. Wait for pilot. Over*". Det översta meddelandet saknar avsändaren samt *alpha* som är en del av Phonetic Alphabet som ska användas. I detta fall resulterar det i att både innehållet saknar information samt att en bokstavering som finns med i svaret saknas i meddelandet.

Det andra meddelandet som visas i 10b liknar det korrekta svaret, men ingen Message Marker har använts och avsändaren (*Europe VTS*) anges i slutet. Detta är en specifik kombination av fel som leder till att meddelandeinnehållet från användaren inte fastställs utan hela meddelandet förutom *Over* sätts som anrop. Det medför att meddelandet bedöms som inkorrekt och flertalet felaktiga kommentarer om meddelandets innehåll visas för användaren.

Det sista meddelanden bedömdes som *Partially Correct* och godkändes därmed för att gå vidare i scenariot. Till skillnad från det korrekta svaret så byttes *advice* ut mot *instruction*. I vissa fall måste inte nödvändigtvis svarets Message Marker vara det enda korrekta, vilket är anledningen till att feedbacken inkluderar *Control that message marker is appropriate*. Dessutom innehåller feedbacken kommenterar om att meddelandet är längre än det behöver vara samt att ett Ambigues Word har använts. Anledningen till detta är att *It should not take too long* har lagts in i svaret och är överflödigt.

5.2.3 Integrationen av AI

Användningen av AI är begränsad, framförallt för eftersträva att chatboten ska vara deterministisk. I figur 10a framgår det att de två översta meddelandena är exakt samma. Dock innehåller feedbacken till det första meddelandet *Content is missing information* medan det andra har raden *Content is correct*. Denna feedback genereras utifrån jämförelsen som nyttjar AI-modellen. Den här jämförelsen har då resulterat i två olika utfall för två identiska meddelanden.

5.2.4 Muntlig kommunikation

Funktionaliteten för röstinmatning till chatboten stöds med Web Speech API. Ett fåtal justeringar, som specificerats i underavsnitt 4.2.4, behövde genomföras med inbyggda funktioner för anpassning till maritima begrepp. Meddelandet behandlas därefter på samma sätt som ett inskrivet och identifierar därmed exempelvis inte om längre pauser har inträffat under röstinmatningen.

5.3 Utvärdering genom enkät

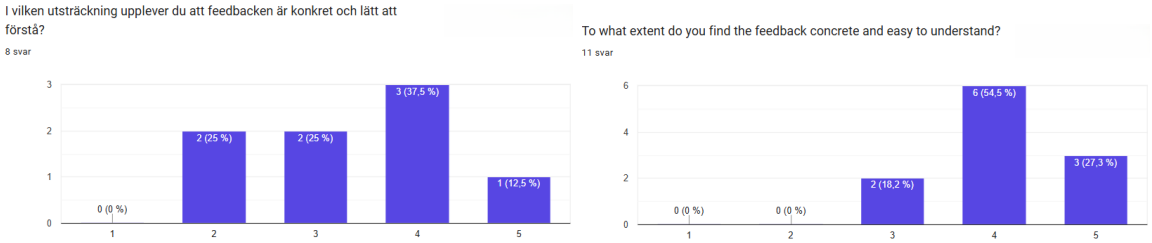
Två enkätundersökningar har genomförts för att utvärdera chatbotens nuvarande funktionaliteter. Det huvudsakliga målet med undersökningarna var att få en gene-

rell uppfattning om chatbotens användarvänlighet samt korrekthet. En enkät har genomförts på svenska och har riktats mot utbildningsansvariga samt studenter på Chalmers Sjöfartshögskola. Enkäten kan ses i sin helhet i Appendix B Figur B.1. Den andra enkäten är uppbyggd med samma format och frågor men har genomförts på engelska för att vara riktad mot internationella studenter. De flesta frågorna var strukturerade för svar mellan ett och fem, där ett är lägsta betyg och fem högsta.

5.3.1 Enkätens frågor samt svar

Åtta svar inkom till den svenska enkäten och elva till den engelska. Svar till samtliga frågar finns i sin helhet från Appendix B Figur B.2. Överlag bedömdes chatboten som relativt användarvänlig samt ett bra verktyg för att utveckla maritim kommunikation.

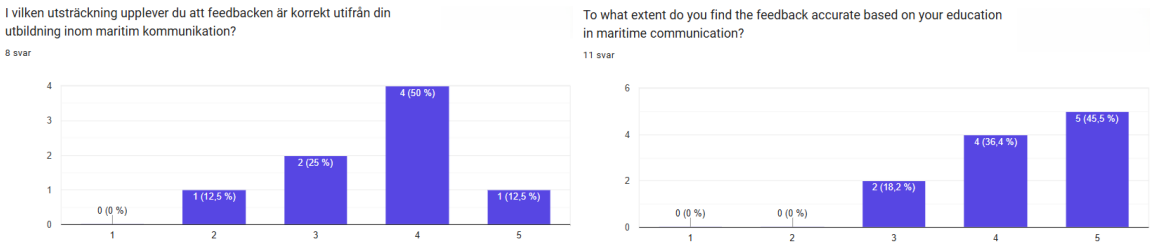
De frågor som fick lite sämre svar gällde feedbacken. Svaren till fråga 5 visas nedan i Figur 11. Den handlade om feedbacken är konkret och lätt att förstå och med 19 svar erhölls ett medelvärde på 3,79 och svaren från den engelska enkäten var betydligt positivare än från den svenska.



(a) Svenska svar. **(b)** Engelska svar.

Figur 11: Svar på fråga 5.

Efterföljande fråga var också relaterad till feedbacken och ses nedan i Figur 12. Den var ställd för att undersöka feedbackens korrekthet baserat på testanvändarens kunskaper. Även för denna fråga var svaren från den engelska enkäten positivare och de 19 mottagna svaren resulterade i ett medelvärde på 4,00.



(a) Svenska svar. **(b)** Engelska svar.

Figur 12: Svar på fråga 6.

5.3.2 Kommentarer från enkäten

Alternativfrågorna från enkäten följdes även av frågan ”Har du några särskilda synpunkter som du vill specificera?” för att kunna erhålla mer specifika åsikter angående chatbotens nuvarande status. Det mottogs svar på den svenska enkäten som framförallt syftade på feedbacken samt utveckling av scenarier. Kommentarerna om feedbacken handlade mestadels om att det var svårt att förstå exakt vad meddelandet förväntas innehålla för information och maritima termer.

6
Slutsats

Detta examensarbete har utförts på uppdrag av Chalmers Tekniska Högskola i syfte att ta fram ett interaktiv hjälpmedel för inlärning av maritim kommunikation. Resultatet är en responsiv webbapplikation, tillgänglig på dator- och mobilenheter, som låter användaren öva på vanligt förekommande scenarier i rollen som fartyg eller VTS operatör.

Systemet är utformat som en fullstack-applikation med ett React-baserat frontendgränssnitt och en Express-backend. Feedback genereras primärt algoritmiskt genom att analysera strukturen på användarens meddelande utifrån maritima konventioner. Utöver algoritmisk analys används även en AI-baserad stavningskontroll och betydelsejämförelse via OpenAI:s API. Scenarion har tillhandahållits av beställaren och systemet är utformat för att enkelt kunna addera nya scenarion utan att redigera källkoden.

De erfarenheter som dragits från detta projekt är relevanta för liknande system där algoritmisk analys kombineras med AI. En viktig lärdom är att AI kan användas för ett väldefinierat delproblem, snarare än som en generell lösning. Således kan man säkerställa att systemet är kontrollerbart, välfungerande och kostnadseffektivt.

6.1 Feedback och användarvänlighet

Överlag uppfylls projektets syfte genom att tillhandahålla ett interaktivt hjälpmedel för att öva maritim kommunikation. Systemet instruerar användaren genom fördefinierade scenarion, ger feedback i realtid och stöder både text- och röstinmatning.

Däremot har svagheter i meddelandehanteringen identifierats och framgår av resultaten i underavsnitt 5.2.2 och 5.2.3. Analysen av ett meddelande riskerar att misslyckas om anropsnamn eller mottagarnamn först nämns i slutet av meddelandet. Dessutom kan betydelsejämförelsen som görs med hjälp av integrerad AI-modell bedöma liknande meddelandeinnehåll på olika sätt. Dessa svagheter riskerar att ge användaren felaktig eller tvetydig feedback.

6.2 Avvikelser från konstruktionsflödet

Konstruktionsflödet som beskrevs i avsnitt 2.3 utgjorde grunden för hur applikationens utformades, men ett antal avvikelser gjordes under projektets gång med hänsyn

till tidsbrist och medvetna prioriteringar.

Identifiering av Action Verbs implementerades inte. Beslutet fattades för att prioritera övriga delar av feedback-algoritmen, såsom identifiering av meddelandestruktur, Phonetic Alphabet och Ambiguous Words. Dessa bedömdes som mer centrala för att ge en användarvänlig feedback inom projektets tidsram.

Identifiering av Procedure Words har delvis implementerats genom att algoritmen identifierar meddelandets avslut. En fullständig implementation har dock prioriterats bort på grund av tidsbrist.

Analys av pauser och tydlighet vid röstinmatning implementerades inte. Detta är en direkt konsekvens av nyttjandet av Web Speech API, vilket inte ger någon teknisk möjlighet till sådan analys, då rådata från ljudinspelningen inte är tillgänglig. Dessutom hade det varit alltför tidskrävande att genomföra inom ramen för projekttiden.

Sammantaget täcker applikationen en viss del av konventionerna för maritim kommunikation och utgör inte en fullständig implementation av SMCP. Detta bör beaktas vid användning av systemet i ett utbildningssammanhang.

6.3 Tekniska vägval

Ett centralt tekniskt vägval var att utforma feedback-genereringen algoritmiskt snarare än att helt förlita sig på AI. Detta beslut fattades för att få kontroll över systemets beteende och att minimera konstader, då varje API-anrop mot OpenAI innebär en kostnad för beställaren. Fördelen är att systemet blir förutsägbart och det är tydligt varför ett visst svar bedöms som korrekt eller inkorrekt. Nackdelen är att algoritmerna är begränsade och således täcks inte hela SMCP.

Kommunikationsflödet mellan klient och server för inlämning av användarmeddelande, hämtning av feedback samt hämtning av nästa tur implementerades genom sekventiella HTTP-anrop. Ett alternativ hade varit att samla hela flödet via en endpoint. Nackdelen med detta är att klienten hade behövt invänta samtliga steg. Givet den tid det tar för externa API-anropen till OpenAI, under feedbackgenereringen, hade risken varit att gränssnittet uppfattas som alltför långsamt och trögt. Med det valda upplägget får istället klienten direkt bekräftelse på inlämning av meddelande, medan feedback hämtas asynkront. Ett ännu mer naturligt val hade varit att implementera detta flöde med hjälp av *Web sockets*, vilket hade möjliggjort att server skickar feedback och nästa tur när de är redo, utan att klienten explicit behöver efterfråga det.

Röstinmatning implementerades via webbläsarens inbyggda Web Speech API. Valet motiverades utifrån dess enkelhet att integrera i förhållande till funktionaliteten det tillför. Samtidigt är API:et inte anpassat för sammanhanget maritim kommunikation. För ett läromedel där uttal av maritima termer är central utgör detta en stor nackdel.

6.4 Användarutvärdering

Svaren på enkäterna visar att testanvändare anser att chatboten är ett användbart verktyg för att utveckla sina maritima kommunikationskunskaper. Dock finns det även oklarheter som bör åtgärdas för att förbättra chatboten. Responsen syftade framförallt på förbättringar genom att tydligare visa för användare hur ett meddelande behöver förändras för att bedömas som korrekt. Detta kan genomföras med möjliga förbättringar som nämns senare i underavsnitt 6.5.3.

6.5 Vidareutveckling

Det finns flertalet potentiella vidareutvecklingar som skulle kunna genomföras för att åstadkomma att chatboten blir ett mer interaktivt och användarvänligt utbildningsverktyg.

6.5.1 Systemförändringar

Användningen av *Web sockets* skulle förbättra flödet under ett pågående scenario och göra det mer intuitivt. Detta på grund av att feedback kan skickas till klienten så fort det är redo och undvika att invänta http-anrop och också göra chatboten mer responsiv.

En till förbättring skulle vara att byta ut Web Speech API för röstinmatning. Vid muntlig kommunikation i allmänhet och över radio i synnerhet är både tydligt uttal samt att tala i lagom rytm mycket viktigt. Web Speech API saknar dessvärre funktionaliteten att ge information om dessa och är därmed suboptimalt för detta ändamål. En ytterligare punkt till detta är att implementera ett val för uppläsning av chatbotens meddelanden.

6.5.2 Algoritmer

Applikationens algoritmer för att generera feedback skulle kunna förbättras och utökas för att göra chatboten mer omfattande och tydlig för användare. Detta skulle kunna inkludera följande punkter.

- Implementera algoritmer för Action Verbs.
- Implementera algoritmer för Procedure Words.
- Förbättra undersökningen av Message Markers. I nuläget identifieras inte hur dessa ord används, alltså om det är en markör innan information ges eller om de används mitt i en mening.
- Undersöka om det finns specifika begrepp som ska användas vid vissa tillfällen under maritim kommunikation. Potentiellt genom att kontrollera om de finns i svaret och i så fall även i användarens meddelande.

6.5.3 Integrering till webbsida med utbildningsmaterial

I bakgrunden, avsnitt 1.1, nämndes att denna chatbot är en del av ett större projekt för utbildningsmaterial inom maritim kommunikation. Integration av chatboten till en webbsida med information om maritim kommunikation skulle möjliggöra att genererad feedback kan bli interaktiv. Feedback skulle ges med länkar till relevant information, såsom Phonetic Alphabet eller Message Markers när de saknas eller är fel. Detta skulle tillhandahålla mer konkret information till användare för att enklare förstå hur ett meddelande bör förändras. Ytterligare skulle även information om meddelandets struktur och dess ingående delar kunna göras tillgänglig för användarna. Detta skulle bidra till en ökad förståelse för de principer om meddelandets struktur och termer som ligger till grund för utvecklingen av chatboten.

6.5.4 Anpassning till andra kommunikationsdomäner

Systemets utformning är inte strikt anpassad till maritim kommunikation, utan är snarare anpassat utifrån scenariernas JSON-struktur. Detta gör det principiellt möjligt att de algoritmiska lösningar som implementerats för att identifiera maritimspecifika begrepp kan bytas ut till andra domänspecifika algoritmer. En potentiell vidareutveckling av applikationen skulle alltså kunna omfatta andra kommunikationsprotokoll som används exempelvis inom flygledning samt polisiär och militär kommunikation.

Referenser

[1] G. Gabedava och Y. Hu, ”Enhancing maritime safety through linguistic analysis: a case study of communication failures in maritime accidents,” *WMU Journal of Maritime Affairs*, årg. 24, nr 3, s. 347–361, april 2025. DOI: 10.1007/s13437-025-00371-y.

[2] International Maritime Organization, *IMO Standard Marine Communication Phrases*, [Online]. hämtad 22 jan. 2026. Tillgänglig: https://www.imo.org/en/ourwork/safety/pages/standardmarinecommunicationphrases.aspx.

[3] Meta Open Source, *React*, [Online]. hämtad 5 maj 2026. Tillgänglig: https://react.dev/.

[4] Microsoft, *What is TypeScript?* [Online]. hämtad 6 febr. 2026. Tillgänglig: https://www.typescriptlang.org/.

[5] Tailwind Labs Inc., *Styling with utility classes*, [Online]. hämtad 6 febr. 2026. Tillgänglig: https://tailwindcss.com/docs/styling-with-utility-classes.

[6] Shadcn, *Introduction*, [Online]. hämtad 5 maj 2026. Tillgänglig: https://ui.shadcn.com/docs.

[7] OpenJS Foundation, *Express - Node.js web application framework*, [Online]. hämtad 5 maj 2026. Tillgänglig: https://expressjs.com/.

[8] International Maritime Organization, *IMO Standard Marine Communication Phrases*, Resolution A.918(22), [Online], nov. 2001. hämtad 22 jan. 2026. Tillgänglig: https://wwwcdn.imo.org/localresources/en/OurWork/Safety/Documents/A.918(22).pdf.

[9] D. Ernstsson, privat kommunikation, April. 2026.

[10] Meta Platforms, Inc., *Higher-Order Components*, [Online]. hämtad 6 maj 2026. Tillgänglig: https://legacy.reactjs.org/docs/higher-order-components.html.

[11] *How to use vanilla stores in React*, [Online]. hämtad 6 maj 2026. Tillgänglig: https://zustand.docs.pmnd.rs/reference/hooks/use-store.

[12] MDN Contributors, *Web Speech API*, [Online]. hämtad 6 maj 2026. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API.

[13] MDN Contributors, *Using the Web Speech API*, [Online]. hämtad 6 maj 2026. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API.

[14] Motion, *Get started with Motion for React*, [Online]. hämtad 6 maj 2026. Tillgänglig: `https://motion.dev/docs/react`.

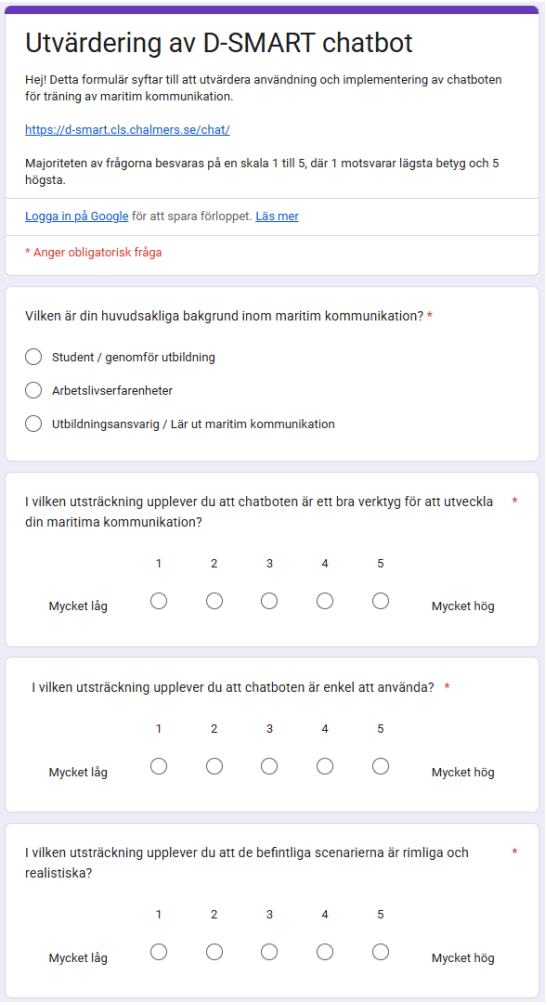
A

Appendix A

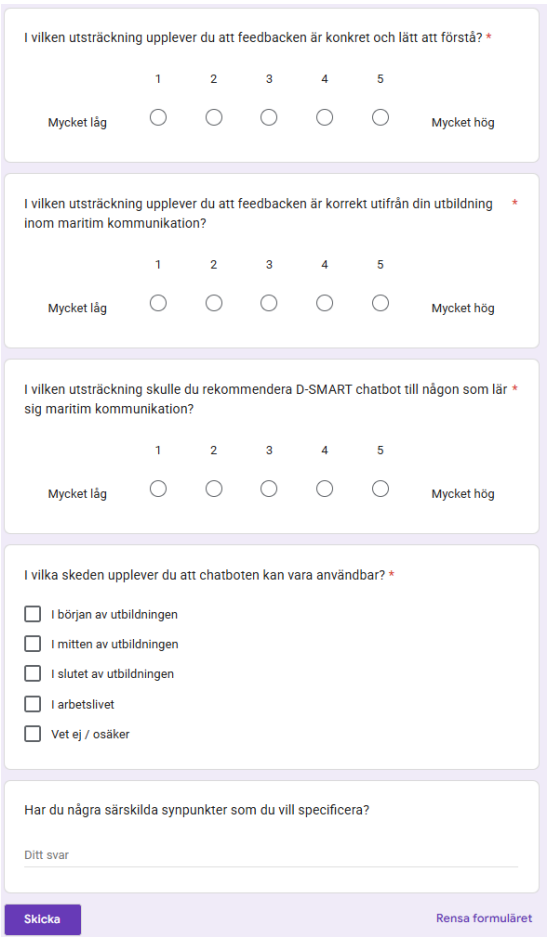
Figur A.1: Fullständigt klassdiagram över applikationens backend

B

Appendix B

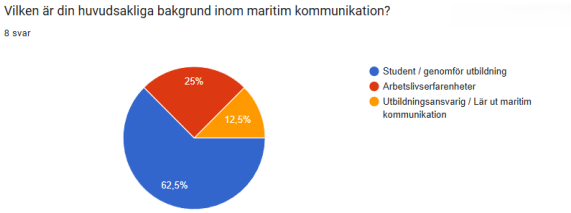


(a) Enkätens första del

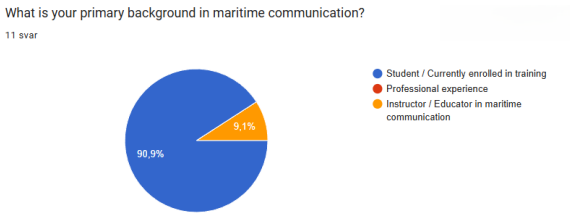


(b) Enkätens andra del

Figur B.1: Enkätens utformning



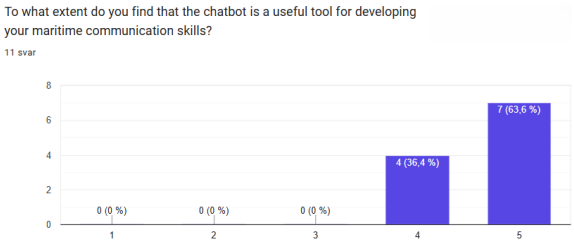
(a) Svenska svar.



(b) Engelska svar.

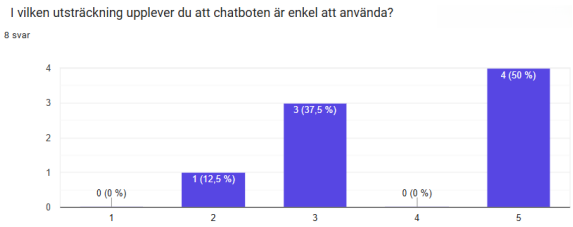
Figur B.2: Svar på fråga 1.

(a) Svenska svar.

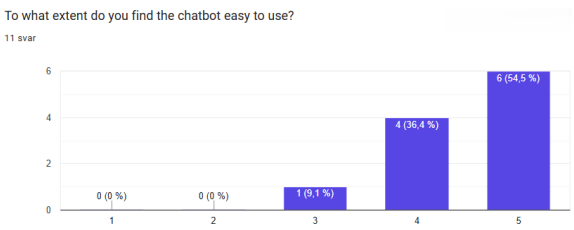


(b) Engelska svar.

Figur B.3: Svar på fråga 2.

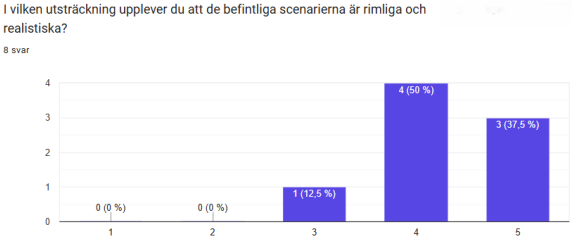


(a) Svenska svar.

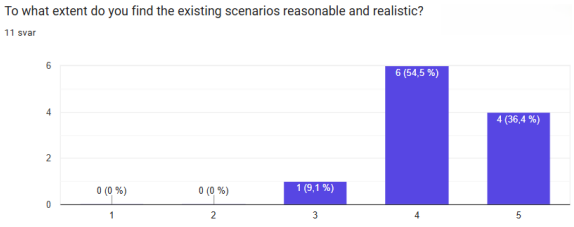


(b) Engelska svar.

Figur B.4: Svar på fråga 3.

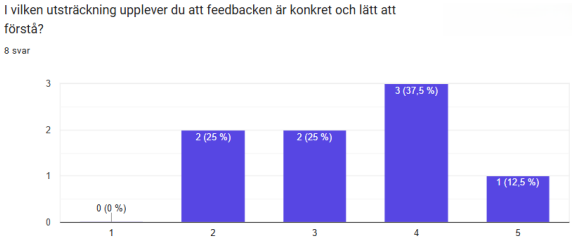


(a) Svenska svar.

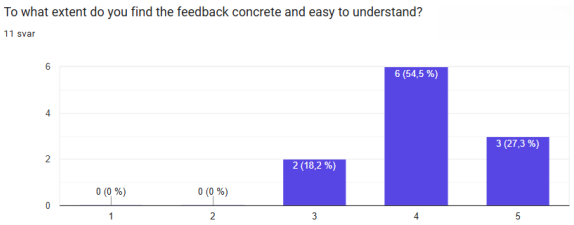


(b) Engelska svar.

Figur B.5: Svar på fråga 4.

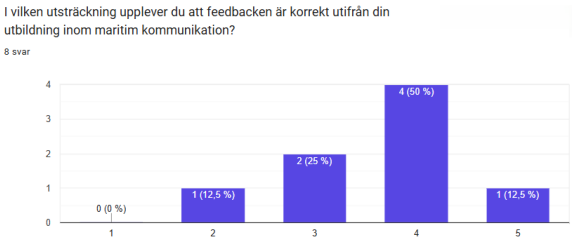


(a) Svenska svar.

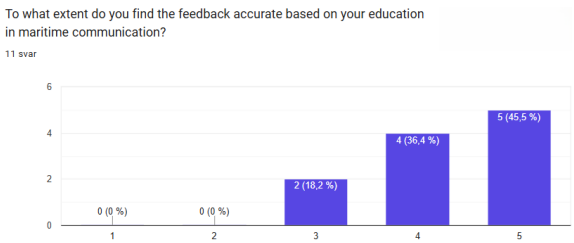


(b) Engelska svar.

Figur B.6: Svar på fråga 5.

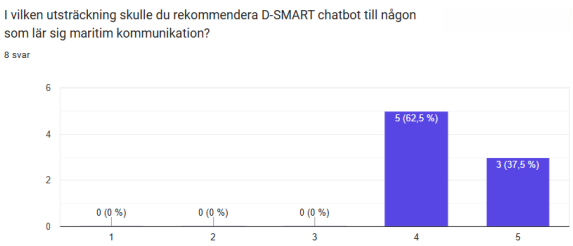


(a) Svenska svar.

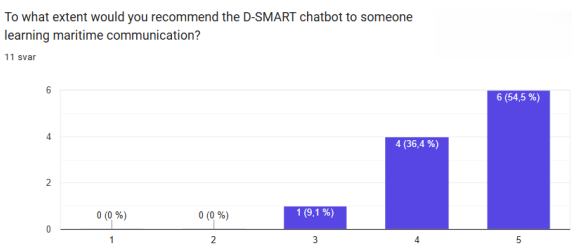


(b) Engelska svar.

Figur B.7: Svar på fråga 6.

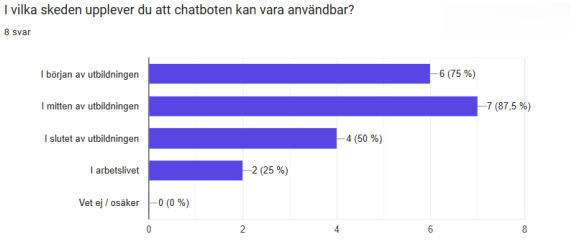


(a) Svenska svar.

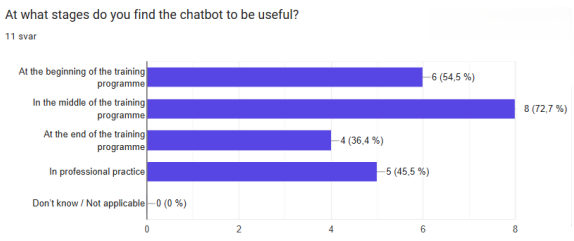


(b) Engelska svar.

Figur B.8: Svar på fråga 7.



(a) Svenska svar.



(b) Engelska svar.

Figur B.9: Svar på fråga 8.

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

CHALMERS