Integrera digitala utbildningsmedier i en interaktiv applikationsprototyp

Design av webbapplikation för inlärande syften

Kandidatarbete i kursen LMTX38

Filip Sjöström och Tony Quach

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2026
www.chalmers.se

LMTX38

Integrera digitala utbildningsmedier i en interaktiv applikationsprototyp

Filip Sjöström,
Tony Quach

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2026
www.chalmers.se

Abstract

A primary cause of maritime accidents is human error, stemming from factors like inadequate communication and a lack of digital competency. Maritime education and research are still dispersed throughout the world despite technology advancements and the standardised SMCP methodology. The EU-funded Erasmus program and the DigiMar project (2023) has developed digital learning materials from authentic communication recordings to address this gap.

The purpose of this project was to build upon this foundation and develop a web application that serves as an educational platform for maritime communication. To achieve this, the design combines theory and practice to provide a comprehensive and practical learning experience, while also enabling teachers to monitor and track student progress. The material used is intended for maritime communication students, while teachers can set requirements and track student progress. Based on these educational and administrative needs, The solution was to create an interactive platform that offers courses in different levels (basic and advanced). Users can easily navigate between the offered courses/chapters. The course material is presented in slides and the student progress is tracked to give a personal learning experience. A user can also create a classroom and invite others to join. This allows them to track each member’s learning progress and set goals for specific content that needs to be completed.

Sammanfattning

En primär orsak till sjöolyckor är mänskliga fel, som kommer från faktorer som otillräcklig kommunikation och brist på digital kompetens. Maritim utbildning och forskning är fortfarande otillräcklig över hela världen trots tekniska framsteg och den standardiserade SMCP-metoden. Det EU-finansierade Erasmus-programmet och DigiMar-projektet (2023) har utvecklat digitala läromedel för att försöka bemöta denna brist.

Syftet med detta projekt var att bygga vidare på denna grund och utveckla en webbapplikation som fungerar som en utbildningsplattform för maritim kommunikation. För att uppnå detta kombinerar designen teori och praktik för att ge en omfattande och praktisk lärandeupplevelse, samtidigt som lärare kan övervaka och följa elevernas framsteg. Materialet som används är avsett för elever i maritim kommunikation, medan lärare kan ställa krav och följa elevernas framsteg. Baserat på dessa utbildnings- och administrativa behov var lösningen att skapa en interaktiv plattform som erbjuder kurser på olika nivåer (grundläggande och avancerade). Användare kan enkelt navigera mellan de erbjudna kurserna/kapitlen. Kursmaterialet presenteras i bilder och elevernas framsteg spåras för att ge en personlig lärandeupplevelse. En användare kan också skapa ett klassrum och bjuda in andra att delta. Detta gör det möjligt för dem att följa varje medlems lärandeframsteg och sätta mål för specifikt innehåll som behöver slutföras.

Innehållsförteckning

1. Inledning

Mänskliga faktorer som brist på kommunikation och begränsad digital kompetens är en stor orsak till olyckorna inom sjöfarten. Trots betydande tekniska framsteg och etablerade internationella kommunikationsprotokoll menar Standard Marine Communication Phrases (SMCP) att den maritima utbildningen fortfarande håller en otillräcklig nivå. För att bemöta dessa utmaningar har Erasmus, ett EU-finansierat program, tillsammans med DigiMar projektet (2023) utvecklat digitala läroverktyg baserade på autentiskt maritimt kommunikationsmaterial. Med utgångspunkt i tidigare forskning och relaterade kandidatprojekt, syftar detta arbete till att integrera dessa läromedel i en interaktiv webbapplikation för att på sikt främja den internationella förståelsen för säker och effektiv kommunikation till sjöss.

1.1 Syfte

Uppgiften är att utveckla en webbapplikation som ska ta existerande lärandematerial och integrera med ett användargränssnitt. Arbetet genomförs parallellt med flera andra grupper som jobbar med en databas-API och en chatbot. Databas-API:n kommer att lagra allt läromedel från DigiMar vilket inkluderar dokument, videor och transkript till alla videos. Chatbottens syfte är att fungera som en instuderings-agent vilket studenter ska kunna använda sig av efter att de har läst till sig material. Syftet är också att integrera dessa arbeten med vårt arbete så att vi kan hämta information via API:n och så att studenter skall kunna navigera till chatbotten för att kunna öva in sin kunskap. Arbetet kommer främst vara fokuserat på frontend, men kommer också att inkludera användarhantering för backend. Som användare ska man kunna navigera och lätt ta del av DigiMar-materialet med hjälp av webbapplikationen.

1.2 Mål

Målet med arbetet är att ta fram och implementera ett intuitivt gränssnitt med hjälp av ett öppet bibliotek för webbutveckling, som samtidigt uppfyller följande krav:

- En intuitiv och responsiv design som gör upplevelsen samt lärandet engagerande för studenter.
- Material ska kunna hämtas och visas upp via extern API på ett visuellt tilltalande sätt på webbapplikationen.
- Lärare ska kunna skapa målsättningar för studenter i syfte att säkerställa att studenterna har tillräckliga förkunskaper inför kommande kursmoment, exempelvis laborationer och simuleringar.

1.3 Avgränsningar

Våra avgränsningar för projektet är följande:

- Webbapplikationen kommer inte att integreras med något annat externt förutom databas-API:n och chatbotten.
- Webbapplikationen ska inte baseras på en allmän lärplattform.
- Webbapplikationen får inte användas i marknadsföringssyften.

2. Teknisk bakgrund

Under detta avsnitt kommer en teknisk bakgrund ges gällande begrepp/koncept, ramverk och verktyg som använts under arbetsprocessen.

2.1 Webbarkitektur

2.1.1 Klient-server modellen

I denna webbapplikation används klient-server-modellen för att separera användargränssnittet med logiken för att hantera webbapplikationen. Klienten (frontend) är vanligtvis en webbläsare eller applikation som användaren interagerar med. Den visar information och skickar förfrågan till servern. Servern (backend) tar emot inkommande förfrågningar, bearbetar dem och skickar en motsvarande respons till klienten. Klient-server-modellen separerar inte bara användargränssnittet och serverlogiken utan tillåter även flera klienter att koppla sig till en och samma server samtidigt. Detta gör webapplikationen mer säker eftersom känsliga operationer sker på servern [1].

2.1.2 REST-API

Ett REST-API fungerar som den huvudsakliga kommunikationskanalen mellan webbapplikationens klient och server. Servern har något som kallas för “endpoints”, där varje endpoint hanterar olika typer av information och förfrågan. När användaren interagerar med frontenden, så skickas HTTP förfrågan (GET, POST, PUT eller DELETE) till dessa endpoints. Servern hanterar varje förfrågan, interagerar med databasen om det behövs och returnerar en respons i form av JSON. Detta tillåter frontenden att visa aktuell information och hantera förfrågan utan att ladda om sidan [2].

2.2 Frontend-teknologier

2.2.1 React

React är ett JavaScript-ramverk som används vid utveckling av användargränssnitt för exempelvis webb- och mobilapplikationer. Ramverket är strukturerat kring komponenter vilket möjliggör en större separering av logiken i programmet. “State” (tillstånd), är en grundläggande idé inom React vilket tillåter komponenter att lagra data som ändras under en lång tid. När ett tillstånd ändrar sitt värde, uppdaterar komponenterna sig för att visa den nya informationen. En annan viktig funktion inom React är “Props”, vilket är ett enkelt sätt att skicka data från en “parent-komponent" till en “child-komponent” [3].

En Document Object Model (DOM) är ett programmerargränssnitt för webbdokument. DOM ansluter hemsidor till skript och de programmeringsspråk som beskriver sidans innehåll, exempelvis HTML. Begreppet syftar oftast på JavaScript, även om modellering av dokument som HTML-objekt egentligen inte ingår i själva kärnan av Javascript-språket. React använder något som kallas för en virtuell DOM vilket är en kopia av den riktiga DOM:en. Detta innebär att hela sidans innehåll inte behöver renderas om varje gång något ändras, utan endast de delar som faktiskt behöver uppdateras renderas på nytt [4].

2.2.2 React Router

React Router är ett bibliotek inom React som tillåter navigering bland flera vyer för React-applikationer. Vissa vyer går att “skydda” med hjälp av någon autentisering. Oautentiserade användare som försöker nå en skyddad sida omdirigeras till exempelvis en inloggningssida. En React-applikation utan någon router skulle resultera i en enda vy, utan möjlighet att navigera mellan olika sidor eller komponenter [5].

2.2.3 TypeScript

TypeScript är ett programmeringsspråk som bygger på JavaScript men lägger till stöd för typer i syntaxen. Genom att definiera värden, funktionsparametrar, returvärden och

datastrukturer, kan typescript-kompilatorn validera dessa typer innan koden körs. Detta leder till att man fångar vanliga fel tidigt (som att anropa en funktion med fel data eller anta att en datatyp finns när den kan vara odefinierad) och en bättre integration med kodredigeraren. All kod i TypeScript transpileras till Javascript vilket innebär att all kod i Typescript kan köras som vanlig JavaScript-kod [6].

2.2.4 CSS

CSS (Cascading Style Sheets) används för att anpassa innehållet av en webbsida genom att lägga till designelement som exempelvis färger, teckensnitt och mellanrum. Det är möjligt att ändra allt från den övergripande layouten till enskilda HTML-element. CSS består av selektorer och egenskaper. Selektorer används för att ange vilka egenskaper som taggar ska ha, till exempel kan stycken (**<p>**) få ett visst typsnitt och storlek medan rubriker (**<h1>, <h2>, <h3>**) kan ges en annan stil [7].

2.3. Backend-Teknologier

2.3.1 Node.js och Express

Node.js är en kodmiljö som fungerar på flera olika operativsystem, till exempel Windows, Mac och Linux. Det gör det möjligt att köra JavaScript-program i miljöer som inte är webbläsarbaserade, direkt på datorn eller servern [8]. Att skapa en webbsida enbart med Node.js kan vara ganska komplicerat [9]. Därför används ofta Express.js, ett ramverk byggt på Node.js, för att förenkla processen. Express.js erbjuder enkla sätt att definiera sökvägar, hantera förfrågningar och svar, samt fungerar som ett middleware för flera funktioner som till exempel autentisering, loggning och felhantering. Därför väljer många utvecklare att använda Node.js och Express tillsammans för att skapa en mer effektiv och hållbar kodmiljö [10].

2.3.2 JSON

JSON (JavaScript Object Notation) är ett format för att lagra och transportera data. Formatet följer en namn/värde-konvention och all data separeras med kommatecken.

JSON kan lätt omvandlas till JavaScript objekt då båda är syntaktiskt identiska med varandra. JSON-objekt kan lagra mer än ett datavärde med hjälp av “curly braces {}” vilket betyder att flera namn/värdes-par kan skapas inom ett objekt. Slutligen kan objekt omges av listor (arrays) vilket gör att flera objekt dessutom kan struktureras i samma dokument [11].

2.3.3 PostgreSQL och Sequelize

PostgreSQL är ett objekt-relationsdatabassystem som använder sig av programmeringsspråket SQL för att lagra och hantera data i databaser. Det överensstämmer till en mycket hög grad med den allmänna SQL-standarden och är kompatibelt med ett flertal operativsystem däribland Windows, Linux och macOS. Med Postgres är det möjligt att skapa egna datatyper för datan som lagras, exempelvis Integer, String och JSON/JSONB [12].

För att definiera och skapa data kan hjälpmedlet Sequelize användas tillsammans med TypeScript eller JavaScript. Sequelize är en ORM (Object-Relational Mapping) tool som bland annat kan översätta TypeScript till SQL. Det är då möjligt att definiera data, skapa associationer och queries med hjälp av skriptspråk genom att låta Sequelize sköta översättningen till SQL [13].

2.4 Produktion av webbsidan

2.4.1 Docker

Docker är en öppen plattform som används för att utveckla och köra applikationer. Plattformen möjliggör paketering av applikationer till en mindre, mer isolerad version som kallas för containers. Allt som behövs för att applikationen skall kunna köras tillhandahålls av Docker och flera containrar kan dessutom köras på samma host vilket minskar antalet beroenden som krävs för att få en fungerande webbplats [14].

2.4.2 HTTP/HTTPS

HTTP står för Hypertext Transfer Protocol och är ett centralt protokoll på internet som används för att hämta och överföra webbsidor. När en användare gör en förfrågan på internet skickas en HTTP-request till en server, som i sin tur svarar med ett HTTP-response. I en sådan förfrågan ingår information om vilken resurs som efterfrågas samt ytterligare data som hjälper servern att producera korrekt svar. All data som överförs via HTTP skickas i vanlig text, vilket innebär att informationen i teorin kan avlyssnas av obehöriga [15]. Utifrån det har HTTPS (Hypertext Transfer Protocol Secure) utvecklats i syfte att göra surfning på nätet mer säkert. HTTPS använder sig av kryptering för att skydda information från obehörig avlyssning och gör överföringen av data mer säker på webben [16].

3. Metod

Metoderna för att genomföra examensarbetet började främst med inhämtning av information från digitala källor, exempelvis genom att sätta sig in i DigiMar-materialet samt studera designidéer. Examensarbetet genomfördes i par och delades in i flera mindre delmoment. Under varje vecka lades det fokus på särskilda implementationer som sedan färdigställdes. Regelbundna avstämningsmöten hölls i slutet av varje månad tillsammans med tillhörande grupper inom projektet. Arbetssättet kan beskrivas som delvis agilt med mycket fokus på användaren. Veckovisa möten planerades in för att diskutera projektets status och utifrån detta gjordes en informell lista med uppgifter som skulle göras under veckan. Inga direkta “User stories” formulerades utan uppgifterna var allt från små justeringar till större implementationer. Anledningen till att agile scrum valdes var för att anpassa utvecklingen av projektet till användarna och att lämna större utrymme för ändringar beroende på vad som efterfrågades.

3.1 Planering och arbetsfördelning

Projektet inleddes med en analys av det tidigare upplägget för DigiMars läromaterial riktat till sjöfartsstudenter. Analysen fokuserade särskilt på den struktur och presentation som det befintliga läromaterialet hade. Resultaten från genomgången och de efterföljande diskussionerna kring designidéer användes som grund för att ta fram en HTML-mockup som visualiserade det föreslagna nya upplägget. Arbetsfördelningen var främst uppdelad mellan backend och frontend mellan medlemmarna, med några undantag gällande vissa implementationer. Uppdelningen av arbetsuppgifter bestod av en att-göra-lista som omfattade flera implementationer som skulle göras på webbapplikationen. Denna listan uppdaterades i samband med att möten hölls veckovist mellan medlemmarna, då nya funktionaliteter tillkom varje vecka.

3.2 Val av verktyg

Ramverket React användes för webbapplikationen då det ansågs vara mest passande för utvecklingen av detta projekt. GitHub användes för versionshantering i syfte att

säkerställa ett effektivt arbetsflöde. För att simulera en databas för webbplatsen användes containerisering med PostgreSQL-images som kördes lokalt.

3.3 Design och jämförande analys

En iterativ process har använts för att bygga inlärningsflödet och användargränssnittet. Ett flertal olika designprototyper skapades och presenterades för projektansvariga: handledare Annamaria Gabrielli, tekniklektor inom nautiska studier Daniel Ernstsson samt handledaren för databas-API:n John J. Camilleri. Utifrån den mottagna feedbacken förbättrades designen stegvis, samtidigt som analyser av liknande lärplattformar genomfördes för att identifiera effektiva sätt att presentera information. Genom denna kombination av feedback och jämförande analys, där olika lösningar prövades, växte webbapplikationens slutgiltiga utseende fram.

4. Systemkonstruktion

Följande avsnitt kommer att beskriva designen av webbapplikation samt den bakomliggande systemkonstruktionen.

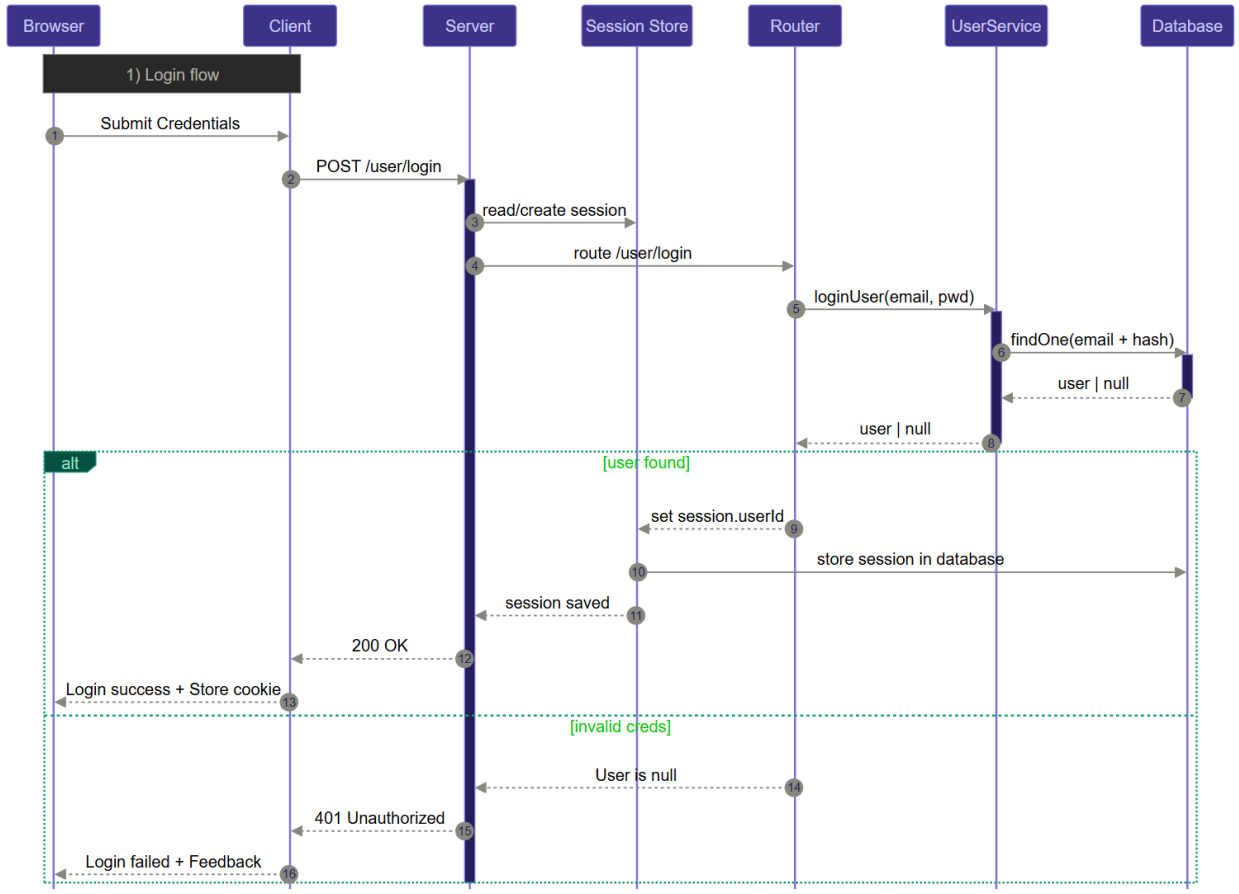
4.1 Systemdesign och arkitektur

Webbapplikationen använder klient-server-modellen och är uppbyggd i tre större komponenter: Klient (Frontend), Server (Backend), Databas.

4.1.1 Övergripande arkitektur

Klienten innefattar främst den visuella designen av webbsidan. Component-mappen innehåller mindre komponenter som återanvänds på vissa sidor och har som syfte att minska repetitiv kod. I klienten används data från databasen som den inte vet exakt hur den är strukturerad. Därav finns interface-mappen som definierar hur datan är strukturerad i databasen och fungerar som en modell för hur informationen ska hanteras i applikationen. Service-mappen hanterar kommunikationen mellan klienten och servern när klienten behöver utföra handlingar på servern. State-mappen används för att hantera inloggning och autentisering samt förhindra användare från att komma åt sidor som kräver inloggning. Slutligen finns en mapp för alla vyer (views) där strukturen definieras i HTML, samt en CSS-mapp som ansvarar för webbplatsens design och utseende.

I servern hanteras data som omvandlas till JSON-format innan den skickas till klienten samt definitioner för hur datan i databasen är strukturerad. Backenden hanterar tjänster och ansvarar främst för att dirigera förfrågningar och svar mellan klienten och databasen. Servern är slutligen den delen av programmet som upprättar en anslutning mellan databasen och sig själv.



Figur 1. UML sekvensdiagram

Figur 1 visar hur flödet mellan användaren och databasen fungerar. I flödet ovanför försöker användaren logga in med angivna uppgifter. Därefter skickas en begäran från klienten till servern där en session skapas i förväg. Routern i servern dirigerar användarens begäran tillsammans med inloggningsuppgifterna vidare till användartjänsten som kontrollerar i databasen om användaren existerar. Tjänsten svarar antingen med användaren eller datatypen “null” beroende på om användaren hittas eller inte. Om användaren hittas kopplas sessionen till användarens ID och servern svarar klienten med att inloggningen lyckades (200 OK). Om användaren däremot inte hittas skickar servern istället ett obehörigt svar (401 Unauthorized), vilket informerar användaren om att inloggningen misslyckades.

4.1.2 Databasmodellering

Ett av projektets mål var att lärare ska kunna skapa förutsättningar och sätta mål för studenter. För att stödja detta använder webbapplikationen en separat databas som är oberoende av databas-API:n. Datan som webbapplikationen använder och lagrar är uppdelad i entiteterna Users, Classrooms, Enrollments, Sections, StudentSections och Goals. Tillsammans beskriver dessa användare, klassrum, kursinnehåll och studenternas utveckling genom systemet.

User representerar både lärare och elever och fungerar som den centrala modellen för autentisering och användarhantering. Tabellen lagrar bland annat e-postadress, lösenord och flaggan “inClassroom”. Flaggan markerar om en student tillhör ett klassrum eller inte. E-postadressen är unik och valideras som en giltig adress för att säkerställa korrekt användaridentifiering. Classrooms beskriver de klassrum som skapas av användare och innehåller bland annat klassrummets namn, användarens id via “teacherId” samt en unik “joinCode” som används när elever ansluter till klassrummet. För att koppla elever till klassrum används Enrollments vilket fungerar som en associationslista mellan användare och klassrum genom attributen “studentId” och “classroomId”.

Kursmaterialet organiseras genom Sections, som beskriver olika delar av innehållet och innehåller både namn och kurstyp. Kurstyperna “basic” och “advanced” används för att kategorisera innehållet. För att följa studenternas progression används StudentSections. Tabellen kopplar en elev till ett avsnitt och lagrar statusvärden av antingen “in_progress” eller “completed”. Slutligen används Goals för att koppla samman klassrum och kursinnehåll. Tabellen innehåller målens namn samt referenser till både klassrum och avsnitt tillsammans med kurstyp. På så sätt kan lärare skapa mål kopplade till specifika delar av kursmaterialet och följa studenternas progression inom dessa områden.

4.2 Design av Frontend

Frontenden för denna webbapplikation är utvecklad med fokus på modularitet, robusthet och skalbarhet. Applikationen är byggd i React, vilket underlättar utvecklingen av användargränssnitt genom ramverkets komponentbaserade arkitektur och omfattande bibliotek. Designen är därför utformad för att tydligt separera olika ansvarsområden, vilket återspeglas i projektets logiska struktur som helhet.

4.2.1 Projektstruktur

Källkoden av frontenden är organiserad i en huvudmapp (client) som innehåller separata kataloger för källkod (src) och statiska resurser (public). Denna huvudmapp följer de konventioner som bestäms av “Create React App” och har en struktur som logiskt separerar olika funktionalitet. Detta hjälper till med att förbättra kodens läsbarhet och förståelse för vidareutveckling. Katalogen src är den primära mappen för källkoden och kan organiseras efter följande:

- **components/:** Denna mapp innehåller återanvändbara komponenter till användargränssnitt, vilket används i olika delar av applikationen. Genom att bryta ut vanliga UI-element, såsom knappar, till egna komponenter uppnås både ett mer visuellt konsekvent gränssnitt och en högre grad av kodåteranvändning.
- **views/:** Denna mapp innehåller en högre nivå av komponenter som representerar olika sidor eller vyer i webapplikationen, såsom startsidan, inloggningssidan och användarprofilen. Dessa vykomponenter ansvarar för att bygga upp den övergripande strukturen för en sida vilket görs genom att sätta ihop mindre, mer detaljerade komponenter från components.
- **services/:** Denna mapp innehåller tjänster som ansvarar för att hantera kommunikationen mellan klienten och backend-servern. Moduler i denna katalog ansvarar för att göra HTTP-förfrågningar till REST-API:et. Denna uppdelning isolerar datahämtning från UI:t, vilket gör komponenterna mer fokuserade på presentation.

- **state/**: Denna mapp används för att definiera och hantera applikationens globala tillstånd. Den innehåller implementationer av Reacts Context API, såsom AuthContext.js för att dela tillstånd över hela applikationen utan att behöva skicka egenskaper genom flera komponentlager.
- **css/**: Denna mapp innehåller Cascading Style Sheets (CSS) som används för att bestämma applikationens utseende. Genom att separera styling och komponentlogik säkerställs en tydlig åtskillnad mellan struktur och presentation av UI:t.
- **interfaces/**: Denna mapp innehåller definitioner för TypeScript-gränssnitt. Dessa filer definierar de datastrukturer som används i hela applikationen och möjliggör statisk typkontroll i TypeScript, vilket bidrar till ökad dataintegritet och minskar risken för typrelaterade fel.
- **App.js/**: App.js-filen fungerar som en central router för hela React-applikationen. Den använder React Router för att definiera applikationens navigeringsstruktur och mappa URL-sökvägar till specifika vykomponenter. Den omsluter alla rutter inom kontextleverantörer, såsom AuthProvider och DataProvider, vilket gör globala tillstånd som användarautentisering och kursdata tillgängliga för alla underordnade komponenter.

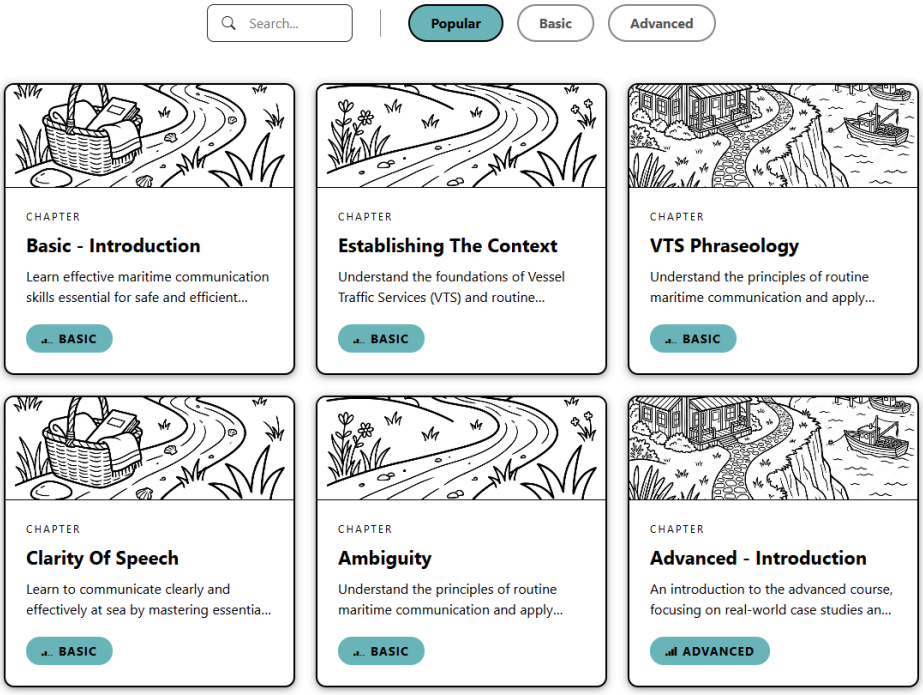
4.2.2 Centrala komponenter

Användargränssnittet är uppbyggt av vyer som är centrala React-komponenter, var och en med distinkta ansvarsområden som bidrar till applikationens övergripande funktionalitet. Den modulära karaktären hos dessa komponenter visar en tydlig separation av ansvarsområdena och belyser återanvändbarheten av kod.

4.2.2.1 HomePage.jsx

HomePage-komponenten fungerar som applikationens huvudsakliga framsida. Den ansvarar för att presentera en översikt över tillgängligt utbildningsmaterial och vägleder användaren till relevant innehåll. Hemsidan består av flera mindre komponenter, inklusive en startsida för ett välkomstmeddelande, en introduktionssektion och en kurslista som visar tillgängliga kurser. Den interagerar med DataContext för att hämta

och visa kursinformation och inkluderar ett kategori-fält som låter användare filtrera de visade kurserna efter kategori (t.ex. "Populär", "Grundläggande", "Avancerad") eller söka efter specifika ämnen, vilket ger en dynamisk och interaktiv användarupplevelse.



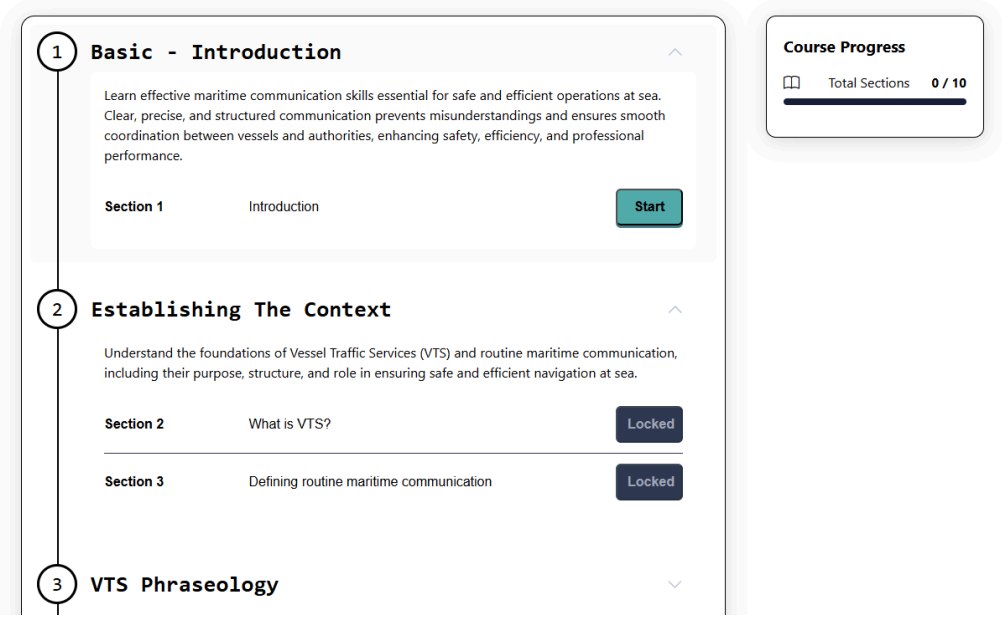
Figur 2. Tillgängliga kurser på startsidan

4.2.2.2 NavBar.jsx

NavBar-komponenten fungerar som programmets primära navigeringselement. Den visas högst upp i användargränssnittet och ger åtkomst till webbplatsens huvudavsnitt, såsom "Learn", "Practice" och "About Us". Komponentens tillstånd är dynamiskt vilket innebär att den visar olika åtgärder baserat på användarens autentiseringsstatus. För autentiserade användare visar den en profilikon och en utloggningsknapp, medan den för oautentiserade användare visar en "Sign-in"-knapp.

4.2.2.3 Course Structure and Progress Tracking

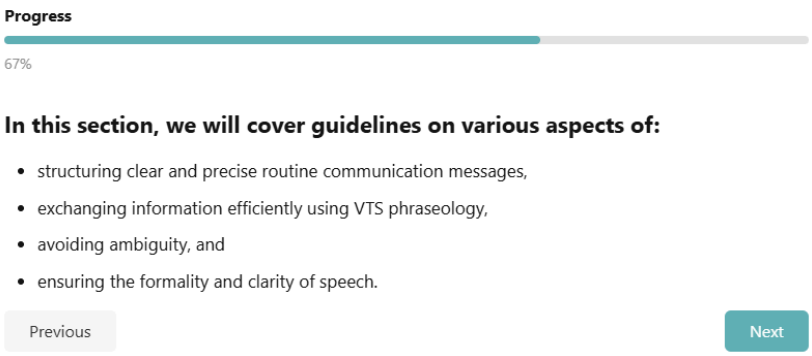
Applikationens utbildningsinnehåll är organiserad i två kurstyper: **Grundläggande** och **Avancerad**. Varje kurstyp presenteras på en egen dedikerad sida, BasicContentPage respektive AdvancedContentPage. Dessa sidor gör det möjligt för studenter att få tillgång till läromaterialet, som är organiserat i en serie kapitel och avsnitt. Båda sidorna har en översiktlig beskrivning av kursen och en knapp för att starta eller fortsätta kursen, som navigerar användaren till det första tillgängliga avsnittet. Studentens framsteg är en central del av lärande-upplevelsen och hanteras gemensamt i systemet. Detta sammanhang ansvarar för att hämta, lagra och uppdatera statusen för varje avsnitt för den inloggade studenten. När BasicContentPage eller AdvancedContentPage laddas, hämtas den studentens framstegsdata från detta sammanhang. Data som visar om varje avsnitt är **avklarat** eller **pågående** kombineras sedan med information om kursens innehåll. Detta gör att användargränssnittet kan visuellt representera studentens framsteg, till exempel genom att visa bockmarkeringar bredvid slutförda avsnitt eller markera nästa avsnitt som ska slutföras. När en elev avslutar ett avsnitt anropas en funktion som uppdaterar backend och sedan uppdaterar framstegsdata, vilket säkerställer att användargränssnittet alltid är synkroniserat med elevens senaste prestationer.



Figur 3. Användarens framsteg i BasicContentPage

4.2.2.4 ContentPage.jsx

ContentPage-komponenten är sidan där själva inlärningen sker och är utformad för att visa materialet för ett specifikt avsnitt. Denna komponent använder React Router för att avgöra vilken kurs och avsnitt som ska renderas baserat på URL:en. Den hämtar motsvarande bilder och interaktiva element, såsom “slides”, från backenden via ett service-anrop. Anropet är kopplat till en service som i sin tur kommunicerar med den externa API-databasen för att hämta läromaterial. Backenden tar emot informationen och strukturerar om den till JSON-format, som sedan skickas till och presenteras på frontenden. ContentPage hanterar användarens framsteg genom slidsen, spårar den aktuella sliden och eventuella inskickade svar. Den innehåller också en sidomeny för att navigera mellan olika avsnitt och en mätare för att ge användaren visuell feedback om deras framsteg.



Figur 4. Upplägg på material i ContentPage

4.2.2.5 ClassRoomPage.jsx

ClassroomPage-komponenten fungerar som en instrumentpanel som anpassar sitt gränssnitt beroende på om användaren har själv skapat klassrummet eller om användaren gått med i ett klassrumm. För användare som inte tillhör ett klassrum visas vyn CreateOrJoinClass, som innehåller alternativ för att antingen skapa ett nytt klassrum eller gå med i ett befintligt. För användare som gått med i ett klassrum, visas vyn StudentDashboard, ett enkelt gränssnitt som bekräftar deras medlemskap och erbjuder en funktion för att lämna klassrummet. För användaren som har skapat klassrummet visas vyn TeacherDashboard, ett mer omfattande gränssnitt som är uppdelat i tre olika flikar för att hantera klassrum. Den första fliken, “Stream”, fungerar som klassrummets startsida. Där visas bland annat klassrummets namn, en unik kod som elever kan använda för att gå med och antalet studenter i klassrummet. Fliken “People” visar en lista över alla elever som är inskrivna i klassrummet, vilket gör det enkelt för läraren att hålla koll på vilka som deltar. Den sista fliken, “Classwork”, är den centrala delen för undervisningen. Där kan läraren organisera material, skapa uppgifter och följa elevernas arbete och progression genom kursinnehållet.

4.2.2.6 Goals

Funktionen Goals, som huvudsakligen hanteras på fliken “Classwork” i TeacherDashboard, fungerar som en central komponent för att sätta och följa upp

akademiska mål för elever. Lärare kan skapa nya mål, vilket definieras av ett namn, ett specifikt kursavsnitt och en kurstyp (grundläggande eller avancerad). Dessa mål listas sedan i en GoalList-komponent. När en lärare väljer ett mål visas komponenten “GoalProgress”, som ger en detaljerad vy över alla elevers framsteg mot det specifika målet. Denna vy visar vilka elever som har slutfört det associerade avsnittet och vilka som fortfarande pågår, och den innehåller filter för att begränsa listan efter elevens e-postadress eller slutförandestatus. Hela funktionen drivs av GoalService, som hanterar all kommunikation med backend för att skapa, hämta och hantera mål.

4.2.3 State Management

För att hantera applikationens olika tillstånd på ett skalbart och hållbart sätt används Reacts Context API. Detta gör det enklare att dela tillstånd mellan komponenter i applikationen och eliminerar behovet av att skicka data genom flera komponenter i kedjan.

4.2.3.1 AuthContext

AuthContext är den viktigaste implementationen för applikationens mönster att hantera olika tillstånd. Den fungerar som en central enhet för autentiseringsrelaterade tillstånd, såsom användarens inloggningsstatus, profilinformation och funktioner som kräver inloggning. AuthContext hanterar även utloggningsprocedurer och ser till att rätt information och funktioner är tillgängliga beroende på om användaren är inloggad eller inte. Komponenter som förlitar sig på den här typen av information, som till exempel Navbar och route-skyddade mekanismer, kan använda tillståndet och uppdateras automatiskt med aktuell information när något förändras. Detta säkerställer att hela användargränssnittet är synkroniserat med användarens autentiseringsstatus.

4.2.3.2 DataContext och StudentProgressContext

DataContext och StudentProgressContext används för att hantera andra former av globala tillstånd. DataContext kan t.ex. användas för att cachelagra data som hämtas från servern, såsom kurslistor, vilket förhindrar repetitiva API-anrop och förbättrar applikationsprestanda. Genom att samla hanteringen av delat tillstånd på ett ställe

hjälper Context API till att göra applikationen mer strukturerad, förutsägbar och enklare att felsöka.

4.2.4 Responsive Design

Gränssnittet på frontend är utformat för att vara responsiv vilket skapar en bra användarupplevelse på en mängd olika enheter, från stationära bildskärmar till mobiltelefoner. Genom att använda breakpoints och media queries, tillsammans med flexibla layouter med hjälp av Flexbox och Grid, kan webbplatsen anpassas efter olika skärmstorlekar och enheter. Viktiga komponenter, som t.ex. NavBar.jsx justerar dynamiskt sin layout baserat på skärmbredden. På större skärmar visar NavBar samtliga navigeringslänkar, medan den på mindre skärmar (1024px eller mindre) ersätts av en kompakt vy med en hamburgermenyknapp som visar eller döljer navigeringslänkarna. Detta säkerställer att navigeringen blir tillgänglig även med det begränsade skärmutrymmet på mobila enheter.

4.3 Design av Backend

4.3.1 Projektstruktur

Servern, även kallad backend, har som syfte att kommunicera data mellan klienten och databasen. Backenden definierar datan med hjälp av verktyget Sequelize, vilket kommer att beskrivas mer i detalj under kommande avsnitt. För att data ska vara användbar krävs det att en given logik och struktur existerar så att datan kan ta sig från klient till server respektive andra hållet. När backenden startar ansluter den samtidigt till databasen. Dataflödet är uppdelat i tre huvudfunktioner, strukturerat under tre mappar:
Interfaces för metoder (serviceModel), Implementation av metoder (services), Endpoints som dirigerar data (routers).

Interfacen för metoderna beskriver övergripande vad respektive metod ska utföra. I detta fall definierar de vilka parametrar funktionerna tar emot samt vilka värden de returnerar. Dessa interface ligger till grund för logiken kring dataflödet i applikationen

och implementeras senare för att säkerställa en tydlig samt konsekvent implementering av metoder i systemet.

```typescript
import { User } from "../../db/user.db";

export interface IUserService {

    // POST
    signupUser(email: string, password: string): Promise<User>

    // POST
    loginUser(email: string, password: string): Promise<User | null>

    // PATCH
    editProfile(id: number, email: string, password: string): Promise<User | null>

    // GET
    getUser(id: number): Promise<User | null>

    // VOID
    logoutUser(): Promise<void>
}
```

Figur 5. Exempel på ett interface

Figur 5 visar ett exempel på hur ett interface är definierat. Alla metoder har ett “Promise” som säger vad metoden ska returnera för datatyp samt parametrar beroende på vad metoden har för syfte. Noterat ovanför varje metod står det också vilken typ av HTTP-förfrågan varje metod förväntas vara associerad med. Ett undantag i figur 5 är utloggningen, som är kopplad till en metod med returtypen void – det vill säga att den inte returnerar något värde. Metodernas specifikation kan alltid ändras om andra behov skulle uppstå.

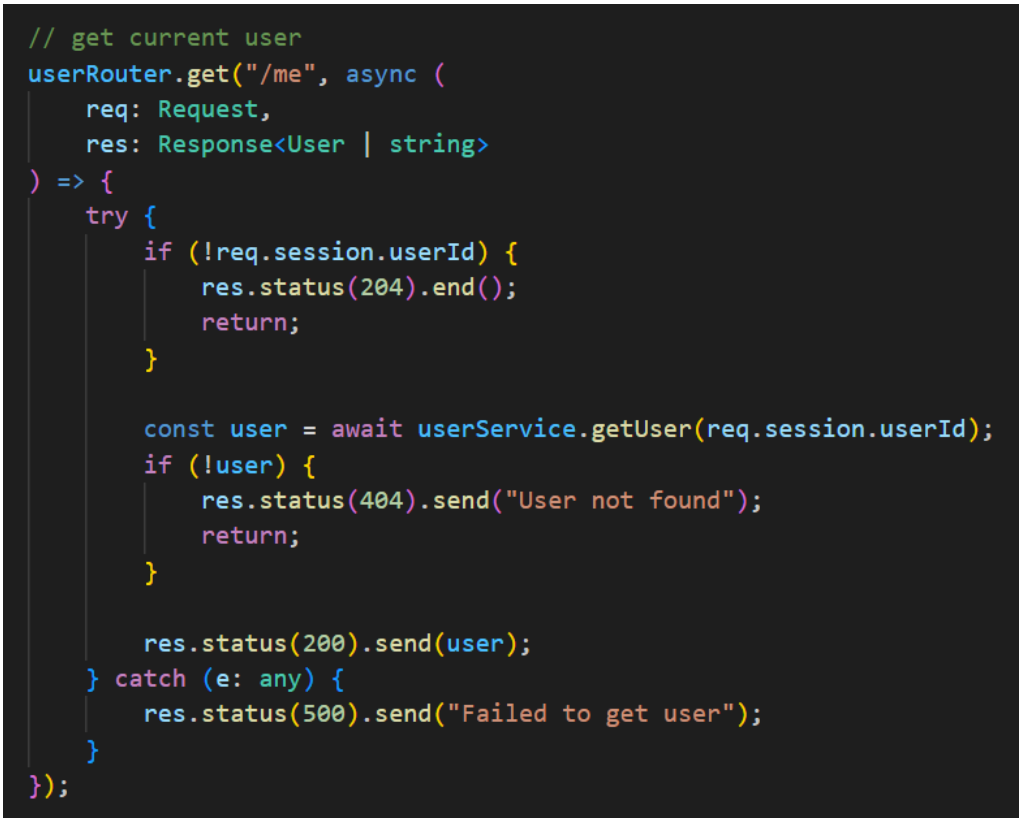
Själva implementeringen av metoderna sker i service-filerna, exempelvis UserService för användarhantering och ClassroomService för hantering av klassrum. Dessa filer implementerar respektive tillhörande interface och ansvarar för applikationens logik kopplad till datan. Databasen som används är en PostgreSQL-instans och är oberoende från API-databasen som hämtar läromaterial.

```
async loginUser(email: string, password: string): Promise<User | null> {
    const hashedPassword = await this.generateSha256Hash(password)
    return await User.findOne({ where: {email: email, password: hashedPassword }})
}
```

Figur 6. Implementation av inloggning

Figur 6 visar hur inloggningen sker via kommunikation med databasen. Lösenord lagras inte i klartext i databasen utan hashas via **SHA256 algoritmen** innan det lagras. När en användare försöker logga in hashas det angivna lösenordet och jämförs med det lagrade värdet i databasen. Om uppgifterna stämmer returneras användaren från databasen, annars null, vilket innebär att inloggningen misslyckades.

Metoderna används i applikationens routers där logiken för datahanteringen styrs genom API-endpoints. Dessa endpoints är kopplade till olika sökvägar och hanterar specifika HTTP-förfrågningar, exempelvis GET, PATCH och DELETE. Varje endpoint kan konfigureras för att hantera förfrågningar och svar beroende på dess syfte.



```
// get current user
userRouter.get("/me", async (
    req: Request,
    res: Response<User | string>
) => {
    try {
        if (!req.session.userId) {
            res.status(204).end();
            return;
        }

        const user = await userService.getUser(req.session.userId);
        if (!user) {
            res.status(404).send("User not found");
            return;
        }

        res.status(200).send(user);
    } catch (e: any) {
        res.status(500).send("Failed to get user");
    }
});
```

Figur 7. Exempel på routing av data

Figur 7 visar hur en användare hämtas med hjälp av en GET-request. Routerns uppgift är att anropa relevanta services som är kopplade till varje endpoint. Routern ansvarar även för hantering av olika felfall, exempelvis om användaren saknar en aktiv session, om användaren inte finns i databasen eller om ett generellt serverfel uppstår. Detta görs för att säkerställa att fel hanteras korrekt och inte förbises.

4.3.2 Autentisering och sessioner

Sessioner används för att koppla den inloggade användaren till webbplatsen. När klienten skickar giltiga inloggningsuppgifter till routern och autentiseringen lyckas tilldelas användaren en session kopplad till sitt användar-id. Sessionen lagras i databasen genom den Sequelize-baserade lösningen “session.Store”. Användarspecifika endpoints förlitar sig därefter på sessionen för att identifiera vilken användare som skickar förfrågningar. När användaren loggar ut tar backenden bort sessionen i databasen, vilket bryter kopplingen mellan webbläsaren och användaren.

4.3.3 Datahantering

Kommunikation med databasen går genom en Sequelize-instans som konfigureras med dialect, host, användare och lösenord. Instansen använder en anslutningspool och kör parameteriserade PostgreSQL-kommandon och utför antingen SQL som genereras av Sequelize eller råa frågor via sequelize.query(). Under services definieras applikationens logik tillsammans med databasanrop. Sequelize översätter dessa anrop till SQL-queries som körs mot databasen och mappar därefter resultaten tillbaka till modellinstanser som klienten kan använda.

```
Classroom.belongsTo(User, { foreignKey: 'teacherId', onDelete: 'CASCADE' })
User.hasOne(Classroom, { foreignKey: 'teacherId' })
```

Figur 8. Exempel på hur associationer ser ut

Databasen för webbsidan är en relationsdatabas där varje Sequelize-modell motsvarar en tabell. Varje modell definierar kolumner, datatyper, valideringar, default-värden och associationer. Dessa associationer skapas när modeller kopplas samman i databasen, t.ex. med hasMany, belongsTo och belongsToMany. Associationen i Figur 8 är konfigurerad med kaskadering (CASCADE). Det innebär att en radering av en rad i tabellen “User” triggar en automatisk radering av alla matchande rader i “Classroom” som refererar till användaren via foreign key “teacherId”.

4.4 Docker-konfiguration

Detta projekt utnyttjar containerisering med Docker för att ge portabilitet mellan olika miljöer. En Dockerfile definieras därför i projektet, där applikationen byggs upp genom flera separata steg.

4.4.1 Multi-stage build:

Multi-stage build är ett sätt att optimera Docker images genom att använda flera byggsteg inuti en Dockerfile. Varje steg börjar med en FROM-instruktion och kan använda en annan “image”. Fördelen med den här metoden är möjligheten att separera

byggmiljön från den slutliga produktionsmiljön, vilket resulterar i ett mer minimalt och säkert resultat.

client-builder-stage: Det första steget ansvarar för att bygga React-gränssnittet. Det bygger på en lätt version av Node.js som redan innehåller allt som behövs för att köra programmet och installera paket. I det här steget installeras alla filer och paket som klientsidan behöver för att fungera, och sedan körs kommandot "`npm run build`" för att skapa en färdig version av webbapplikationen. Denna process omvandlar React- och TypeScript-koden till en uppsättning optimerade, statiska HTML-, CSS- och JavaScript-filer som är lämpliga för produktion.

server-builder-stage: Det andra steget förbereder backend-delen av applikationen. Här installeras de paket som servern behöver för att köras, och serverns källkod kopieras in så att applikationen kan starta. Det här steget skapar en färdig och fristående version av serverapplikationen som innehåller allt som behövs för att den ska kunna köras.

final-production-stage: Det sista steget skapar den färdiga versionen av applikationen som är redo att användas i produktion. Det börjar med en ren miljö för att hålla applikationen snabb och minimera onödiga filer. Istället för att replikera byggprocesserna kopierar det här steget selektivt endast de nödvändiga delar från de föregående stegen med hjälp av instruktionen COPY --from. Mer specifikt kopieras de färdiga filerna från klientsidan samt serverns programkod och nödvändiga paket från tidigare steg i byggprocessen.

5. Diskussion och slutsatser

Någon direkt feedback från sjöfartsstudenter samlades inte in under projektets gång. Däremot mottogs designen positivt av Gabrielli, Ernstsson och Camilleri, vilka alla gav återkoppling på såväl den visuella som den systemmässiga designen.

5.1 Diskussion

När vi reflekterar över examensarbetet, uppfyller utvecklingsprocessen de primära mål som angavs. Det första målet var att skapa ett intuitivt och engagerande användargränssnitt för sjöfartsstudenter, och den resulterande applikationen uppnår detta genom en ren, komponentbaserad design och responsiv layout. Uppdelningen av innehåll i tydliga element som startsidan, innehållssidan och klassrummet ger en tydlig och logisk användarupplevelse. Det andra målet: “*Material ska kunna hämtas och visas upp via extern API på ett visuellt tilltalande sätt på webbapplikationen*”, nås genom klient-server-arkitekturen, där frontend dynamiskt renderar innehåll som serveras från backend. Detta säkerställer att läromaterialet kan uppdateras och hanteras oberoende av användargränssnittet. Vidare uppfylls det tredje målet genom att ge lärare möjlighet att förbereda eleverna inför kommande kursmoduler, via den implementerade "Mål"-funktionen i klassrummet. Denna funktion gör det möjligt för lärare att sätta specifika lärandemål kopplade till kursavsnitt och övervaka elevernas framsteg, vilket direkt uppfyller kravet. Projektet höll sig också strikt till sina definierade avgränsningar: att begränsa integrering med externa tjänster, att inte utgå från en befintlig lärplattform samt att säkerställa att applikationen inte utvecklades i marknadsföringssyfte. På grund av den begränsade tidsramen hann chatbotten inte integreras, vilket kan betraktas som framtida arbete.

5.2 Utmaningar

En utmaning som stöttes på under utvecklingsprocessen var integrationen med databas-API:n, som utvecklades parallellt av en annan grupp. Även om detta parallella arbetsflöde möjliggjorde samtidigt utveckling på både webbapplikationen och en separat databas, introducerade det också ett lager av komplexitet relaterat till datakonsistens och kommunikation. Skillnader mellan vad webbapplikationens frontend förväntar sig och API:ns databasstruktur var ett återkommande problem. Till exempel ledde skillnader i tabellnamn och formatet för hämtat innehåll ofta till integrationsproblem som krävde felsökning och nära samarbete mellan de två teamen. Även om problemen slutligen löstes genom testing och kommunikation, skulle ett mer formaliserat och strikt tillämpat dataschema samt en API-specifikation från början sannolikt ha effektiviserat integrationsprocessen.

5.3 Framtida utveckling

I följande avsnitt diskuteras den feedback som togs emot från Gabrielli, Ernstsson och Camilleri, vilket kan ligga till grund för vidareutveckling inför nästa projektiteration.

5.3.1 Roller och användarhantering

Det finns ett flertal funktionaliteter och förbättringar som hade kunnat läggas till på webbsidan. Det finns ingen tydlig rollseparering mellan användarna på webbsidan. Den enda skillnaden är att användaren som skapar ett klassrum tekniskt sett betraktas som lärare, vilket kan behöva förtydligas. Som student finns det inte mycket att se när man väl har gått med i ett klassrum, därav finns det rum för utveckling av en studentvy. Ett klassrum kan dessutom endast ha en lärare, nämligen skaparen, vilket begränsar möjligheten för flera lärare att inspektera studenternas arbete inom samma klassrum.

5.3.2 Funktionalitet och användarupplevelse

Möjligheten för användaren att ta bort klassrum och mål implementerades inte, men behövs läggas till för att minska onödig datamängd och skapa en bättre användarupplevelse. Navigeringen mellan alla slides är för närvarande inte så restriktiv

och hade behövt mer logik bakom sig som hindrar studenter från att skippa till slutet av ett avsnitt. För att ytterligare förbättra användningen av målen hade chatbotten kunnat integreras i mitten eller i slutet av ett avsnitt. Det hade både ökat interaktiviteten samt stärkt kontrollen över studenternas progression i ämnet. Feedbacken vid misslyckad inloggning samt de designmässiga skillnaderna mellan inloggning och registrering hade båda förbättringspotential, vilket blev tydligt under demonstration av webbapplikationen.

5.3.3 Buggar och tekniska problem

En frekvent bugg som återkom var när användaren försökte komma åt läsmaterialet utan en session. Förfrågan resulterade i ett svar med statuskod 401 till klienten, men efter upprepade försök att komma åt innehållet blev klienten istället blockerad från webbplatsen, vilket inte var avsiktligt.

5.3.4 Innehåll och informationssidor

Bortsett från funktionaliteter är det också tänkt att lägga till mer innehåll under “Publications” och “About Us”, förslagsvis mer gällande källmaterialet respektive en formell introduktion till D-Smart.

5.4 Slutsatser

Det slutgiltiga programmet uppfyller samtliga tidigare definierade systemkrav vilket innebär att projektets mål har uppnåtts, även om det fortfarande finns viss förbättringspotential. Designen av webbsidan var väl uppskattad av handledare samt tekniklektor och all feedback som gavs kommer ligga till stor grund inför flera vidareutvecklingar av projektet. Webbplatsen kan hämta läromaterial via ett externt API och därefter presentera innehållet på ett användarvänligt och lättillgängligt sätt. Med hjälp av klassrum och mål kan lärare samla sina studenter på en gemensam plats och skapa förutsättningar utifrån det aktuella läromaterialet.

Genom användningen av en tydlig klient-server-arkitektur skapades en modulär struktur där frontend, backend och databas separerades på ett effektivt sätt. Detta möjliggör

både bättre underhållbarhet och framtida vidareutveckling av systemet. Trots vissa begränsningar och tekniska problem uppfyllde webbapplikationen de centrala målen för projektet och skapade en stabil grund för fortsatt utveckling. Webbplatsen har mycket utvecklingspotential och kan behövas bearbetas mer på för att faktiskt bli användbar i lärorika syften. Arbetet resulterade i en lösning baserad på moderna webbutvecklingstekniker som innefattar React, TypeScript, Node.js, Express och PostgreSQL. Resultatet visar att digitala verktyg kan bidra till att göra forskningsbaserat utbildningsmaterial inom sjöfart mer tillgängligt, strukturerat och pedagogiskt för både lärare och studenter.

Källförteckning

[1] GeeksforGeeks, “Client-Server Model” 2026. [Online]. Tillgänglig: https://www.geeksforgeeks.org/system-design/client-server-model/ (hämtad: 2026-05-06)

[2] GeeksforGeeks, “REST API Introduction” 2026. [Online]. Tillgänglig: https://www.geeksforgeeks.org/node-js/rest-api-introduction/ (hämtad: 2026-05-07)

[3] Limetta, "React - what is this JavaScript framework" 2026. [Online]. Tillgänglig: https://limetta.se/en/tips-metoder-for-digitala-projekt/what-is-react-javascript-framework (hämtad: 2026-03-11)

[4] MDN, "Document Object Model (DOM)", Mars 2026. [Online]. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (hämtad: 2026-03-24)

[5] W3Schools, “React Router” 2026. [Online]. Tillgänglig: https://www.w3schools.com/react/react_router.asp (hämtad: 2026-05-09)

[6] TypeScript, “TypeScript is JavaScript with syntax for types” 2026. [Online]. Tillgänglig: https://www.typescriptlang.org (hämtad: 2026-05-11)

[7] Umraco, "What is CSS and how is CSS used on websites?" 2026. [Online]. Tillgänglig: https://umbraco.com/knowledge-base/css/ (hämtad: 2026-03-29)

[8] W3Schools, “Node.js Introduction” 2026. [Online]. Tillgänglig: https://www.w3schools.com/nodejs/nodejs_intro.asp#:~:text=What%20is%20Node.js%3F,server-side%20development%20with%20JavaScript (hämtad: 2026-05-09)

[9] Node.js, “Introduction to Node.js” 2026. [Online]. Tillgänglig: https://nodejs.org/learn/getting-started/introduction-to-nodejs (hämtad: 2026-05-09)

[10] Express.js, "Express - Fast, unopinionated, minimalist web framework for Node.js" 2026 [Online]. Tillgänglig: https://expressjs.com (hämtad: 2026-05-09)

[11] W3Schools, “What is JSON?” 2026. [Online]. Tillgänglig: https://www.w3schools.com/whatis/whatis_json.asp (hämtad: 2026-05-09)

[12] PostgreSQL, "PostgreSQL: About” 2026. [Online]. Tillgänglig: https://www.postgresql.org/about/ (hämtad: 2026-05-06)

[13] Sequelize, "Sequelize” 2026. [Online]. Tillgänglig: https://sequelize.org (hämtad: 2026-05-06)

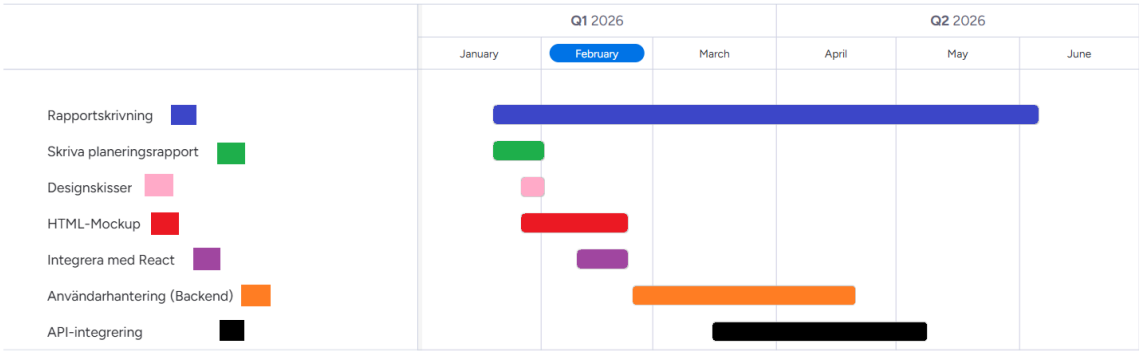
[14] Docker Docks, "What is Docker?” 2026. [Online]. Tillgänglig: https://docs.docker.com/get-started/docker-overview/ (hämtad: 2026-05-06)

[15] Cloudflare, "What is HTTP?” 2026. [Online]. Tillgänglig: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/ (hämtad: 2026-05-06)

[16] Cloudflare, "What is HTTPS?” 2026. [Online]. Tillgänglig: https://www.cloudflare.com/learning/ssl/what-is-https/ (hämtad: 2026-05-06)

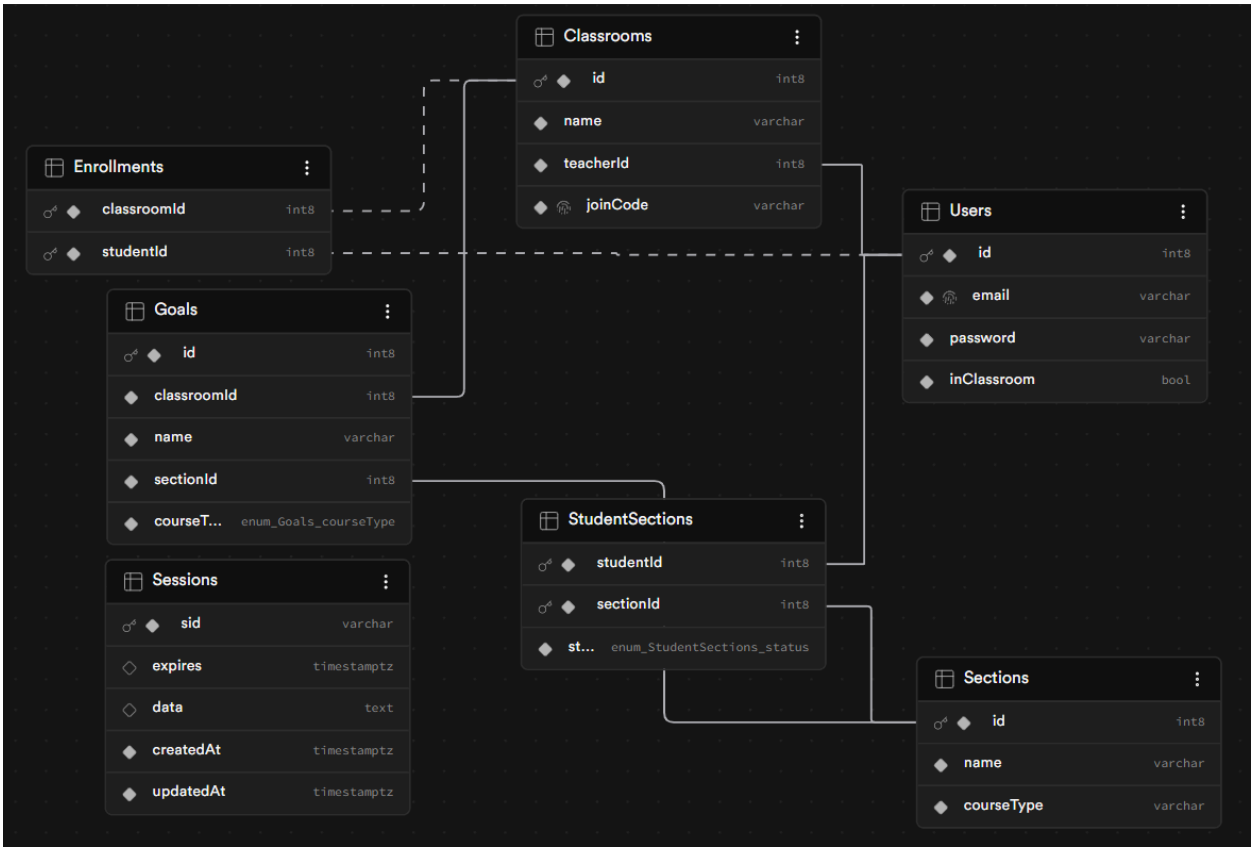
Bilagor

Bilaga 1. Gantt schema



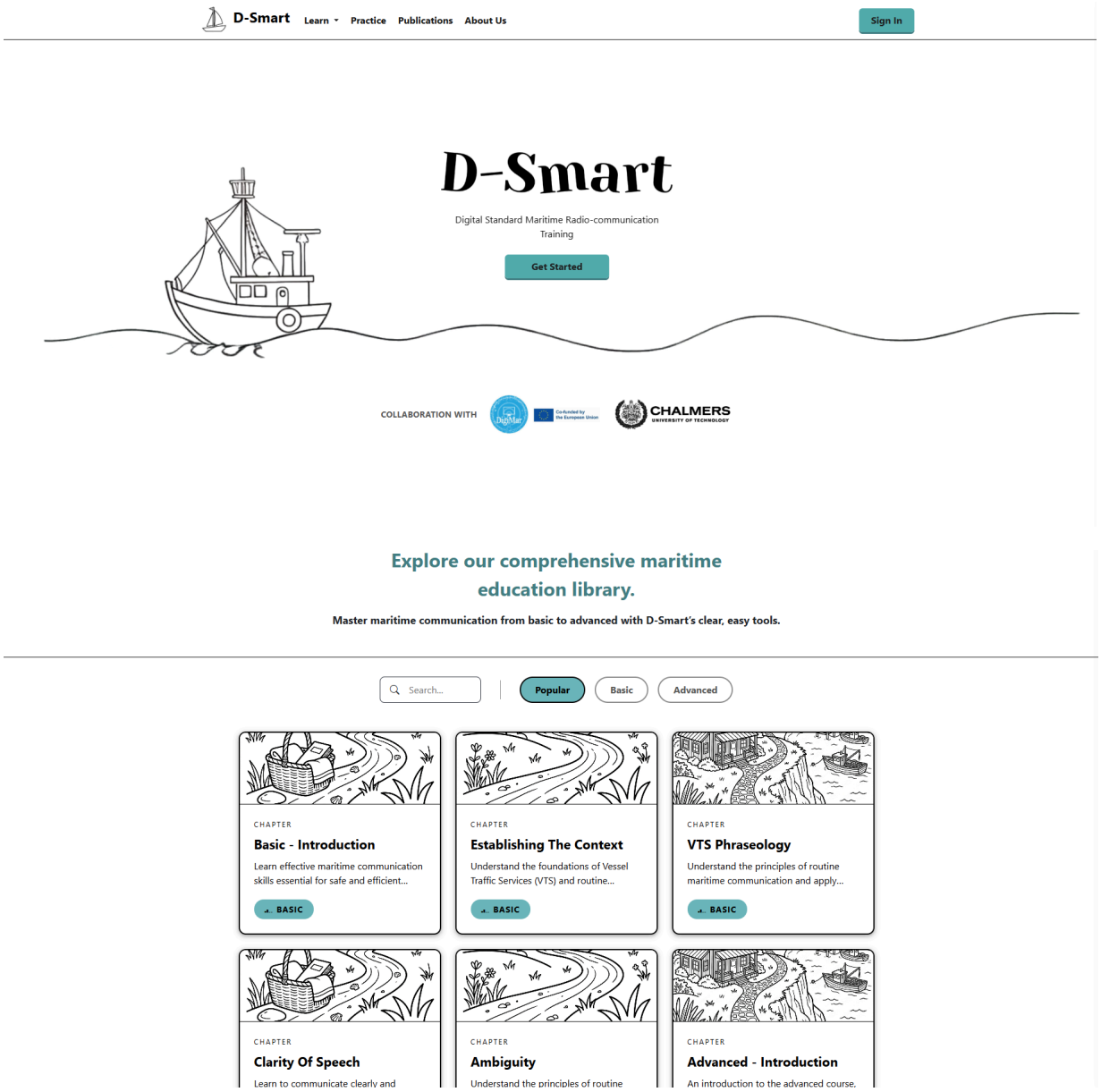
Figuren visar ett Gantt schema som visar vad som ska arbetas med under respektive vecka under projektets gång.

Bilaga 2. Databasschema



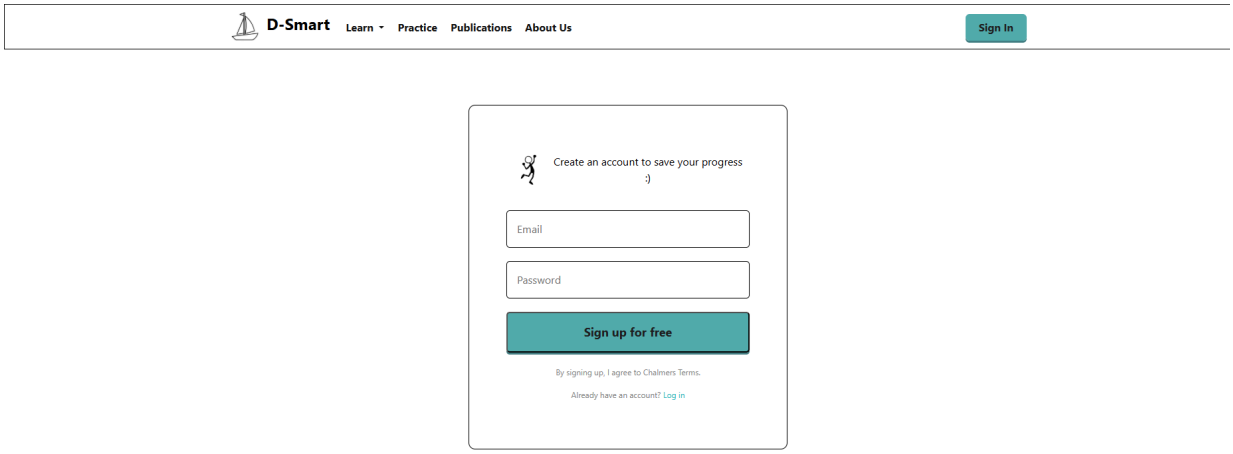
Figuren visar databasens schema och relationer mellan entiteter.

Bilaga 3. Hemsidan



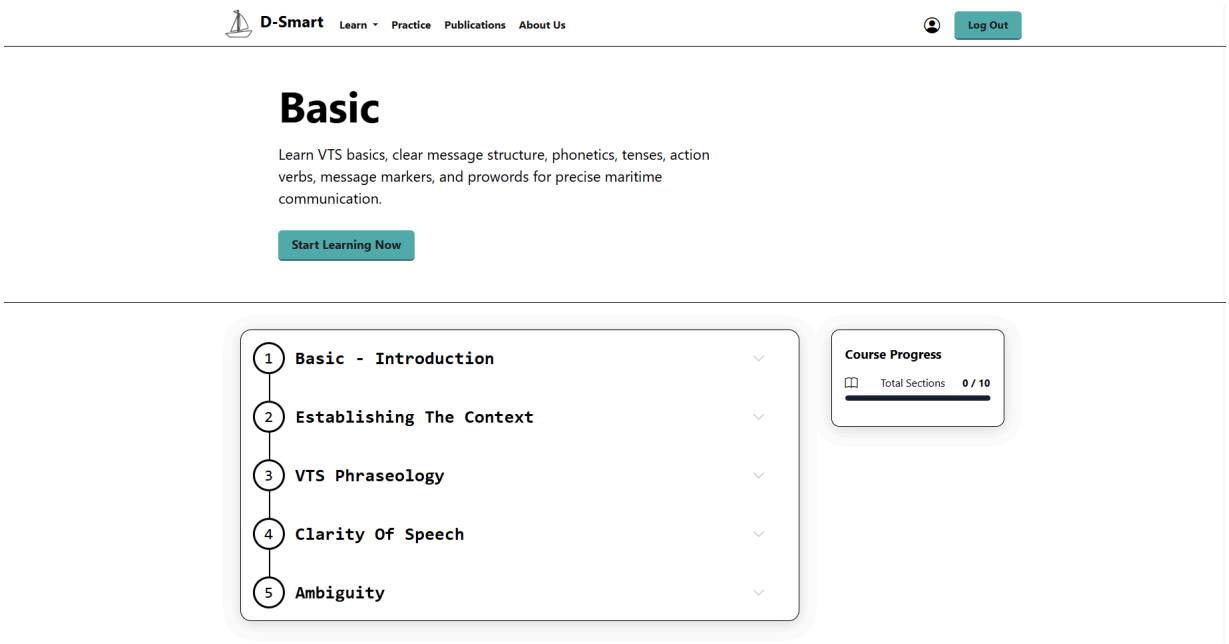
En översikt över programmets hemsida, som visar huvudbannern, ett introduktionsavsnitt och en lista över tillgängliga kurser för studenter.

Bilaga 4. Inloggning/registrering



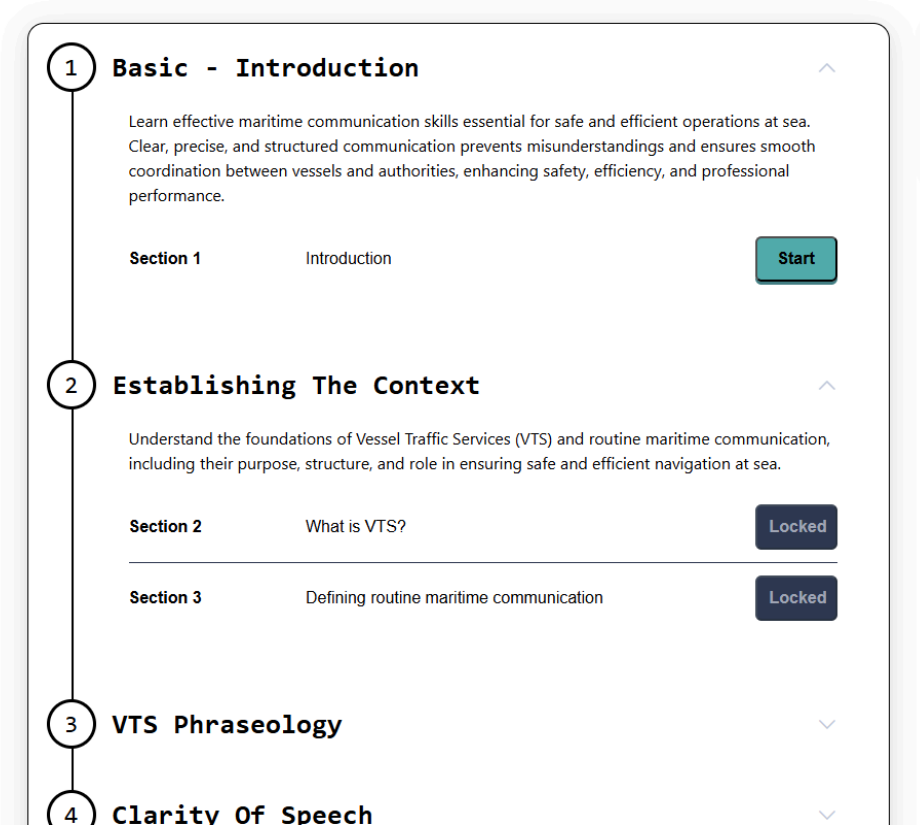
En vy för webbapplikationens registreringssida som nås via navigeringsfältet ovanför. Användaren kan byta mellan att logga in eller registrera sig.

Bilaga 5. Upplägg av kursmaterial Del 1



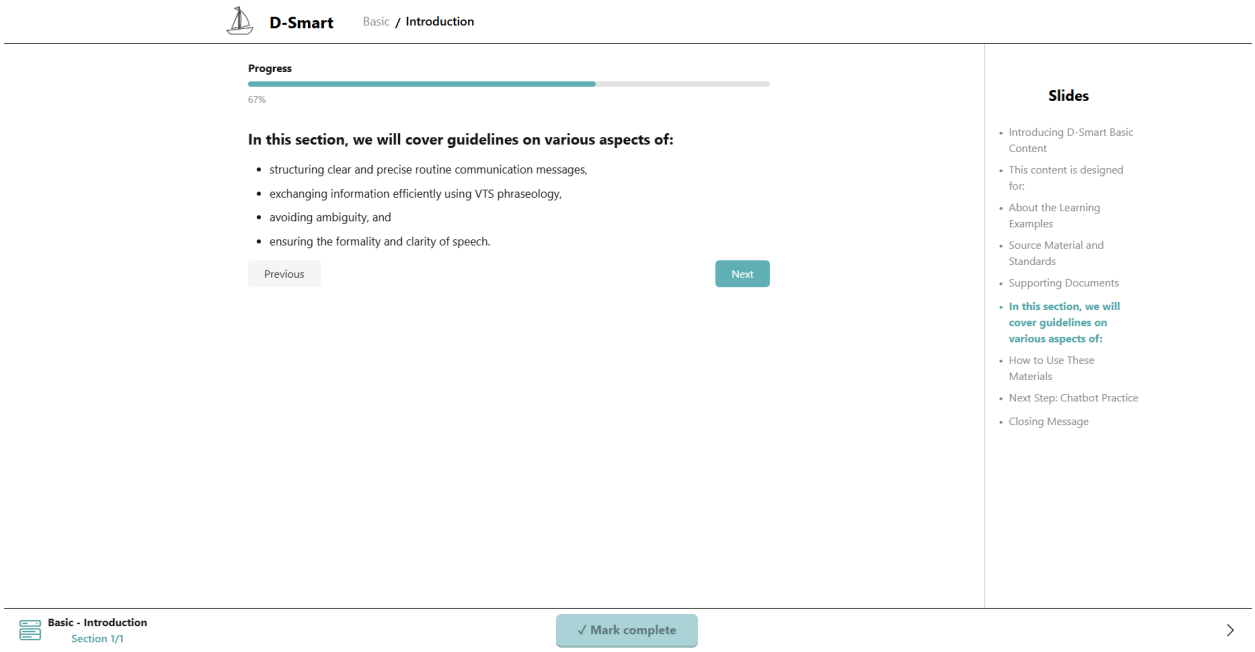
Figuren ger en översikt över kursstrukturen, med en kurs-banner, en lista över kapitel och en förloppsindikator som visar det totala antalet slutförda avsnitt.

Bilaga 6. Upplägg av kursmaterial Del 2



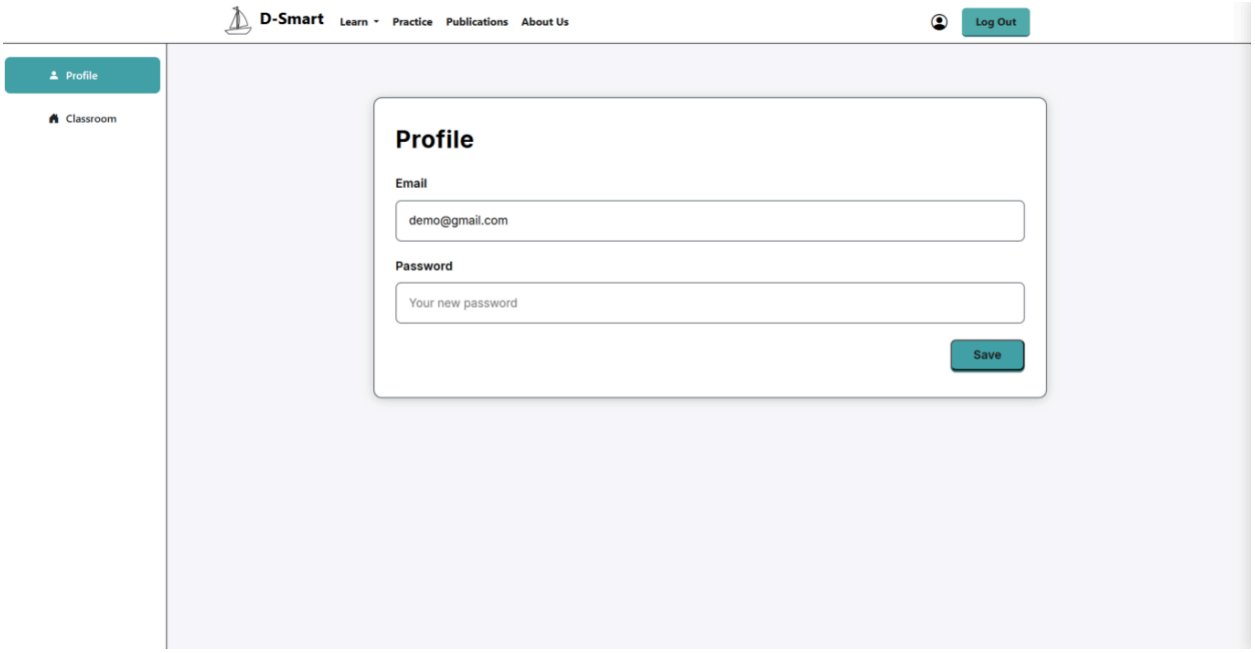
När ett kapitel utökas på innehållssidan kan användare se detaljerade beskrivningar och komma åt enskilda avsnitt. Låsta avsnitt och en "Start"-knapp vägleder studentens inlärning och progression genom kursen.

Bilaga 7. Struktur av kursmaterial



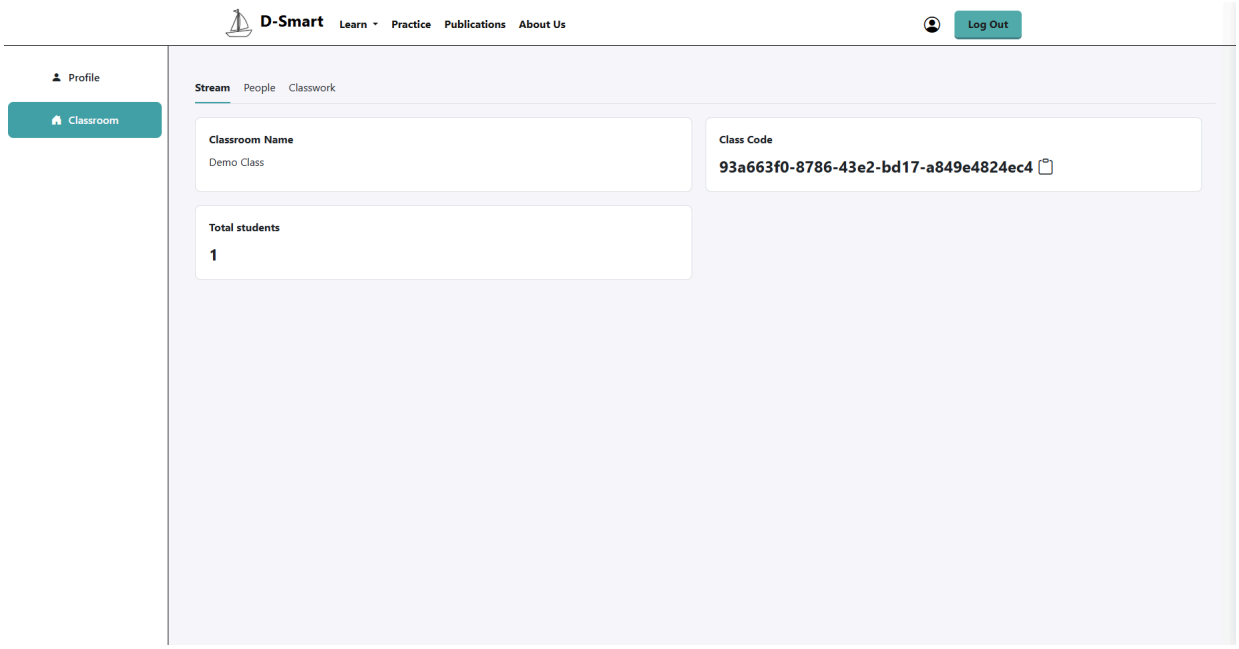
Figuren visar ett interaktivt inlärningsavsnitt. Huvudområdet innehåller instruktionerna för den aktuella bilden, medan sidofältet gör det möjligt för användaren att följa sin position och navigera mellan olika ämnen i avsnittet.

Bilaga 8. Användarprofil



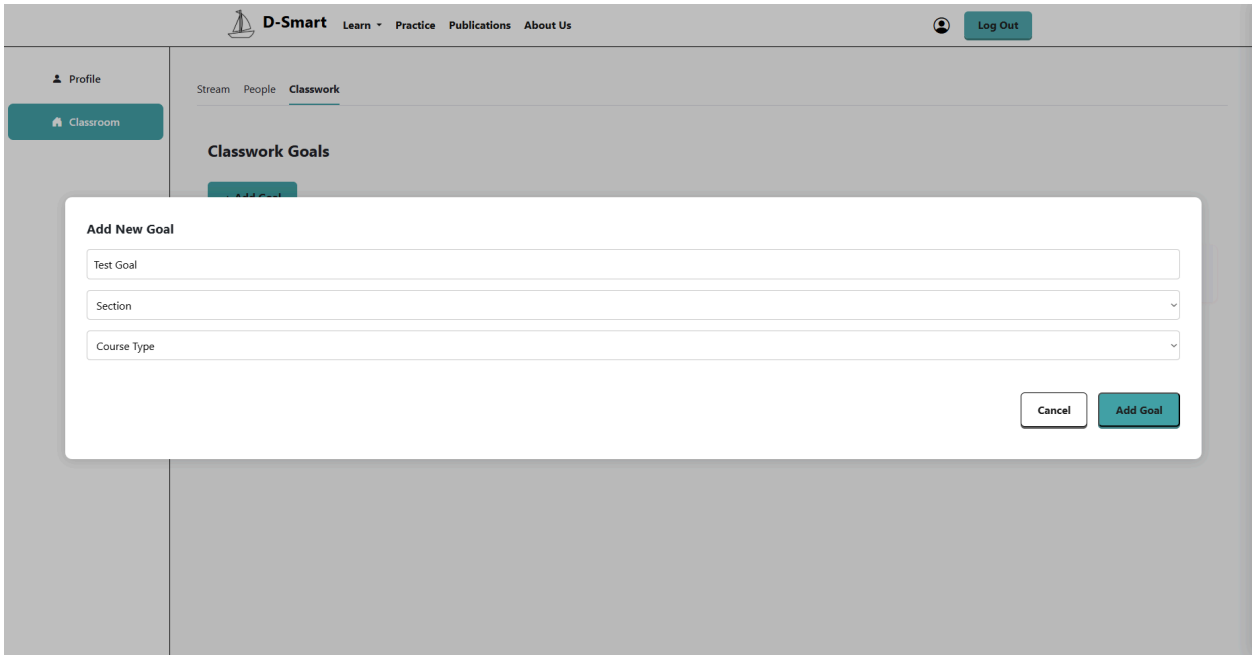
Figuren visar användarens profil där användaren kan ändra sina uppgifter vid behov.

Bilaga 9. Klassrum



Figuren visar en översikt över klassrummet med dess namn, unik klasskod för elevregistrering och det aktuella antalet inskrivna elever.

Bilaga 10. Målsättningar



Fönstret för Mål-skapande i ett klassrum, där en lärare kan definiera ett nytt mål genom att ange dess namn, tillhörande avsnitt och kurstyp innan det läggs till i klassrummet.