Innhold

Figurer

Begreper .

Chatbot = Samtalerobot

Prompt = Ledetekst, instruksjon til ChatGPT

PoC = Proof of Concept

UX = Brukeropplevelse

Tag = Emneknagg

KI = Kunstig Intelligens

NLP = Naturlig Språkbehandling

API = Applikasjons-programmerings-grensesnitt

VCS = Versjonskontrollsystem

ChatGPT = Chat Generative Pre-trained Transformer

NPM = Node Package Manager

Open-source = Fritt-tilgjengelig

Merge = Sammenfletting

Backlog = Prioriteringsliste

Sprints = Jobbing i korte tidsintervaller

Wireframes = Skisser

Fullstack = Utvikling av både front- og backend

Dummy = Kunstig eller fiktiv

Abstract

This project involves the development of an artificial intelligence-driven conversational agent (chatbot) for Sunnmørsposten, a news platform aiming to enhance reader engagement through advanced digital solutions. Focusing on providing readers with a deeper understanding and better navigation of news articles, our chatbot utilizes modern technologies such as OpenAI's API to generate dynamic responses based on user inquiries.

The goal of the chatbot is to summarize, explain complex issues, and suggest relevant articles to improve the user experience. The implementation is designed to fit seamlessly into smp.no, with functionalities supporting universal design to ensure accessibility for all users. Despite technological and security constraints, the project has demonstrated how artificial intelligence can be effectively integrated to meet specific needs within news dissemination.

Through methodical testing, including multiple user tests, we have evaluated the chatbot's functionality and user response. The results indicate an improvement in user engagement and a positive reception of the chatbot's features. Future work will focus on integrating more comprehensive databases and improving natural language understanding to further enhance the chatbot's efficiency and accuracy.

The project can contribute positively to Sunnmørsposten by providing them with an innovative solution to increase reader engagement, while also offering us valuable experience and learning in the fields of artificial intelligence and news dissemination.

Sammendrag

Dette prosjektet omhandler utviklingen av en kunstig intelligensdrevet samtalerobot (chatbot) for Sunnmørsposten, en nyhetsplattform som søker å forbedre leserengasjementet gjennom avanserte digitale løsninger. Med fokus på å tilby leserne en dypere forståelse og bedre navigasjon i nyhetsartikler, tar vår chatbot i bruk moderne teknologier som OpenAI's API for å generere dynamiske svar basert på brukernes forespørsler.

Målet med chatboten er å gi sammendrag, forklare komplekse saker, og foreslå relevante artikler for å forbedre brukeropplevelsen. Implementasjonen er utført til å passe inn smp.no, med funksjonaliteter som understøtter universell utforming for å sikre tilgjengelighet for alle brukere. Til tross for teknologiske og sikkerhetsmessige begrensninger har prosjektet demonstrert hvordan kunstig intelligens kan integreres effektivt for å møte spesifikke behov innen nyhetsformidling.

Gjennom metodisk testing, inkludert flere brukertester, har vi evaluert chatbotens funksjonalitet og brukerrespons. Resultatene indikerer en forbedring i brukerengasjement og en positiv mottakelse av chatbotens funksjonaliteter. Videre arbeid vil fokusere på integrering av mer omfattende databaser og forbedring av naturligspråkforståelsen for å ytterligere øke chatbotens effektivitet og nøyaktighet.

Prosjektet kan bidra positivt til Sunnmørsposten ved å gi dem en innovativ løsning for å øke leserengasjementet, samtidig som det har gitt oss verdifull erfaring og læring innen kunstig intelligens og nyhetsformidling.

1 Introduksjon

Dette kapittelet vil introdusere bakgrunnen og formulere oppgaven i dette prosjektet. Den skal gi et overblikk på bacheloroppgaven og vise til mål. Senere vil vi vise hvordan vi har strukturert rapporten.

1.1 Bakgrunn

Journalistene i Sunnmørsposten dekker ofte komplekse saker som er del av et større sammenhengende narrativ. For å gi leserne en grundig forståelse, er de avhengige av tilgang til informasjon fra tidligere artikler. Dessverre krever dette en tidsinvestering som mange lesere ikke har anledning til. Sunnmørspostens analyseverktøy indikerer at leserne i gjennomsnitt bruker kun ett minutt per artikkel. Derfor har vi valgt dette problemet som tema for vår bacheloroppgave, med mål om å utvikle en løsning som forenkler tilgangen til bakgrunnsinformasjon og styrker lesernes opplevelse.

1.2 Problemformulering

Å forstå konteksten i nyhetsartikler kan være utfordrende. Tidkrevende navigering mellom artikler og mangel på tilstrekkelig bakgrunnsinformasjon kan føre til at viktige detaljer misforstås eller overses av leserne. Vårt mål er å utvikle en chatbot som skal kunne svare på spørsmål relatert til artikkelen, tilby sammendrag og forklare bakgrunnen for ulike saker. I tillegg vil den kunne foreslå relevante artikler basert på brukerens interesser.

1.3 Målsetning

Målene for dette prosjektet er som følger:

- Utvikle en samtalerobot som kan forklare bakgrunnen for en sak basert på tidligere artikler.
- Roboten skal kunne foreslå relevante spørsmål for å engasjere leseren og tilby sammendrag og lenker for ytterligere informasjon.
- Implementere chatboten på smp.no eller som en nettleserutvidelse.
- Sikre at chatboten støtter universal design.
- Bevise konseptet til en KI-dreven chatbot på nyhetsnettside.

1.4 Dagens løsninger

Moderne nyhetsformidling møter utfordringer med å engasjere lesere i en tid hvor informasjonsflommen lett kan overvelde. Flere tiltak er tatt i bruk for å forbedre brukerinteraksjon og tilgjengelighet på digitale plattformer, som dynamiske brukergrensesnitt. Vår løsning integrerer en avansert chatbot som benytter de nyeste teknologiene innen kunstig intelligens og naturlig språkforståelse til å forbedre brukeropplevelsen.

1.5 Begrensninger

Vår gruppe står overfor begrensninger i tilgangen til systemer og data fra Sunnmørsposten. Av hensyn til konfidensialitet og sikkerhet er direkte tilgang til data begrenset. Dette hemmer vår evne til å teste og validere systemet under realistiske forhold, noe som kan påvirke vår forståelse av systemets ytelse i produksjonsmiljøer.

Prosjektet er spesifikt rettet mot utvikling og implementering av en chatbot, uten å inkludere dypere dataanalyse eller manipulering. Dette er en viktig distinksjon ettersom det påvirker prosjektets omfang i vesentlig grad. Løsningen krever kun håndtering av brukerinteraksjoner gjennom chatboten, noe som reduserer prosjektets kompleksitet, men tillater samtidig en mer fokusert tilnærming til å utvikle en sikker, stabil og brukervennlig chatbot.

På grunn av de nevnte begrensningene har gruppen konsentrert seg om chatbotens kjernefunksjoner. Vår begrensede evne til å teste fullskala integrasjoner under komplekse forhold kan imidlertid påvirke løsningens robusthet og tilpasningsevne i reelle miljøer. Dette understreker behovet for ytterligere testing og tilpasning.

1.6 Rapportens struktur

Rapporten er strukturert som følger:

- Kapittel 1 – Introduksjon: Presenterer rammen for rapporten og gir en oversikt over innholdet som følger.
- Kapittel 2 – Teoretisk grunnlag: Beskriver den teoretiske kunnskapen nødvendig for dette prosjektet.
- Kapittel 3 – Metode: Inneholder en beskrivelse av metodene og materialene som er nødvendige for prosjektet.
- Kapittel 4 – Resultater: Fremlegger resultatene til prosjektet.
- Kapittel 5 – Drøfting: Inneholder en diskusjon av resultatene, fremtidig arbeid, og hvordan vi kom frem til resultatet.
- Kapittel 6 – Konklusjoner: Dette kapittelet oppsummerer funnene fra prosjektet.

2 Teori

I dette kapittelet av rapporten vil vi gjøre rede for de teoretiske prinsippene som ligger til grunn for vårt prosjekt.

2.1 Teoretiske prinsipp

2.1.1 Proof of concept

Proof of Concept (PoC) brukes til å demonstrere den praktiske muligheten for en teori, metode eller ide. PoC er en innledende fase i utviklingen av en løsning, hvor målet er å fastslå om en ide som kan realiseres i praksis. Implementeringen av PoC er avgjørende for å minimasere risiko og maksimere sansynligheten for suksess i nye prosjekter. Meier mfl., 2022 er et godt eksempel på PoC innenfor teknologi.

2.1.2 Etikk og personvern

Med utviklingen av kunstig intelligens (KI) vokser også bekymringene for etikk og personvern. KI-systemer, som behandler og analyserer enorme mengder data, skaper spørsmål om personvern og datasikkerhet. KI kan utføre analyser som potensielt kan avsløre svært personlig informasjon uten brukerens samtykke. Videre kan KI generere partisk informasjon som kan spre feilaktig informasjon. Det er derfor kritisk viktig at utviklere og brukere av KI-teknologi følger etiske retningslinjer. Dette er for å sikre at teknologien anvendes på en måte som respekterer personvern og fremmer rettferdighet. Disse problemstillingene og deres konsekvenser er diskutert i Alan Westins bok Westin, 1967, som fortsatt er relevant for dagens teknologidebatter. (Jobin mfl., 2019)

2.1.3 Iterativt design

Iterativt design er en metodikk som er fleksibel og dynamisk prosess gjennom hele prosjektet. Dette er fortløpende planlegging, utførelse, evaluering og revisjon av designet. Dette tilpasser prosjektet etter hvert som nye data og innsikter oppnås. I Verschuren og Doorewaard, 2010 diskuteres det hvordan en iterativ tilnærming tillater utviklere å være responsive til faser og tilbakemeldinger gjennom prosjekt og forskning. Utviklere som anvender denne metoden, returnerer regelmessig til tidligere faser i prosessen for å justere mål, metoder og tilnærminger basert på hva de lærer underveis. Dette bidrar til å sikre at forskningsprosjektet forblir relevant og at resultatene er så nøyaktige som mulig. Feng mfl., 2022 Omhandler en iterativ design metode og fremhever hvordan iterativt design kan balansere brukertilfredshet og bærekraft.

2.1.4 Brukergrensesnitt og tilgjengelighet

Brukergrensesnitt (UI) design er studiet og praksisen av å lage grensesnitt i programvare eller datamaskiner som fokuserer på stil og brukervennlighet. Et effektivt UI-design er viktig for å sikre at sluttbrukerne kan interagere effektivt med teknologien. Dette er alt fra oppsett av skjermen og visuelle design til responsivitet og tilgjengelighet. Brophy og Craven, 2007 sier: “Tilgjengeligheten

til webbasert informasjon kan forbedres i to hovedveier: Gjennom bruk av tilgangsteknologi og gjennom å ta i bruk god praksis innen grensesnittdesign. Begge er like viktige".

2.1.5 Brukeropplevelse (UX)

Brukeropplevelse (UX) er en sentral komponent i design av digitale interaksjoner, Berni og Borgianni, 2021 definerer og hvordan brukeropplevelse-prinsipper er fundamentalt for å skape intuitive og effektive brukergrensesnitt. Don Norman fremhever i sin bok, "The Design of Everyday Things" Norman, 2013, viktigheten av å designe produkter som er forståelige og brukervennlig. Norman introduserer konsepter som 'affordance', som refererer til egenskaper ved et objekt som tillater brukere å forstå hvordan det skal brukes, og 'signifiers', som gir brukerne ledetråder om hvordan interaksjoner skal finne sted. Disse prinsippene er direkte anvendbare i design av chatboter, hvor det er essensielt å skape klare og logiske flyter som brukerne intuitivt kan navigere. Ved å implementere Normans prinsipper, kan utviklere sikre at program ikke bare forstår brukerens henvendelser, men også kommuniserer tilbake på en måte som er lett for brukeren å forstå og følge. Dette bidrar til å skape en positiv brukeropplevelse.

2.1.6 Interaksjonsdesign

Interaksjonsdesign spiller en avgjørende rolle i utviklingen av brukervennlige digitale produkter og tjenester (Rogers mfl., 2023). Det fokuserer på å skape grensesnitt som lager effektiv og behagelig interaksjon mellom brukeren og produktet. Dette designområdet handler om å utforme hvordan brukere samhandler med teknologiske systemer, med spesiell vekt på brukervennlighet og tilgjengelighet. Elementer som brukervennlige knapper er sentralt her. En av de ledende ressursene i dette feltet, "About Face: The Essentials of Interaction Design" (Cooper mfl., 2014), tilbyr dyptgående innsikt i prinsippene som styrer effektiv interaksjonsdesign. Forfatterne undersøker hvordan designbeslutninger kan påvirke brukeropplevelsen og tilbyr strategier for å utforme mer intuitive, responsive og tilpassede grensesnitt.

2.2 Teknologi

2.2.1 Kunstig intelligens (KI/AI)

Kunstig intelligens (KI) er et bredt felt innen informatikk som omhandler utviklingen av intelligente maskiner som er i stand til å utføre oppgaver som tradisjonelt krever menneskelig intelligens. Dette inkluderer blant annet evner som: Å lære, problemløse, språkforståelse og beslutningstaking. (Russell og Norvig, 2016).

2.2.2 Natural Language Processing (NLP)

"Natural Language Processing (NLP) er en samling av datateknikker for automatisk analyse og representasjon av menneskelige språk, motivert av teori." (Chowdhary, 2020). Natural Language Processing (NLP), er en gren av datavitenskapen som fokuserer på å utvikle dataprogrammer og algoritmer som kan forstå og behandle menneskelig språk på en intelligent måte. Disse teknikkene

brukes til å analysere og tolke tekst og tale, slik at datamaskiner kan utføre oppgaver som tekstforståelse, oversettelse, informasjonsekstraksjon og automatisert svar på spørsmål. Målet med NLP er å skape datamaskiner som kan kommunisere og samhandle med mennesker på en naturlig og effektiv måte.

2.2.3 Maskinlæring

Maskinlæring handler om å utvikle datamaskinmodeller og algoritmer som tillater maskiner å lære fra erfaring og forbedre seg over tid. Dette kan sammenlignes med hvordan dyr og mennesker lærer gjennom erfaring og observasjon. Maskinlæring handler om å la en datamaskin lære fra erfaringer eller data, slik at den kan bruke denne kunnskapen senere. "Maskinlæring har som mål å designe datamaskiner som kan tilpasse og lære fra erfaring uten inngrep fra programmerere." (Chowdhary, 2020). Målet med maskinlæring er å skape datamaskiner som kan tilpasse seg og lære av erfaring uten manuell programmering.

2.2.4 Dialogsystemer

Dialogsystemer er teknologier og metoder for å skape og håndtere samtaler mellom mennesker og maskiner. Disse systemene kan variere fra regelbaserte systemer, som responderer med forhåndsdefinerte svar på spesifikke kommandoer, til mer sofistikerte maskinlæringsdrevne systemer som tillater dynamisk og kontekstavhengig interaksjon. Dialogsystemene spiller en viktig rolle i utviklingen av interaktive KI-applikasjoner, inkludert virtuelle kundesupportagenter og interaktive underholdningssystemer.

2.2.5 Integrering av databaser og API-er

Integrering av databaser og API-er (Application Programming Interfaces) er en teknisk disiplin som involverer koblingen av forskjellige datasystemer og programvareapplikasjoner for å tillate dem å arbeide sammen. Dette feltet er avgjørende for å utvikle skalérbare, effektive og interaktive systemer. Integrering gjør det mulig for applikasjoner å hente, utveksle og manipulere data sammen og støtter alt fra webtjenester til komplekse forretningsprosesser og sky-tjenester. Forklarer bruken av OpenAI sine API-er. (Paredes mfl., 2023)

2.2.6 Nyhetsnettsted

Nyhetsnettsteder spiller en avgjørende rolle i hvordan publikum tilgang til og konsumerer nyheter i den digitale tidsalderen. Utviklingen av disse nettstedene krever integrasjon av flere teknologier og prinsipper for å sikre effektivitet, tilgjengelighet og brukervennlighet.

2.2.7 Øke leserengasjement

For å maksimere engasjement og forlenge den tiden leserne tilbringer på et nyhetsnettsted, er det avgjørende at redaktører og journalister anvender en rekke nøye utvalgte strategier. Produksjon av dyptgående og tankevekkende innhold står sentralt, sammen med integrasjonen av multimediale

elementer som videoer, interaktive grafikker og infografikker. (Lehmann mfl., 2016). Disse teknikkene gjør ikke bare historiene mer tiltalende, men også mer tilgjengelige og forståelige for et bredere publikum. Som Heather L. O'Brien påpeker, er det viktig å lage et design og innhold for nyhetssider med et helhetlig mål. Hun sier i sin forskning der hun har undersøkt nettsted-engasjement:

"Resultatet av denne utforskningen demonstrerer fordelene med å ta en mer helhetlig tilnærming til informasjonsinteraksjon i domener som online nyheter, hvor informasjonsatferd er utforskende og ikke alltid målrettet, og hvor grensesnittet er designet for å fremme nettlesing" (O'Brien, 2012).

Dette sitatet fra undersøkelsen understreker betydningen av å lage nyhetsgrensesnitt som oppmuntrer til utforsking gjennom interaktive elementer, i stedet for å diktere hvordan brukerne skal navigere på nettstedet. En slik tilnærming kan lede til dypere engasjement og lengre oppholdstid på siden, da brukerne ikke bare søkes etter spesifikk informasjon, men også lokkes inn i en omfattende utforskning av tilgjengelig innhold.

2.3 Prosjekt metodikk - SCRUM

SCRUM er et rammeverk for smidig programvareutvikling som er iterativt og trinnvis. Det ble først definert av Hirotaka Takeuchi og Ikujiro Nonaka i 1986 i deres artikkel "The New Product Development Game" (Sachdeva, 2016).

SCRUM baserer seg på prinsipper om fleksibilitet og nært samarbeid innen gruppen. Det vektlegger selvorganisering og ansvarliggjøring av gruppemedlemmer, hvor gruppen fungerer som en enhet mot et felles mål.

2.3.1 SCRUM oversikt

SCRUM fokuserer på å observere og reagere basert på det som faktisk skjer i prosjektet, ikke på forutinntatte forventninger. Dette gjøres gjennom gjennomsiktighet, inspeksjon og tilpasning (Sachdeva, 2016). I motsetning til tradisjonell utvikling som ofte er stiv, tilbyr SCRUM en mer fleksibel og adaptiv tilnærming som er ideell for prosjekter med høy grad av usikkerhet og endringer i kravene.

2.3.2 SCRUM-roller

Ifølge Sachdeva, 2016 er dette rollene i SCRUM:

- Produktowner: Denne rollen innebærer styring av produktbackloggen og sikrer at de mest verdifulle funksjonene utvikles først. Produkteieren må være én person og ha klart definert ansvar for å maksimere produktverdien.
- SCRUM Master: Fungerer som en coach for både utviklingsgruppen og produkteieren og sikrer at SCRUM-prosessen følges. SCRUM Masteren hjelper gruppen med å fjerne hindringer og beskytter dem mot ytre forstyrrelser.
- Utviklingsgruppe(Testere/utviklere): En tverrfaglig gruppe som selv styrer hvilke oppgaver de skal utføre og er ansvarlig for å levere høykvalitets produkter. Gruppen skal være liten (vanligvis 7 +/- 2 medlemmer) for å optimalisere effektivitet.

2.3.3 SCRUM tilpasning for små grupper

Tradisjonelt har SCRUM blitt brukt av grupper bestående av tre eller flere medlemmer, men det er viktig å merke seg at prinsippene i SCRUM kan tilpasses til mindre grupper, inkludert grupper med kun to medlemmer.

SCRUM for to er et konsept som tar sikte på å tilpasse SCRUM-prinsippene til et gruppe bestående av kun to personer. Ifølge Fauver, 2022 kan SCRUM for to være like effektivt som tradisjonell SCRUM fordi det gir struktur og fokus for gruppen, selv om det kun består av to medlemmer. Artikkelen i BreakFree Solutions, (Fauver, 2022), viser hvordan SCRUM-prinsippene kan tilpasses for å passe til en gruppe med kun to personer. Fauver understreker viktigheten av å tilpasse SCRUM-strukturen, opprettholde gjennomsiktighet, og utøve SCRUM-verdier som engasjement og fokus, selv i et lite gruppemiljø.

Det er viktig å tildele rollene som product owner og utvikler til hver av de to medlemmene. Dette oppsettet sikrer at én fokuserer mer på utviklingsoppgaver mens den andre tar seg av ledelsesaspekter. Dette arrangementet er effektivt fordi SCRUM verdsetter fleksibilitet innenfor gruppen.

Ved å implementere prinsippene i SCRUM for to, kan grupper med begrenset antall medlemmer oppnå økt produktivitet og måloppnåelse, samtidig som de opprettholder kvaliteten og strukturen som er kjennetegnet ved tradisjonell SCRUM-metodikk.

2.3.4 Sprint

En sprint er en definert tidsperiode, vanligvis to til tre uker der en gruppe arbeider med å levere et bestemt sett med funksjoner eller oppgaver. Sprints er kjernen i SCRUM-metodikken og inneholder flere viktige hendelser:

- Sprintplanlegging: Planlegger arbeidet som skal gjøres i løpet av sprinten. Dette møtet skal være til en bestemt tid, dette sikrer at gruppen forplikter seg til en realistisk mengde arbeid.
- Daglig SCRUM-møte: Et kort møte der hvert gruppemedlem rapporterer hva de gjorde sist og hva de planlegger å gjøre for dagen. Det kan ofte ble holdt flere slike møter om dagen.
- Sprintgjennomgang: Det er viktig å holde et reflektert møte der gruppen går gjennom hva de har oppnådd og reflekterer over hva som må gjøres fremover.

2.3.5 Sprint Backlog

Det er essensielt å vedlikeholde og revidere to nøkkelartefakter i SCRUM: produktbackloggen og sprintbackloggen. Produktbackloggen inneholder en oversikt over kravene til produktet, som gruppen kan referere til kontinuerlig. Sprintbackloggen består av en liste over oppgaver (issues) som gruppen har forpliktet seg til å gjennomføre i løpet av sprinten. Denne listen er dynamisk og justeres fortløpende etter behov.

2.4 Versjonskontroll

Versjonskontrollsystemer (VCS) spiller en avgjørende rolle i moderne programvareutvikling ved å muliggjøre effektivt samarbeid, sporbarhet av endringer og sikkerhetskopiering av kodebasen. (Spinellis, 2005). Git er et av de ledende distribuerte versjonskontrollsystemene, som tilbyr utviklere muligheten til å lagre hele historikken til et prosjekt lokalt på hver enkelt datamaskin.

2.4.1 Fordeler med VCS

Bruk av et versjonskontrollsystem (VCS) forhindrer at koden overskrives av andre og oppretter et historisk arkiv over endringer. Gjennom versjonshåndtering er det mulig å se forskjeller mellom ulike versjoner av samme fil, identifisere hvilke utviklere som endret spesifikke linjer, og fastslå tidspunktet for disse endringene. Videre tillater versjonskontroll at utviklingen deles inn i separate grener, som er ideelt når flere utviklere arbeider på samme prosjekt. Dette gjør det mulig for teamet å jobbe parallelt og effektivt ved å splitte og flette grener etter behov. Versjonskontrollsystemet kan også brukes til å spore feilrettinger. Det bidrar til en mer strømlinjeformet utviklingsprosess. (Spinellis, 2005).

3 Metode

I dette kapittelet av rapporten vil vi beskrive metodene og teknikkene som ble benyttet i prosjektet. Vi vil forklare prosessen for datainnsamling og -analyse, samt de verktøyene og teknologiene som ble brukt for å gjennomføre forskningen.

3.1 Organisasjon

Dette avsnittet omhandler de tre separate organisatoriske enhetene involvert i oppgaven og beskriver deres respektive roller.

3.1.1 Prosjektgruppe

Prosjektgruppen består av to studenter ved NTNU i Ålesund, Trond Lund Slinning og Eirik Løvik. Studentene opptrer som oppdragstaker og driver utviklingen av produktet.

3.1.2 Reorganisering av gruppesammensetning

Under initieringen av prosjektet ble gruppesammensetningen utvidet med et nytt medlem. Det ble etter hvert klart at det nye medlemmet ikke inneholdt nødvendig forhåndskunnskap, noe som førte til bekymringer omkring kommunikasjon og bidrag til prosjektet. Disse bekymringene ble adressert i samråd med vår veileder. Etter grundige vurderinger og i tråd med akademiske retningslinjer, ble det besluttet å reorganisere gruppesammensetningen for å opprettholde prosjektets kvalitet og framdrift. Denne beslutningen førte til en omfordeling av arbeidsoppgaver blant de gjenværende medlemmene.

3.1.3 Veileder

I løpet av bacheloroppgaven la vi vekt på å opprettholde god kommunikasjon med veilederen for å sikre en smidig prosess gjennom hele prosjektet. Dette ble oppnådd ved å holde regelmessige møter minst annenhver uke, og vi økte kommunikasjonen etter behov hvis gruppen trengte ytterligere veiledning. Veilederens rolle var å gi råd om gjennomføringen av oppgaver, bistå med viktige beslutninger og tilby generell støtte.

3.1.4 Oppdragsgiver

I vårt prosjekt er oppdragsgiveren Sunnmørsposten. Vi har samarbeidet med dem for å forstå deres spesifikke behov og krav til teknologien. Sunnmørsposten har gitt oss betydelig frihet til å utvikle et produkt som passer deres behov, og vi har arrangert møter med dem for å diskutere produktets funksjoner og motta tilbakemeldinger. Dette samarbeidet sikrer at vi kan tilpasse løsningen til de unike kravene til en nyhetsorganisasjon.

3.2 Systemrestriksjoner

En vesentlig utfordring i vårt prosjekt var begrensningene rundt tilgangen til Sunnmørspostens artikkeldatabase og systemer. Sunnmørsposten uttrykte bekymringer for hvordan databasen skulle brukes, spesielt med tanke på at GPT-modellen kunne bruke deres dokumenter til å trene seg opp. I tillegg inneholdt databasen artikler som ikke var tilgjengelige på nettsiden og som Sunnmørsposten ønsket å holde konfidensielle.

Vi diskuterte muligheten for å motta en CSV-fil eller lignende for å fylle inn i vår dummy-database. Dette vil tillate oss å tilpasse chatboten til Sunnmørspostens systemer og eksisterende oppsett. I oppgavebeskrivelsen ble det beskrevet at vi skulle ha tilgang til alle systemer. Imidlertid, da dette ikke ble realisert, var vi nødt til å søke alternative løsninger for utvikling og testing av chatboten.

Tidlig i prosjektet ble det antydet at vi kunne få tilgang til testmiljøet til Sunnmørsposten for utvikling og testing av chatboten. Dette viste seg imidlertid å være utfordrende, ettersom deres nettside er utviklet av Polaris Media. Dette skapte tekniske utfordringer for oppdragsgiver, som måtte koordinere med det eksterne firmaet før de kunne gi oss tilgang. Sunnmørsposten ga dermed uttrykk for at dette ville være vanskelig å realisere, og tilgangen ble aldri innvilget. Sunnmørsposten tilbød oss muligheten til å teste produktet vårt i deres testmiljø i deres lokaler. Derimot ville dette innebære betydelig mer arbeid. Å utvikle produktet separat fra deres systemer og deretter finjustere det hver gang vi skulle teste, var ikke praktisk gjennomførbart innenfor tidsrammen og ressursene vi hadde tilgjengelig. Integrasjonen av produktet i oppdragsgivers systemer vil bli diskutert videre i fremtidig arbeid (Fremtidig arbeid 5.10.5).

3.3 Prosjektplan

3.3.1 Forprosjektsplan

Forprosjektsplanen er et dokument som skisserer planleggingsfasen av et prosjekt, før selve gjennomføringen begynner. (Forprosjektsplan, B). Rapporten inneholder en innledende evaluering av prosjektets omfang, mål, nødvendige ressurser og potensielle utfordringer. Hensikten med rapporten er å gi gruppen et grunnlag for å vurdere prosjektetets, lage plan for gjennomføring og for å ta informerte beslutninger på forhånd.

3.3.2 Ganttdiagram

I planleggingsfasen av prosjektet utarbeidet gruppen et utkast av et Gantt-diagram. Dette diagrammet inkluderte tidspunktene for start og slutt for ulike utviklingsfaser, samt planlagte tidspunkter for brukertester i de angitte fasene. Ganttdiagrammet forprosjektsplanen: (Forprosjektsplan B).

3.3.3 Risikovurdering

I forprosjektsplanen ble det gjennomført en risikovurdering for å identifisere potensielle problemer som kunne oppstå underveis i prosjektet, samt fastsette hvilke tiltak som skulle implementeres dersom disse problemene skulle oppstå. Risikovurderingen er under Førprosjektrapporten. (Forprosjektsplan B).

3.3.4 Mål

Vi hadde et tidlig møte med Sunnmørsposten for å definere felles mål for prosjektet. Sunnmørsposten var åpne for at prosjektet enten kunne være et Proof of Concept eller et ferdig produkt klart for bruk. Basert på dette, besluttet vår gruppe å utvikle produktet med sikte på å ta det i bruk, samtidig som vi opprettholdt realistiske forventninger og inkluderte Proof of Concept-fasen i vår planlegging. Oppdragsgivers hovedmål var at chatboten skal kunne:

1. Generere sammendrag av artikler for å gi leseren en rask oversikt.
2. Forklare lange tekster og tilføre ytterligere kontekst.
3. Veilede leseren til andre relevante artikler gjennom lenker.
4. Foreslå artikler basert på leserens tekstspørsmål.
5. Engasjere leseren i samtaler om artiklene.

3.4 Programmering og språk

3.4.1 Node.js

Node.js er runtime-miljø som lar deg kjøre Javascript-kode untenfor en nettleser. Det er designet for å bygge skalerbare nettverksapplikasjoner. Med sin støtte for ulike pakker gjennom NPM, gir Node.js muligheten til raskt å integrere og utvide funksjonaliteten i applikasjonen vår.

3.4.2 Node Package Manager

NPM (Node Package Manager) er en av de mest benyttede pakkehåndtererne for JavaScript, og den er spesielt populær blant Node.js-utviklere. Vi bruker npm på grunn av dens tilgjengelige økosystem til tredjepartsbiblioteker. NPM inneholder et enormt bibliotek av pakker som kan gjenbrukes for å akselerere utviklingsprosesser. Den støtter semantisk versjonering og pakkelåsing. Dette sikrer at prosjekter forblir stabile og forutsigbare selv når nye pakker legges til. NPM er også integrert direkte med Node.js, som gjør det til et praktisk valg for oss.

3.4.3 HTML og CSS

Vi bruker HTML og CSS for å utforme brukergrensesnittet til vår chatbot, da disse teknologiene er standard for å skape strukturerte og stilige nettsider. HTML sørger for grunnleggende oppsett, mens CSS tillater detaljert stillegging og responsivt design. Denne tilnærmingen gjør det enkelt å integrere chatboten på tvers av forskjellige enheter, og sikrer at brukeropplevelsen er estetisk og funksjonell. Videre gir direkte utvikling i HTML og CSS oss fleksibilitet til å raskt justere design og layout etter tilbakemeldinger fra brukerne uten å bruke tredjepartsverktøy.

3.4.4 Javascript

I oppstartsfasen diskuterte vi valget av programmeringsspråk for prosjektet. Beslutningen om å anvende JavaScript for både front-end og back-end, ble basert på en vurdering av gruppens tidligere erfaringer og prosjektets realistiske behov. Under et innledende møte med oppdragsgiveren vurderte vi gruppens kompetanse og erfaringer med ulike programmeringsspråk, sammen med oppdragsgiverens allerede eksisterende bruk av programmeringsspråk. Siden oppdragsgiver benyttet JavaScript og gruppemedlemmene hadde erfaring med språket, besluttet vi at JavaScript var det mest realistiske og brukervennlige valget for hele prosjektet.

JavaScript er universelt støttet av alle moderne nettlesere. Dette gjør det til et fundamentalt valg for utvikling av webapplikasjoner. Det spiller en essensiell rolle i å skape dynamiske og interaktive brukeropplevelser på nettsider, og integreres sømløst med HTML og CSS. Språket kjører på klientens nettleser og muliggjør implementering av diverse funksjoner. Dette inkluderer dynamisk innlasting av innhold uten å oppdatere hele siden, samt skaping av engasjerende visuelle interaksjoner som reagerer på brukerens handlinger.

3.4.5 Express.js

Dette rammeverket er et av de mest populære innen Node.js-økosystemet for håndtering av HTTP-forespørsler. Vi valgte Express på grunn av dets fleksibilitet og støtte for en rekke nyttige middleware, som gjør det mulig å utvide og tilpasse vår chatbots funksjonaliteter effektivt.

3.4.6 Body-parser

Body-parser er et Express-middleware som behandler innkommende forespørsler ved å lagre dem som et JavaScript-objekt under 'req.body'-egenskapen. Dette gjør det enkelt å håndtere data i form av JSON (JavaScript Object Notation) som sendes fra klienten til serveren. En body-parser er derfor nødvendig for å kunne lese og forstå JSON-data som blir sendt fra klienten til serveren.

3.4.7 CORS (Cross-Origin Resource Sharing)

For å sikre at vår chatbot kan motta forespørsler fra forskjellige domener uten sikkerhetsrisikoer, implementerte vi CORS. Denne mekanismen er essensiell for å opprettholde strenge sikkerhetsstandarder samtidig som den tillater nødvendig ressursdeling over domener.

3.4.8 Process

Inneholder miljøvariabler for den gjeldende prosessen. Dette elementet er essensielt for å sikre at API-nøklene og andre konfigurasjonsdetaljer kan hentes trygt fra .env-filen.

3.4.9 Dotenv

Et hjelpebibliotek som laster inn miljøvariabler fra en .env-fil til 'process.env', som muliggjør enkel tilgang til konfigurasjonsdata gjennom hele applikasjonen.

3.4.10 Path

Et modul som benyttes for å håndtere og konstruere filstier på en måte som er kompatibel over forskjellige operativsystemer.

3.4.11 Rammeverk

Gruppen besluttet om å ikke benytte et rammeverk for utviklingen av vår chatbot. Dette valget var basert på flere vurderinger. Først og fremst var prosjektet vårt kun rettet mot å utvikle en chatbot, og vi hadde derfor ikke behov for de ytterligere funksjonene som et rammeverk tilbyr. Videre var tidsbesparelse en høy prioritet. Vi ønsket å unngå den tidskrevende læringsprosessen som ofte følger med å sette seg inn i et nytt rammeverk. Videre la vi vekt på fleksibilitet og enkelhet i kodebasen. Dette tillot oss å tilpasse løsningen mer direkte til våre spesifikke behov uten de restriksjonene og den ekstra kompleksiteten som rammeverk ofte fører til. Ved å velge en direkte implementering med JavaScript, sikret vi en effektiv utviklingsprosess.

3.4.12 Open-AI ChatGPT

Ved å benytte OpenAI's API bruker vi den nyeste teknologien innen naturlig språkforståelse. GPT-modellene er bygget på et stort transformbasert nettverk som er trent på en bred samling av tekster. Dette gjør det ideelt for å generere menneskelignende, informativ og relevant tekst basert på brukerinput. ChatGPT er et bra verktøy for en chatbot ettersom det kan hjelpe brukere gjennom samtaler og gi relevant informasjon basert på brukers meldinger. Med ChatGPT blir det betydelig enklere å håndtere utfordringer knyttet til avansert språkforståelse, som ofte forekommer i chatbot-systemer. For eksempel, utfordringer som å oppsummere artikler kan effektivt adresseres. Man kan enkelt programmere chatboten til å følge spesifikke instruksjoner og svare basert på gitte ledetråder. Dersom chatboten støter på spørsmål den ikke kan svare på, er den designet til å omdirigere brukeren til alternativ informasjon på egen hånd.

3.4.13 Postgres og Supabase

Gruppen bestemte seg sammen å bruke Postgres. Dette bestemte vi oss hovedsaklig for fordi det er en åpen-kildekode, (open source), relasjonsdatabase som er fleksibel å bruke. Supabase er et sett med verktøy og tjenester bygget rundt Postgres. Vi valgte dette verktøyet fordi det er ekstremt brukervennlig og sparer oss mye tid og krefter. Det tilbyr også ferdige løsninger for å håndtere vanlige utfordringer som kan oppstå.

3.4.14 Lighthouse

Gruppen benytter 'Lighthouse' for å sikre at prosjektet møter høyeste standarder. Dette verktøyet er essensielt for å vurdere kvaliteten på webapplikasjoner, med spesiell vekt på ytelse, tilgjengelighet og beste praksiser.

- Måle ytelse: Analyserer lastetider og interaktivitet for å forbedre brukeropplevelsen.

- Vurdere tilgjengelighet: Sikrer at nettstedet er tilgjengelig for alle brukere.
- Sjekke beste praksis: Verifiserer at nettstedet følger moderne webutviklingsstandarder.

3.4.15 WAVE

WAVE, som står for Web Accessibility Evaluation Tool, er et program for å hjelpe utviklere og designere med å gjøre deres nettsider mer tilgjengelige for personer med ulike former for funksjonshemninger. Det gir en visuell representasjon av hvordan en nettside presterer i henhold til webtilgjengelighetsstandarder, fremhever elementer som er godt implementert og peker ut feil.

Gruppen brukte WAVE fordi det er viktig at teknologiske løsninger, som en chatbot, er tilgjengelige for alle brukere, inkludert de med nedsatt syn, hørsel, motoriske ferdigheter, og andre behov.

3.5 Prosjektmetodikk - SCRUM

Gjennom prosjektet jobbet gruppen i sprints der det ble satt opp mål for én- til to-ukers perioder gjennom utviklingen av prosjektet. Dette gjorde vi for å oppnå en god måte å dokumentere arbeidet vi jobbet med og for å holde god oversikt på hva som måtte prioriteres.

3.5.1 Stand-ups

Vi holdt våre faste stand-up-møter tidlig på dagen på den avtalte møtedagen. I disse møtene gjennomgikk vi detaljert hva vi skulle fokusere på i tiden fremover. På vanlige arbeidsdager arrangerte vi kortere stand-ups der vi delte våre daglige planer, diskuterte pågående utfordringer, og identifiserte potensielle fremtidige problemer.

3.5.2 Gruppeoppsett

Som følge av en tidligere nevnt reorganisering ble vår gruppe redusert med én person, noe som førte til en endring i gruppedynamikken. Til tross for dette ønsket vi å fortsette med SCRUM-metodikken så langt det var mulig. Dette medførte at vi måtte tilpasse arbeidsoppgavene til en mindre gruppe. Vi utpekte en hovedutvikler og en produktansvarlig. Selv om begge rollene er fleksible og omfatter varierte oppgaver, vil produktansvarlig ha et større ansvar for det administrative og kvalitetssikring, mens hovedutvikleren primært vil fokusere på fremgang i utviklingsarbeidet. Valget av roller føltes naturlig, og med SCRUM-metodikkens innebygde fleksibilitet var det også naturlig å bytte oppgaver underveis i prosjektet etter behov. Imidlertid var det én oppgave som ikke ble omorganisert: å skrive møtereferater. Dette resulterte i at møtereferater i stor grad ble forsømt i ettertid.

3.5.3 Backlog

Alle funksjoner som skulle inkluderes i produktet ble oppført som issues i en backlog. Deretter ble det opprettet sub-issues for å legge til rette for en smidigere, trinnvis utvikling. Produktbacklogen, basert på oppgavebeskrivelsen, var et kontinuerlig referansepunkt for gruppen.

3.5.4 Sprints

Sprintene ble organisert i Jira, Confluence og Google Docs. Sprintmetodikken var relativt ny for alle i gruppen, noe som gjorde det utfordrende å oppdatere Jira, men dette ble stort sett gjort fortløpende mens gruppen jobbet. Hver sprint varte i to til tre uker. Gjennom ukentlige møter holdt gruppen hverandre godt oppdatert på sprintene og arbeidsoppgavene. Ved slutten av hver sprint forsøkte vi å arrangere et kort møte med veileder for å presentere hva vi hadde oppnådd og diskutere eventuelle spørsmål som oppsto. Dette ga ofte gruppen verdifull innsikt i hvordan vi burde fortsette.

3.6 Versjonskontroll

I løpet av prosjektet har vi implementert praksiser for versjonskontroll for å håndtere og integrere kodeendringer effektivt. Ved å utnytte grenbasert utvikling har vi sikret at alle nye funksjoner og feilrettinger ble utviklet i isolerte grener, adskilt fra hovedgrenen (main). Dette tillot gruppen å utføre utviklingsarbeid parallelt uten å risikere integriteten til den produksjonsklare koden.

For å opprettholde kodekvaliteten, ble det ikke tillatt å pushe uferdig kode direkte til hovedgrenen. I stedet ble kodeendringer først slått sammen i en dedikert utviklingsgren hvor tester og kvalitetssikring ble gjennomført. Etter en vellykket valideringsprosess ble den endelige koden integrert i hovedgrenen gjennom en 'merge'-prosess. Denne metoden med 'pull'-forespørsler og kodegjennomganger før sammenslåing sikret at alle endringer ble vurdert og testet, noe som reduserte feilraten og forbedret systemets pålitelighet og stabilitet. Bruken av denne strategiske versjonskontrollen bidro ikke bare til en mer effektiv kodehåndtering, men også til en kultur av kvalitetsbevissthet og samarbeid innad i gruppen.

3.7 Kommunikasjon og styringsverktøy

3.7.1 Jira

På anbefaling fra vår veileder begynte vi å bruke programmet Jira. For å holde oversikten over arbeidsflyten, opprettet vi en backlog og organiserte sprints i Jira. Programmet er tett integrert med Confluence, noe som ytterligere forbedret vår prosjektstyring.

3.7.2 Confluence

Confluence er et kraftig verktøy for å håndtere dokumenter, notater og diagrammer. Vi begynte å bruke Confluence i prosjektets oppstartsfase for strukturert dokumentasjon og prosjektstyring. Men siden vi bare var to i gruppen, fant vi det mer praktisk å samarbeide direkte gjennom Google Docs for rask og enkel notatføring om løpende oppgaver.

3.7.3 Google Docs

Google Docs er en enkel og tilgjengelig plattform for lagring og deling av dokumenter på nettet. På grunn av vår kjennskap til den enkle bruken og umiddelbare tilgjengeligheten av dokumenter i Google Docs, ble det vårt foretrukne verktøy for samarbeid og deling av informasjon.

3.7.4 Figma

Figma er et kraftig og brukervennlig verktøy for å utvikle grensesnittdesign. Vi brukte Figma til å utvikle de første utkastene av wireframes for produktet.

3.7.5 Discord

Discord er en gratis kommunikasjonsapp designet for interaksjon gjennom tekst- og talekanaler. Appen er også godt egnet for å dele kodeutdrag. Gruppen vår brukte Discord for å koordinere hjemmekontor og samarbeide når det var nødvendig.

3.7.6 Overleaf

Overleaf er en skybasert LaTeX-editor som er optimal for samarbeid, hvor flere brukere kan arbeide på dokumentet samtidig. Dette verktøyet gir brukerne mulighet til å forme dokumentene med stor nøyaktighet. Derfor valgte gruppen å bruke Overleaf for å skrive vår bachelorrapport.

3.8 KI - Hjelpepartner

3.8.1 ChatGPT

I vårt bachelorprosjekt benyttet vi kunstig intelligens (KI) som en sentral hjelpepartner for å effektivisere utviklingen av vår software. Vi benyttet spesifikt ChatGPT til å foreslå fremgangsmåter, strategier og kodeutvikling. ChatGPT ble også anvendt for å generere kodeforslag som adresserte spesifikke programmeringsutfordringer vi støtte på. Dette omfattet assistanse til feilsøking og debugging, hvor ChatGPT foreslo mulige løsninger på problemer som oppsto. Selv om vi utviklet den endelige koden selv, fungerte ChatGPT effektivt som en verdifull assistent.

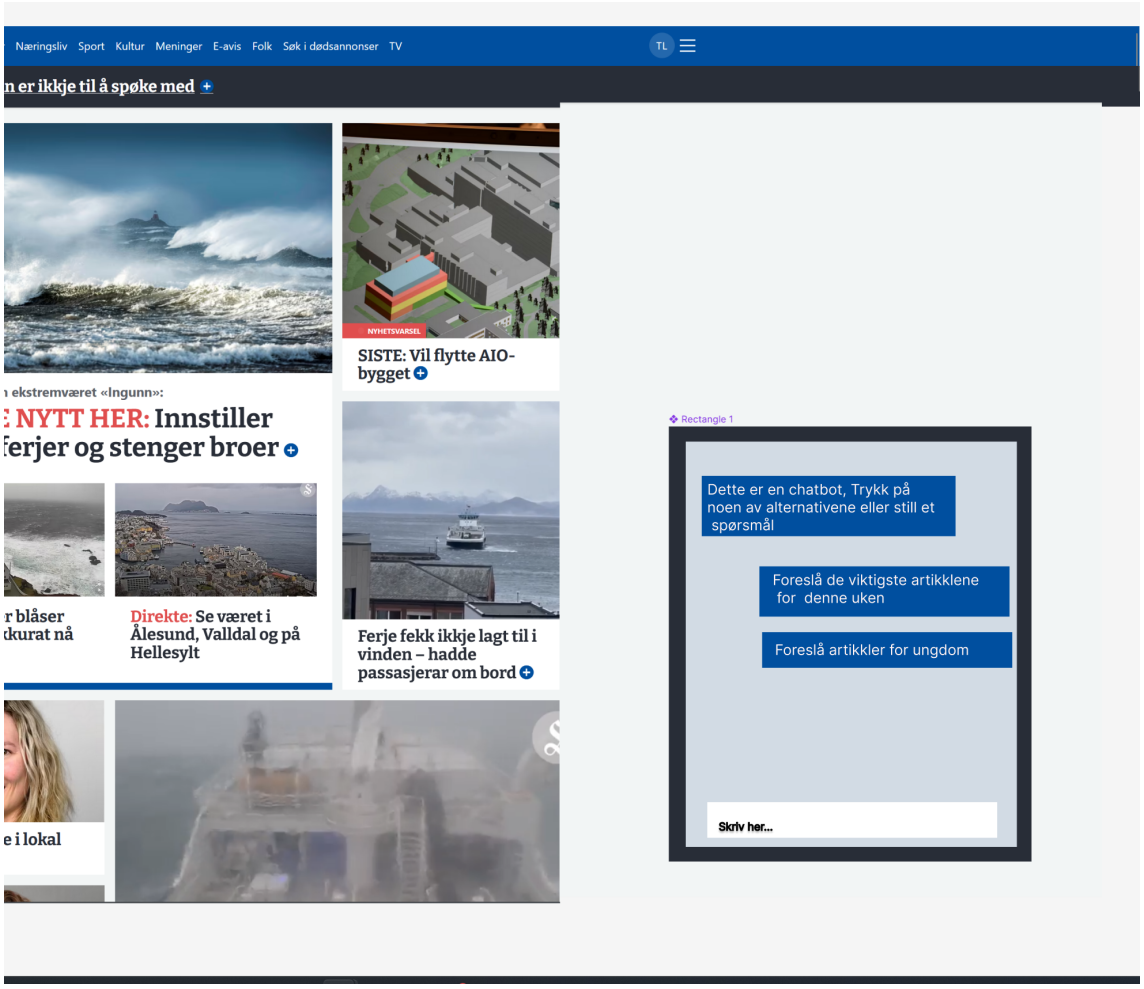
3.8.2 Github-Copilot

Videre benyttet vi GitHub Copilot som en AI-drevet kodeassistent for å forbedre vår hastighet og effektivitet ved å foreslå hele kodeblokker basert på konteksten i arbeidet vårt. Denne teknologien bidro til å automatisere deler av kodeutviklingen, slik at gruppen kunne konsentrere seg mer om komplekse problemstillinger og design.

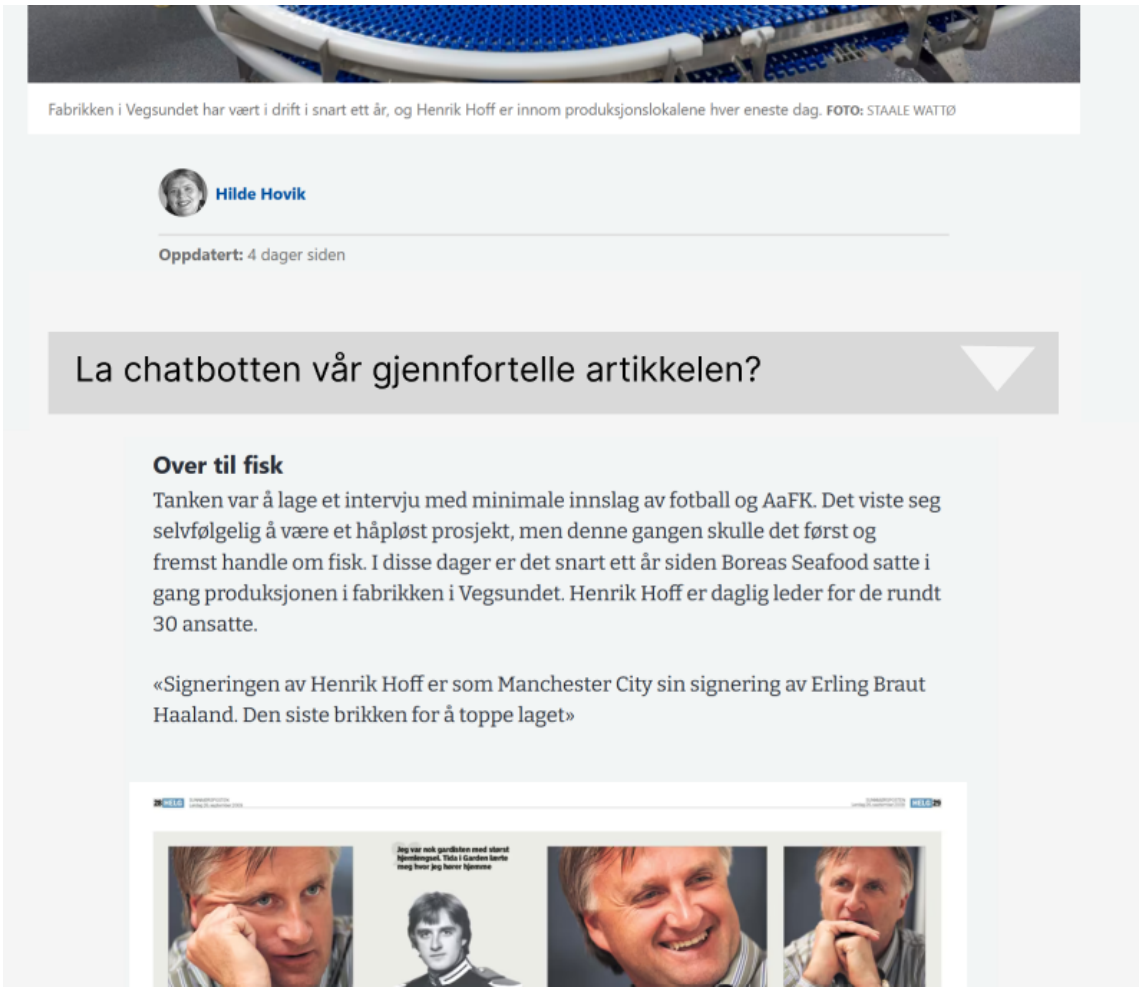
3.9 Produkt design

Vi benyttet Figma til å utvikle wireframes, som er grunnleggende skisser som illustrerer både utseendet og funksjonaliteten til chatboten vår. Ved å implementere wireframes tidlig i prosjektet, kunne vi nøye utforme både designet og funksjonaliteten. Dette ga oss innsikt i hvordan vi skulle lage chatboten og tilpasse den til brukernes behov. Wireframes ga oss oversikt over designet og muliggjorde tester for å samle tilbakemeldinger, noe som sikret kontinuerlige forbedringer gjennom prosjektet.

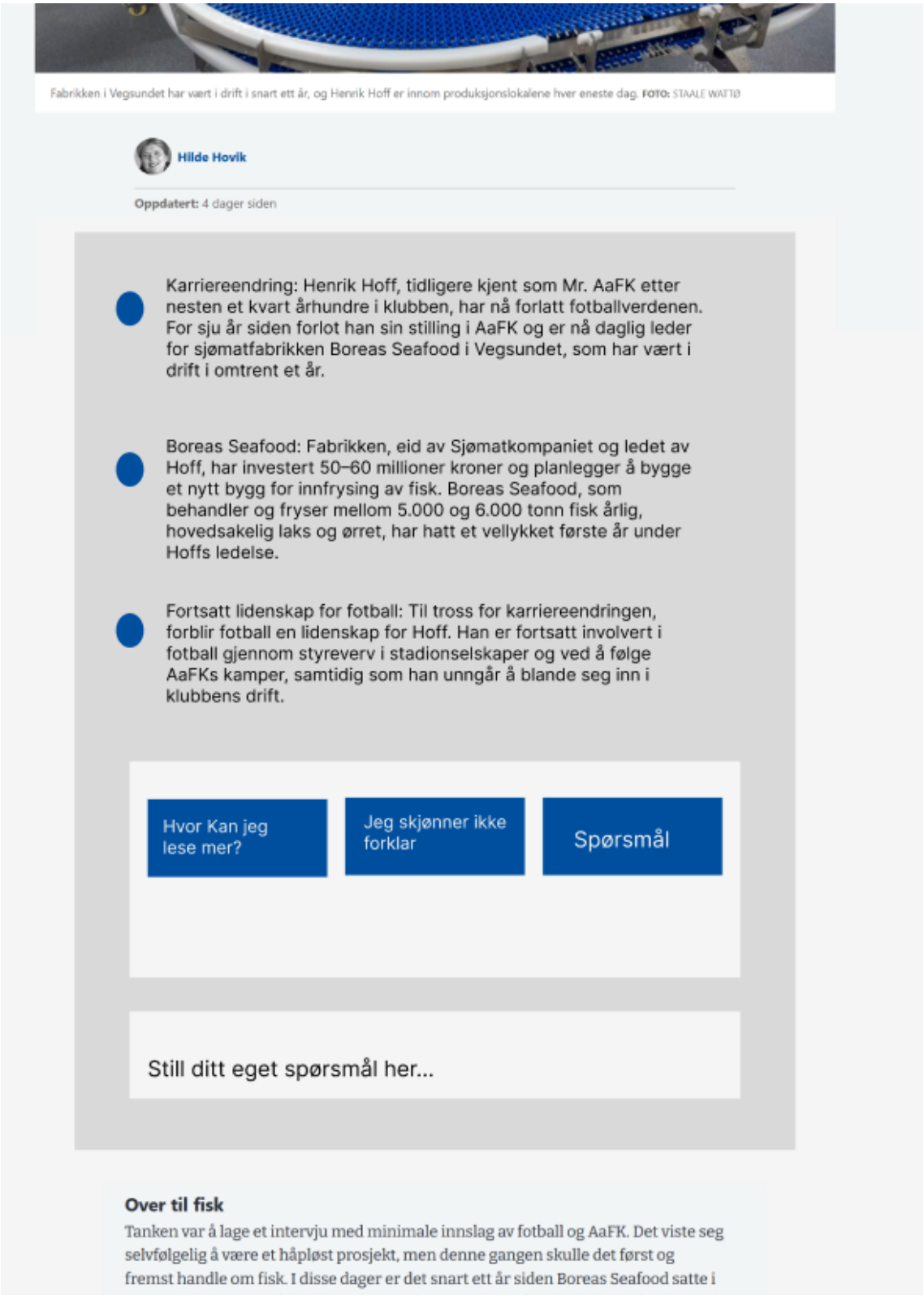
Disse skissene hjalp oss å visualisere brukergrensesnittet og interaksjonsflyten, og sørget for at designet var både intuitivt og effektivt. Tilbakemeldingene vi mottok underveis bidro til å skape en brukervennlig opplevelse, skreddersydd etter brukernes behov.



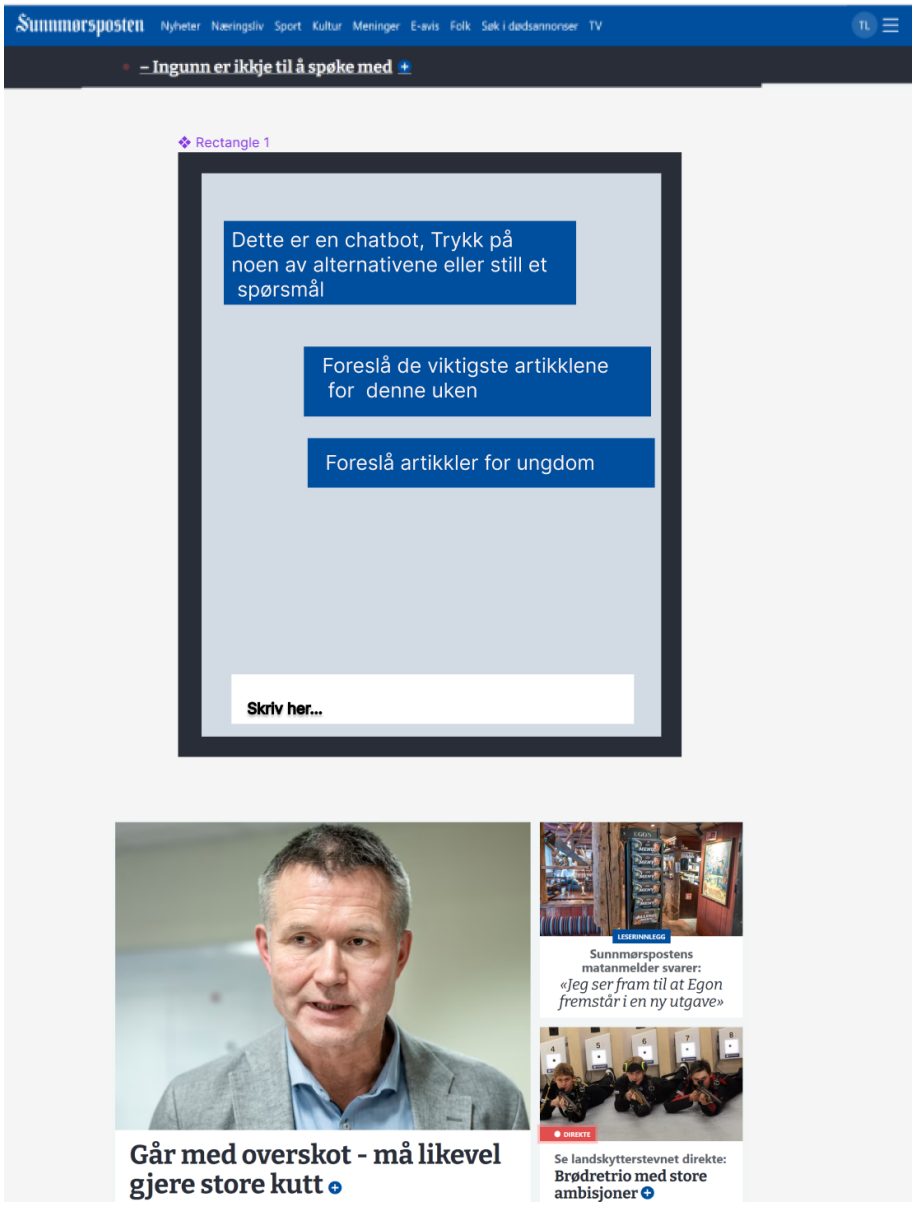
Figur 1: Konsept 1



Figur 2: Konsept 2:1



Figur 3: Konsept 2:2



Figur 4: Konsept 3

3.9.1 Beslutning og vurdering av wireframes

Under utviklingen av vår chatbot utarbeidet vi tre wireframes for å utforske ulike design- og funksjonalitetskonsepter. Hver wireframe representerte en unik tilnærming til chatbotens interaksjon med brukerne. Målet var å sette i gang diskusjoner under en planlagt brukertest for å innhente et bredt spekter av tilbakemeldinger.

1. **Konsept én** Tilbød et tradisjonelt chatvindu som var konstant tilgjengelig på nettsiden og veiledet brukeren gjennom ulike seksjoner.
2. **Konsept to** Var plassert under artikler, designet for å engasjere brukeren i samtaler knyttet spesifikt til artikkelens innhold og tilby funksjoner som sammendrag.
3. **Konsept tre** Fungerte som en egen side hvor brukere kunne utforske brede temaer relatert til hele avisens innhold, og få anbefalinger til artikler og samtaleemner.

Selv om vi opprinnelig antok at et tradisjonelt chatbot-design, som illustrert i konsept én, ville være det mest realistiske, ønsket vi å utforske alle konseptene gjennom brukertester for å trekke inn ideer og funksjonalitet fra hvert av dem. Brukertestene avslørte sterk interesse for konsept én, men brukerne viste også bemerkelsesverdig høy verdsetting av konsept to.

En bruker foreslo at en hybridløsning av konsept én og to ville være optimal. Dette bekreftet våre antakelser og førte til utviklingen av en integrert løsning som kombinerte de mest verdsatte elementene fra begge konseptene. Selv om konsept tre vekket en viss interesse, oppfylte det verken oppdragsgiverens opprinnelige krav eller behov. Derfor besluttet vi å fortsette utviklingen med en tradisjonell chatbot beriket med funksjonaliteten fra konsept to.

3.9.2 Simulering av nettside

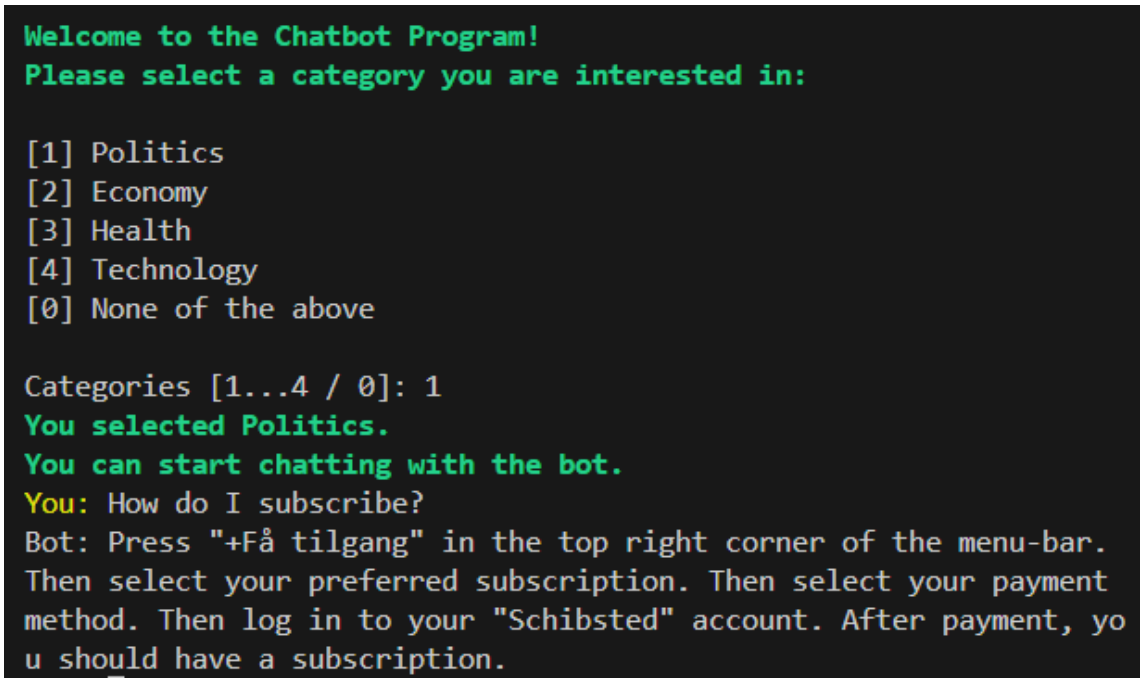
I utviklingsprosessen for vår chatbot stod vi overfor en utfordring da Sunnmørsposten ikke kunne tilby tilgang til deres eksisterende teknologiplattform. Dette tvang oss til å ta en alternativ retning. For å simulere en realistisk interaksjonsopplevelse med en nyhetsside, utviklet vi en enkel nettside som fungerte som et testmiljø for chatboten vår. Denne nettsiden bestod av to hoveddeler: en forside og en artikkelside. Forsiden var designet minimalistisk, med kun en header som presentert på bildet. Artikkelsiden bygde videre på designet fra forsiden ved å inkludere både headeren og en nyhetsartikkel, slik det også vises på bildet. Denne tilnærmingen tillot oss å teste og optimalisere chatbotens funksjonalitet og brukergrensesnitt under forhold som etterlignet et ekte nyhetsnettsted.

Basert på samtaler med oppdragsgiver kunne vi anta hvordan kolonnene i deres database så ut. Dermed kunne vi lage en enkel database som var ment å hjelpe oss i utviklingsprosessen. Vi hadde lenge planlagt å bytte til oppdragsgiverens database, men dette ble aldri gjennomført på grunn av Sunnmørspostens strenge retningslinjer for deling av deres data, som tidligere nevnt i 'System restriksjoner 3.1.4.'

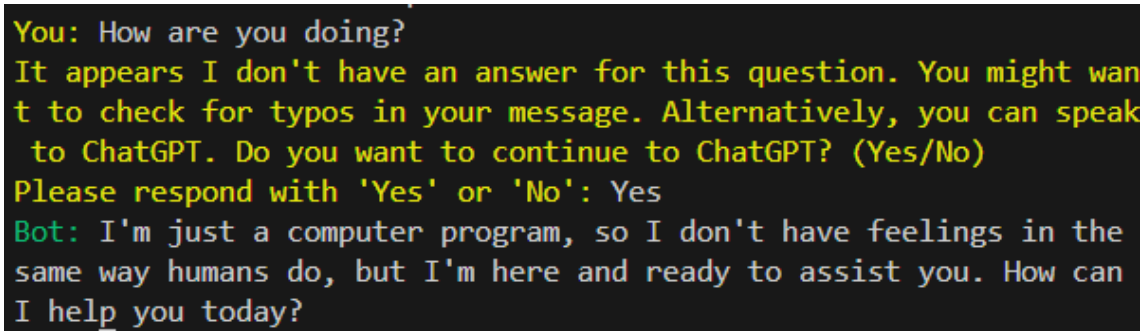
Figur 5: Enkel nettside

3.9.3 Chatbotens Evolusjon

I begynnelsen ble vår chatbot utelukkende kjørt i en terminal. Vi brukte enkle Regex-funksjoner for å identifisere nøkkelord i brukerens forespørsler og returnere forhåndsprogrammerte svar. Dersom chatboten ikke kunne gi et umiddelbart svar, omdirigerte den brukeren til ChatGPT for videre assistanse. Eksempler på denne tidlige interaksjonen er vist nedenfor i 'Figur 6' og 'Figur 7.'

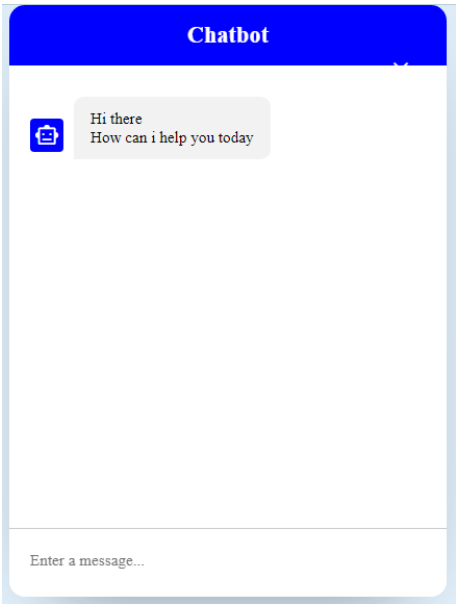


Figur 6: Terminal Chatbot



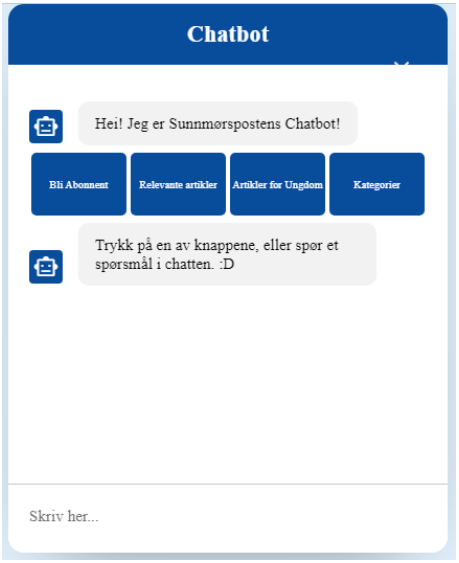
Figur 7: Terminal Chatbot (ChatGPT)

Videre utvikling førte til opprettelsen av en grafisk brukergrensesnitt (GUI) for chatboten. Dette nye designet inkluderte en velkomstmelding som markerte starten på samtalen, noe som gjorde chatboten mer brukervennlig, selv om den fortsatt fremsto som noe spartansk. Et eksempel på dette designet er vist nedenfor i 'Figur 8.'



Figur 8: GUI første utkast

Etter hvert ble funksjonalitet for interaktive knapper lagt til, designet i samsvar med tidligere wireframes. Disse ble opprinnelig plassert horisontalt, men på grunn av utfordringer knyttet til lesbarhet og estetikk når flere knapper ble lagt til, valgte vi senere å arrangere dem vertikalt. Denne endringen samsvarer med verdiene beskrevet i 'Brukergrensesnitt 2.1.3' og 'Brukeropplevelse 2.1.4'. Detaljer om denne iterasjonen er illustrert i Figur 9, som representerer det siste utkastet før chatboten nådde sitt nåværende utseende.



Figur 9: Siste iterasjon før nåverende design

3.10 Brukertester

I løpet av prosjektet gjennomførte vi tre brukertester for å validere designvalg og funksjonalitet for vår chatbot. Disse testene var avgjørende for å sikre at produktet møtte de faktiske behovene og forventningene til oppdragsgiver og potensielle brukere. Etter hver brukertest ble resultatene presentert for oppdragsgiver.

Brukertestene ble gjennomført med unge studenter i midten av tjueårene. Den tredje og siste brukertesten var en konkluderende evaluering for å fastslå fremtidige arbeidsområder og funksjoner som trengte optimalisering før lansering. Dette bidro til å identifisere spesifikke områder hvor chatboten kunne forbedres for bedre å møte brukeres forventninger og forbedre deres interaksjon med systemet frem mot lansering.

3.10.1 Brukertest 1

Tre studenter i begynnelsen av tjueårene deltok i Brukertest 1 for å evaluere tre forskjellige konseptutkast for den planlagte chatboten, som beskrevet i avsnittet 'Produkt design 3.9.' Gjennom tilbakemeldingene fra deltakerne fikk vi verdifull innsikt som vil forbedre chatbotens utvikling, med særlig fokus på brukernes behov og preferanser.

Testen ble gjennomført som en åpen diskusjon hvor studentene ble presentert for de tre konseptene og oppfordret til å dele sine meninger gjennom spørsmål. Deltakerne bestod av to byggingeniørstudenter og én dataingeniørstudent. Samtlige studenter var regelmessige lesere av nettaviser som VG og Dagbladet, og alle var kjent med chatboter. De hadde imidlertid blandede følelser om en chatbots nytteverdi på en nyhetsside.

konsept 1: Tradisjonelt chatvindu Dette konseptet tilbød et tradisjonelt chatvindu som var konstant tilgjengelig på nettsiden. Chatboten skulle veilede brukeren gjennom ulike seksjoner av nettsiden og være tilgjengelig for å svare på spørsmål når som helst.

Tilbakemeldinger: Brukerne var positive til dette konseptet, da det ga dem konstant tilgang til hjelp og veiledning. De mente at et tradisjonelt chatvindu kunne være nyttig for rask tilgang til informasjon og navigasjonshjelp på nettsiden.

konsept 2: Chatbot under artikler Dette konseptet plasserte chatboten under artikler, designet for å engasjere brukeren i samtaler knyttet spesifikt til artikkelens innhold. Funksjoner som artikkelsammendrag og forklaring av komplekse saker ble inkludert for å forbedre brukerens forståelse av innholdet.

Tilbakemeldinger: Brukerne satte stor pris på dette konseptet, da det gjorde det lettere å få bakgrunnsinformasjon og oppsummeringer direkte relatert til artikkelen de leste. De mente at dette kunne forbedre leseopplevelsen deres betydelig og gjøre det lettere å forstå artikler.

konsept 3: Utforskning av brede temaer Dette konseptet fungerte som en egen side på Sunnmørsposten hvor brukere kunne utforske brede temaer relatert til hele avisens innhold gjennom komplekse samtaler. Chatboten ville gi anbefalinger til artikler og samtaleemner basert på brukerens interesser og lesevaner.

Tilbakemeldinger: Selv om brukerne viste en viss interesse for dette konseptet, ble det ansett som mindre relevant for deres umiddelbare behov sammenlignet med de to andre konseptene. Brukerne mente at det kunne være nyttig for å utforske og diskutere artikler grundig, men ikke like praktisk for daglig bruk som de andre konseptene.

Samlet tilbakemeldinger Brukertesten viste stor interesse for både konsept én og konsept to. Brukerne mente at en tradisjonell chatbot som alltid er tilgjengelig ville være svært nyttig, samtidig som de verdsatte funksjonaliteten til en chatbot som kunne tilby oppsummeringer og kontekst under artikler. En av deltakerne foreslo en hybridløsning som kombinerte elementer fra begge konseptene. Dette inspirerte oss til å utvikle en integrert løsning som inkluderer de mest verdsatte funksjonene fra både konsept én og konsept to. Selv om konsept tre hadde noen interessante aspekter, ble det ansett som mindre relevant for våre forventninger og oppdragsgivers spesifikke behov og krav.

Brukertest 1 ga oss verdifulle innsikter som bidro til å forme retningen for videre utvikling av chatboten. Vi besluttet å fortsette med en tradisjonell chatbot beriket med funksjonaliteten fra konsept 2, for å sikre en balansert og brukervennlig løsning som møter både brukernes og oppdragsgivers behov og forventninger.

3.10.2 Brukertest: 2

Brukertest 2 ble gjennomført med samme gruppe studenter som i brukertest 1 for å bygge videre på deres tidligere innsikt og forventninger. Formålet var å få ny forståelse av hvordan brukerne ville samhandle med chatboten etter å ha blitt kjent med de grunnleggende konseptene.

Testen fulgte en lignende struktur som den første, men med større frihet for deltakerne til å utforske chatboten på eget initiativ. Deltakerne fikk et sett med oppgaver og ble oppfordret til å stille spørsmål og teste funksjonene fritt. Spørsmålene gjenspeilte hovedfunksjonene vi hadde utviklet til da.

Oppgaver:

1. Spør chatboten om hva som helst.
2. Finn en artikkel med chatboten.
3. Naviger til en artikkel, og be om en oppsummering av artikkelen.

Observasjoner:

1. Brukerne brukte chatfeltet mer enn forventet, noe som indikerte behovet for en robust chatfunksjonalitet.
2. Flere feil ble oppdaget, inkludert problemer med språkbytte midt i samtalen og feil i artikkelvisning.
3. Brukerne ønsket muligheten til å be om oppsummeringer uten å måtte være inne på en artikkel.

Testen avdekket behovet for forbedringer i databasefunksjonalitet, håndtering av tilfeldige spørsmål, samtalehistorikk og feilretting. Det ble klart at videre utvikling bør fokusere på å gjøre chatboten mer fleksibel. Etter brukertesten fokuserte utvikleren på dette, samt å rette opp i fundamentale feil.

3.10.3 Brukertest: 3

Brukertest 3 ble gjennomført med én til to brukere om gangen, totalt fem tester. Målet var å evaluere chatbotens hovedfunksjoner og brukeropplevelse, med fokus på å identifisere eventuelle hull i funksjonaliteten og sikre at brukerne kunne navigere og løse oppgaver effektivt.

Deltakerne fikk seks oppgaver de skulle gjennomføre, mens brukertestansvarlig noterte merkbare hendelser og stilte ledende spørsmål etter hver oppgave. Dette ga verdifull innsikt i brukernes navigasjonsmønstre og funksjonalitetsbehov.

Oppgaver:

1. Spør chatbotten et hvilket som helst spørsmål.
2. Finn en ny artikkel om kultur med hjelp av chatboten.
3. Finn en artikkel om sport med hjelp av chatfeltet.
4. Finn ut forventet leveringstid på papirversjonen av avisen med hjelp av chatboten.
5. Naviger til en artikkel og prøv chatbotens funksjoner.
6. Still spørsmål om artikkelen du er inne på.

Observasjoner:

1. Brukerne foretrakk oftere chatfeltet over knappene for å finne informasjon, noe som viste behovet for en mer robust chatfunksjonalitet.
2. Flere små feil i layouten ble oppdaget, og det var behov for forbedring av nøkkelordhåndtering i chatten.
3. Brukerne ønsket muligheten til å stille spørsmål om foreslåtte artikler uten å måtte åpne dem.

Testen avdekket flere behov for forbedringer:

- Forbedret chatfunksjonalitet
- Inkludering av kundeservicespørsmål i chatfeltet
- Retting av layoutfeil
- Løsning av problemer med nøkkelordgjenkjenning

Videre testing med oppdragsgivers systemer er nødvendig for å validere disse forbedringene og sikre at chatboten effektivt oppfyller brukernes behov.

3.11 Faser

Prosjektet ble gjennomført i tre hovedfaser: Planlegging, prototypeutvikling, og sluttføring av produktet. I planleggingsfasen la vi grunnlaget for chatboten ved å identifisere nødvendige funksjoner gjennom initielle brukertester med wireframes. I fasen for prototypeutvikling brukte vi innhentede tilbakemeldinger til å skape en funksjonell prototype. Den siste fasen fokuserte på justering av prototypen basert på ytterligere brukerfeedback og krav fra oppdragsgiver. Fasene ble konkludert med en brukertest og feedback fra oppdragsgiver.

3.11.1 Iterativt design

På anbefaling fra vår veileder, benyttet vi et Gantt-diagram for å planlegge arbeidsflyten. Vi besluttet å adoptere en iterativ designmetode 2.1.3, hvor vi startet med en innledende skisse, innhentet tilbakemeldinger, utviklet en prototype, og samlet enda flere tilbakemeldinger. Denne prosessen fortsatte til vi hadde en funksjonell versjon klar. Tilbakemeldinger fra brukere og oppdragsgiver var verdsatt og høyt prioritert.

3.11.2 Planlegging

Planleggingsfasen omfattet definering av prosjektets utfordringer og fastsettelse av nødvendige funksjoner for chatboten. Vi dedikerte de første ukene til møter for å utarbeide nødvendig dokumentasjon, inkludert Gantt-diagrammer og en førprosjektsrapport. Teamet brukte denne tiden til å forske på OpenAI's API, programmeringsspråk, og generelle chatbot-teknologier. En innledende brukertest med wireframes tillot oss å vurdere forskjellige design og funksjoner, og skille mellom essensielle og ønskelige funksjoner basert på tilbakemeldinger fra brukere og oppdragsgiver.

3.11.3 Utvikling oppstart

Under oppstartsfasen av utviklingen fokuserte vi på å utforske mulighetene OpenAI's API tilbød. Ett av våre teammedlemmer tok på seg ansvaret med å forstå funksjonaliteten til API-et ved å utføre tester i terminalvinduet. Vår opprinnelige plan var å utvikle en spesialisert modell basert på data fra Sunnmørsposten for å skape en chatbot som kunne ta automatiserte beslutninger støttet av ChatGPT-modellen. Dette er utdypet i avsnittet 'Opprinnelig plan 5.2.3.'

Parallelt med dette arbeidet et annet medlem med å utvikle chatbotens frontend. Dette inkluderte et elementært design av brukergrensesnittet og forberedelser for å integrere det med backend-systemet. Slik kunne vi raskt kunne etablere en fungerende brukerflate klar for samhandling med backenden så snart vi fant en stabil metode for å håndtere brukerforespørsler via API-et.

Imidlertid viste våre tidlige forsøk på modelltreningsmulighetene seg å være utilstrekkelige. Modellen leverte ikke den nødvendige graden av kontroll eller forutsigbarhet. Vi innså at vår begrensede erfaring med modelltrening gjennom maskinlæring, ledet til utfordringer med å opprettholde relevans og nøyaktighet i chatbotens respons, spesielt gitt risikoen for feilinformasjon og kravet om nøyaktighet i nyhetsinnhold.

Til tross for utfordringene, gitt prosjektets fokus på kunstig intelligens og vår sterke interesse for å utvide vår kompetanse, bestemte vi oss for å fortsette å bruke OpenAI's API der det var praktisk mulig. Dette sikret at vi kunne integrere avanserte AI-funksjoner i vår chatbot mens vi fortsatt lærte og tilpasset teknologien til våre behov.

3.11.4 Utvikling mot prototype

Etter den første brukertesten og møter med oppdragsgiveren, utviklet vi en prototype som inneholdt de fleste av de identifiserte hovedfunksjonene. Målet med denne prototypen var å presentere den i en ny brukertest for å samle verdifull feedback, som ville veilede den endelige utformingen av chatboten.

3.11.5 Hovedutvikling

Basert på tilbakemeldingene fra den andre brukertesten, identifiserte vi flere kritiske områder som trengte forbedring. Vi gjorde de nødvendige justeringene og gikk inn i en intensiv arbeidsperiode for å rette opp feilene som var avdekket. I løpet av de følgende ukene fokuserte teamet dedikert på å utvikle og forbedre nye funksjoner i chatboten. Dette arbeidet involverte både utvikling og design som sikret at alle aspekter av chatboten fungerte sømløst sammen.

Etter å ha arbeidet gjennom de mest prominente problemene, var neste steg å sikre at chatboten ikke bare fungerte teknisk, men også leverte en brukervennlig opplevelse. Vi tilbrakte flere uker på å forbedre brukergrensesnittet og brukeropplevelsen, inkludert å tilpasse dialogflyt og responsivitet i chatboten. Disse forbedringene ble grundig testet internt før vi presenterte en oppdatert versjon for oppdragsgiver.

3.11.6 Fremtidig arbeid

Etter prosjektets avslutning, er det flere veier for videre utvikling og forbedring av chatboten. Dette blir diskutert nærmere i 'Konklusjon av testene og fremtidig arbeid 5.10.5.' Utforskning av avanserte funksjoner som naturlig språkforståelse og integrering med flere databaser kan bidra til å forbedre chatbotens funksjonalitet og brukervennlighet.

4 Resultater

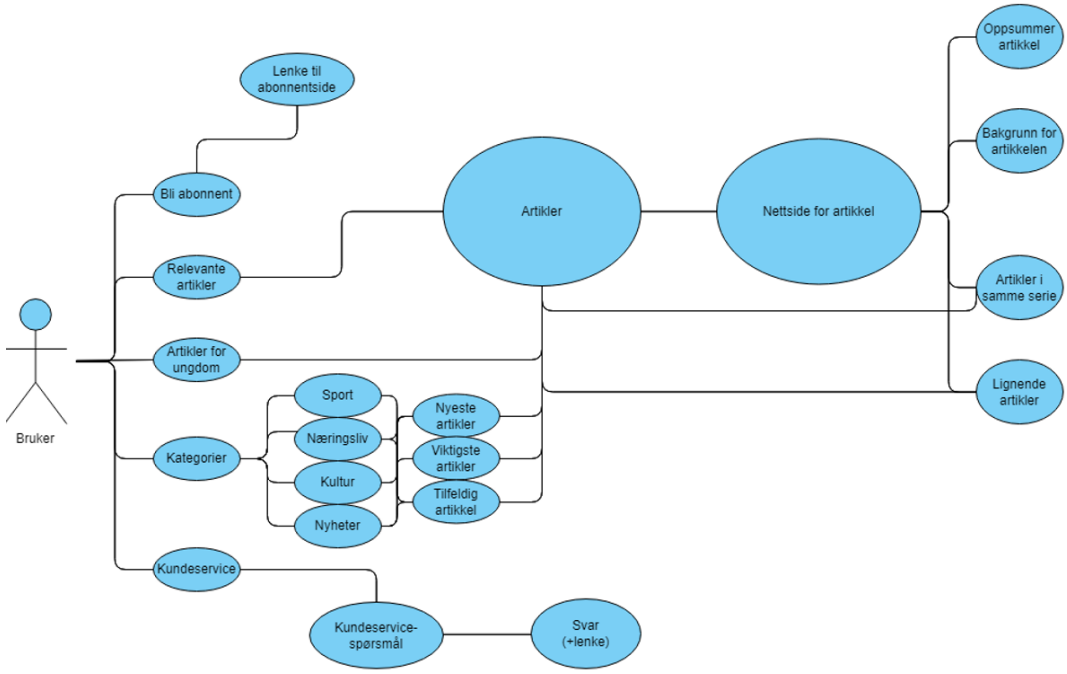
I dette kapittelet vil vi presentere resultatene og det endelige produktet for prosjektet vårt.

4.1 Generelle resultat

I løpet av prosjektet har vi utviklet og testet et produkt som nå inneholder alle planlagte funksjoner. Produktet er utviklet med standard JavaScript, og det endelige resultatet er en fullstack webapplikasjon som ble utviklet og testet lokalt på en enkel nettside. I løpet av utviklingsperioden gjennomførte vi flere brukertester, og avsluttet med en endelig test for å evaluere produktet og planlegge videre utvikling.

4.2 Use-Case

Gjennom oppgavebeskrivelsen identifiserte vi flere essensielle funksjoner som vår gruppe skulle implementere. Selv om vi hadde en relativt klar forståelse av de nødvendige funksjonene for chatboten, var integrasjonen av disse på en sømløs måte en utfordring. For å takle denne usikkerheten utviklet vi Use-Case-diagrammer gjennom hele prosjektet for å foreta justeringer i designet. Disse diagrammene gjorde det mulig for oss å skape en naturlig flyt i applikasjonen, hvor brukerne gradvis får tilgang til bestemte funksjoner, i stedet for å ha alle funksjoner tilgjengelig umiddelbart. Use-Case-diagrammet nedenfor illustrerer hvordan funksjonalitetsknappene fungerer. Når en bruker aktiverer boten, blir vedkommende presentert med en rekke ulike alternativer. Ved navigering til en artikkelside får brukeren tilgang til andre funksjoner spesifikt knyttet til artikler, og kan engasjere seg i samtaler om artikkelen ved hjelp av kunstig intelligens.



Figur 10: Use-case diagram

4.3 Funksjonaliteten til produktet

4.3.1 Introduksjon av Chatbotene

I dette kapittelet beskrives to forskjellige typer chatboter implementert for Sunnmørsposten: en standardbot som er tilgjengelig på forsiden, og en artikkelbasert bot som aktiveres når en bruker er inne på en artikkel.

4.3.2 Initialisering

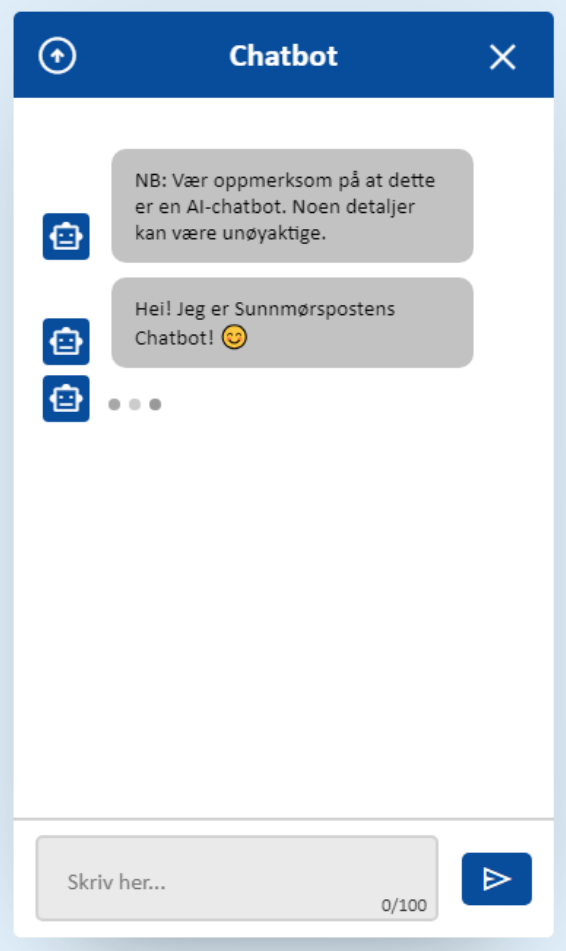
Figur 11: Toggler

Chatboten blir aktivert når brukeren klikker på en knapp som er plassert nederst til høyre på nettsiden. Ved å trykke på denne knappen utløses en jevn animasjon som ekspanderer chatbot-vinduet til full størrelse. Når utvidelsen er fullført, vil chatbot-vinduet se slik ut:

Figur 12: Chatbot før meldinger

I den øverste delen av chatbot-vinduet finner man to intuitive knapper. Den ene er merket med et 'X' og lar brukeren minimere vinduet. Den andre knappen, en sirkel med en pil opp, gjør det mulig for brukeren å raskt returnere til toppen av samtalen. Denne funksjonen ble utviklet på initiativ fra Sunnmørsposten for å forbedre brukeropplevelsen ved å lette navigasjonen innenfor et ofte innholdsrikt chatvindu. Dette var et nødvendig tiltak, da brukerne tidligere fant det utfordrende å bla tilbake gjennom lange svar for å utforske andre funksjoner i chatboten.

4.3.3 Forsideboten



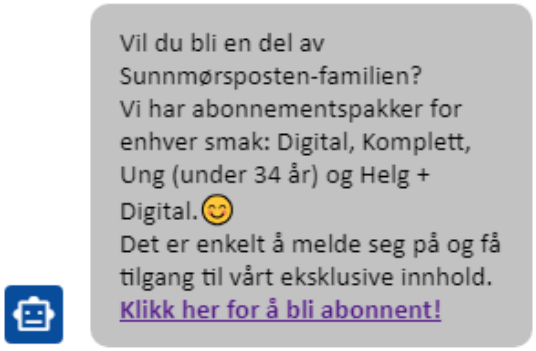
Figur 13: Chatbot med intromeldinger

Ved oppstart av chatboten mottar brukerne først en viktig advarsel om at de interagerer med en AI-drevet chatbot, og at visse opplysninger kan være unøyaktige. Denne tilnærmingen ble valgt for å opprettholde etisk integritet. Deretter presenterer chatboten seg selv som Sunnmørspostens chatbot. Dette tydeliggjør for brukeren hvilken organisasjon som står bak teknologien.

4.3.4 Funksjonalitetsknapper

Figur 14: Chatbot med funksjonalitetknapper

Etter chatboten har presentert seg selv, blir brukeren presentert med funksjonalitetsknappene. Her kan brukeren velge mellom en rekke alternativer. Brukeren kan trykke på knappen 'Bli abonnent' der hen får en kort forklaring i hvilke abonnementstyper som finnes på Sunnmørsposten, samt en direktelenke til abonnementsiden deres.

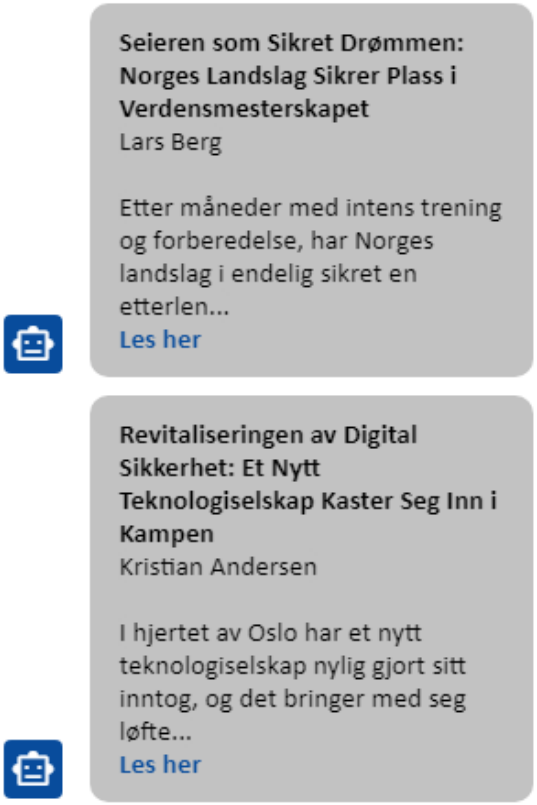


Figur 15: Bli abonnent

Hvis brukeren ønsker artikkelforslag, tilbyr chatboten tre tilgjengelige knapper: 'Relevante artikler', 'Artikler for Ungdom', og 'Kategorier'. Ved å velge 'Relevante artikler', vil chatboten søke i databasen etter de 20 nyeste artiklene og sortere dem etter relevans. Deretter presenterer den de tre mest betydningsfulle artiklene.

'Artikler for Ungdom'-knappen er designet for å øke engasjementet blant yngre lesere. Når denne aktiveres, viser chatboten de tre nyeste artiklene merket med 'Ungdom'.

Når chatboten foreslår artikler, får brukeren vist en oversikt over tittel og forfatter, sammen med et kort utdrag fra hver artikkel. Presentasjonen er i et oversiktlig og tiltalende format, med en tydelig 'Les her'-knapp under hvert utdrag som gir direkte tilgang til den fullstendige artikkelen.



Figur 16: Artikler utforming eksempel

Om brukeren ønsker å finne artikler basert på en gitt kategori kan hen velge 'Kategorier'-knappen. Da vil brukeren bli presentert med knapper basert på hvilke kategorier som finnes i databasen. Her kan hen velge hvilken kategori hen er mest interessert i.

Figur 17: Kategorier

Figur 18: Valgt kategori

Når brukeren har valgt en kategori blir hen presentert med tre nye knapper. Her kan brukeren velge mellom å se de tre nyeste artiklene, tre viktigste artiklene, eller en helt tilfeldig artikkel innen kategorien de har valgt.

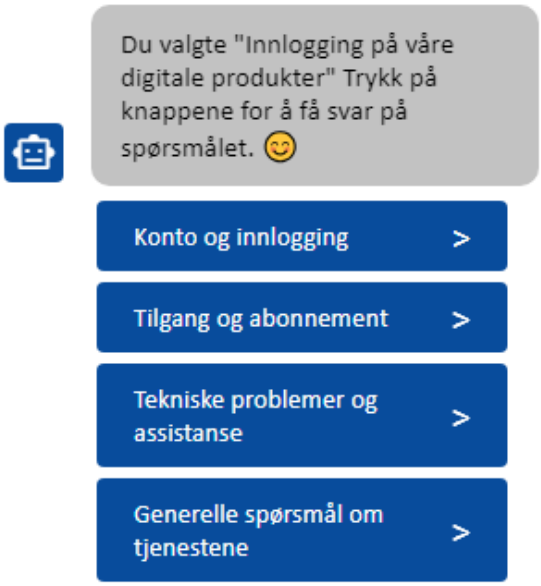
4.3.5 Kundeservice

Sunnmørsposten har en stor liste med kundeservicespørsmål på nettsiden deres. Vi ønsket at chatboten vår skulle håndtere kundeservice spørsmål på en mer ryddig og brukervennlig måte enn Sunnmørspostens nåværende løsning. Vi brukte derfor god tid på å organisere kundeservicespørsmålene i ulike kategorier og underkategorier. Slik ser det ut når du trykker på 'Kundeservice'-knappen:

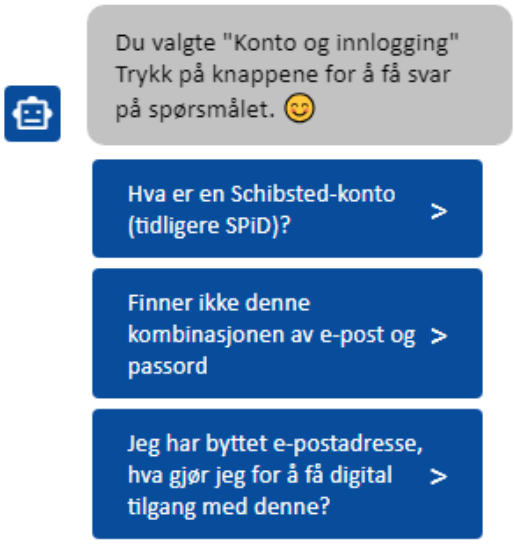
Dette er de samme kategoriene som finnes på Sunnmørsposten sin kundeservice-side. 'Innlogging på våre digitale produktet' og 'Betaling og faktura' inneholder veldig mange spørsmål, og da dette er en chatbot med sterkt begrenset plass ble vi nødt til å dele disse to kategoriene videre inn i underkategorier. For dette eksempelet tar vi for oss: 'Innlogging på våre digitale produktet' som er delt inn i fire underkategorier.

Figur 19: Tilfeldig artikkel sport

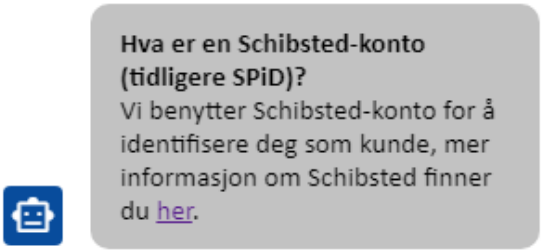
Figur 20: Kundeservice kategorier



Figur 21: Innlogging på våre digitale produkter



Figur 22: Konto og innlogging

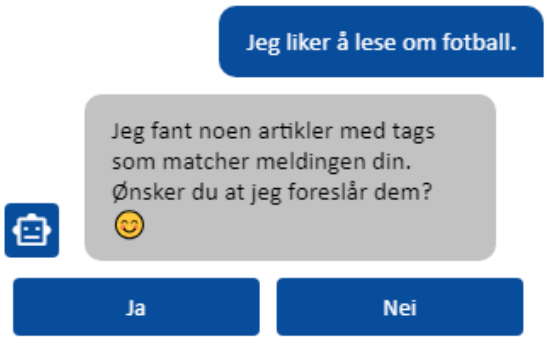


Figur 23: Spørsmål og svar eksempel

Denne løsningen sikrer at brukeren ikke må lese gjennom en overveldende mengde spørsmål for å finne det de lurer på. I stedet kan de bruke en brukervennlig filtreringsmetode for å effektivt navigere seg fram til ønsket informasjon. Når brukeren finner og velger det aktuelle spørsmålet, vil chatboten levere et klart og oversiktlig svar, der spørsmålet er fremhevet med fet skrift og svaret vises i vanlig skrift. Eventuelle lenker som chatboten presenterer er klikkbare og leder direkte til den tilsvarende ressursen.

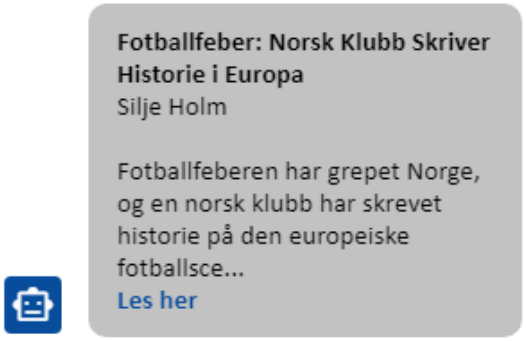
4.3.6 Chatting med chatboten

Chatfeltet er en nyttig funksjon for å raskt få anbefalt artikler basert på dine interesser. Om brukeren sin melding inneholder et nøkkelord som også er en tag i databasen, vil chatboten foreslå å vise disse artiklene for brukeren. (I dette eksempelet, 'fotball')



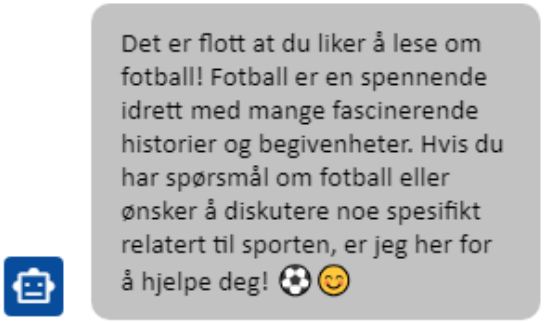
Figur 24: Foreslå artikler basert på brukerens melding eksempel

Om brukeren velger 'Ja', vil hen bli presentert med maksimalt tre artikler som inneholder fotball som tag, sortert etter nyeste først.



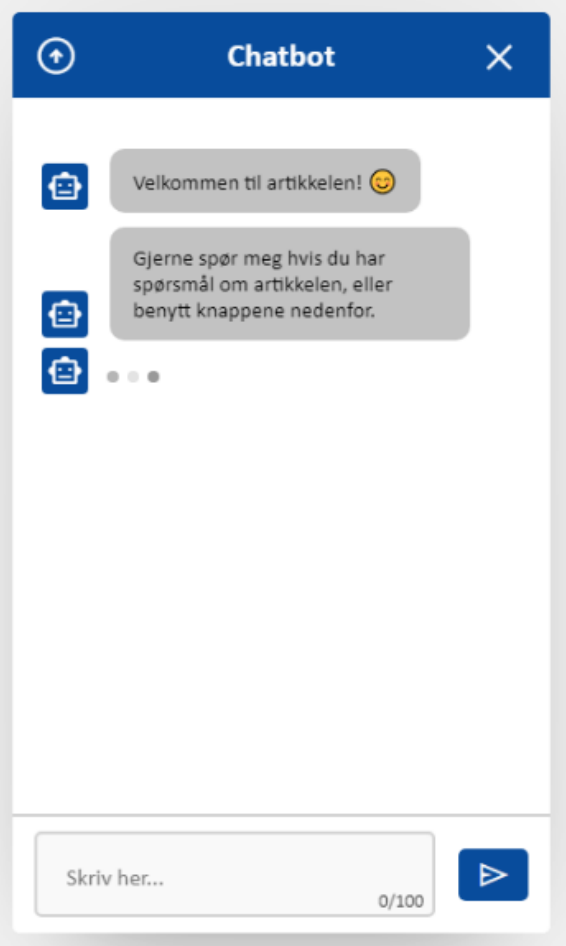
Figur 25: Alternativ 'Ja' eksempel

Om brukeren velger 'Nei', vil meldingen sendes til GPT-modellen som vil svare på brukerens tekstprompt så godt den kan.



Figur 26: Alternativ 'Nei' eksempel

4.3.7 Artikkelboten



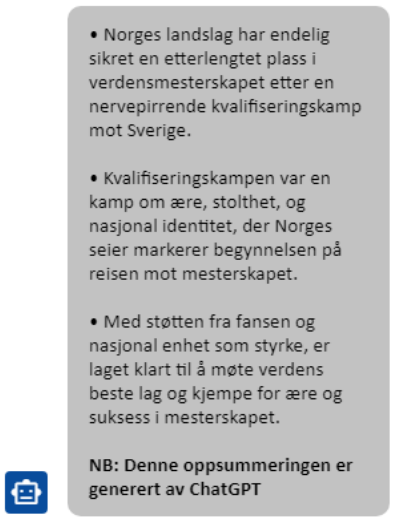
Figur 27: Artikkelbot velkomst

Når brukeren er inne på en artikkel, blir de introdusert til en ny chatbot. Denne chatboten ønsker brukeren velkommen med en melding spesifikt tilpasset artikkelboten og oppfordrer til å stille spørsmål om artikkelen eller å utforske de funksjonelle knappene nedenfor.

4.3.8 Funksjonalitetknapper artikkelbot

Figur 28: Artikkelbot

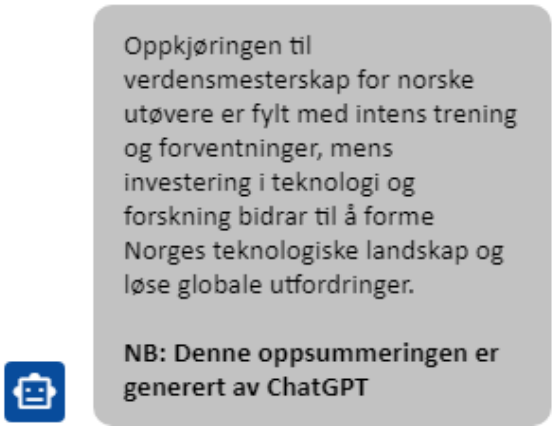
Funksjonalitetknappene på artikkelboten er helt annerledes enn forsideboten. Her får brukeren en rekke alternativer. Ved å trykke på 'Oppsummer artikkel'-knappen får brukeren en oppsummering av artikkelen på tre korte punkt, dynamisk generert av ChatGPT.



Figur 29: Oppsummering eksempel

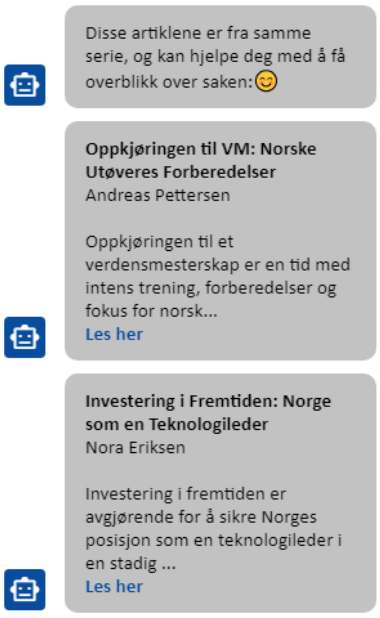
Knappen 'Bakgrunn kort forklart' er designet for å gi brukerne en oversikt over konteksten bak artiklene. På Sunnmørsposten er artiklene ofte del av en større 'Story', hvor forståelsen av helheten er essensiell for å fullt ut sette seg inn i og forstå den enkelte artikkelen. Når denne funksjonen aktiveres, søker den gjennom databasen etter andre artikler fra samme 'Story' og bruker ChatGPT til å generere en kortfattet oppsummering. Dette sikrer at brukeren er godt informert om bakgrunnen for saken.

I eksempelet nedenfor finner du en oppsummering basert på to artikler fra samme 'Story' som den aktuelle artikkelen. De benyttede artiklene er hentet fra vår database, som inneholder tilfeldig genererte dummy-artikler. Disse er ikke nødvendigvis sammenhengende på samme måte som en ekte artikkeldatabase ville vært, men dette illustrerer likevel funksjonaliteten effektivt.



Figur 30: Bakgrunn kort forklart

'Artikler i samme serie'-knappen gir brukeren tilgang til andre artikler som tilhører samme 'Story' som den aktuelle artikkelen. Dette er nyttig hvis 'Bakgrunn kort forklart'-funksjonen ikke gir nok kontekst, og brukeren ønsker å lese artiklene i sin helhet. Disse artiklene danner også grunnlaget for oppsummeringen om bakgrunnen til artikkelen.

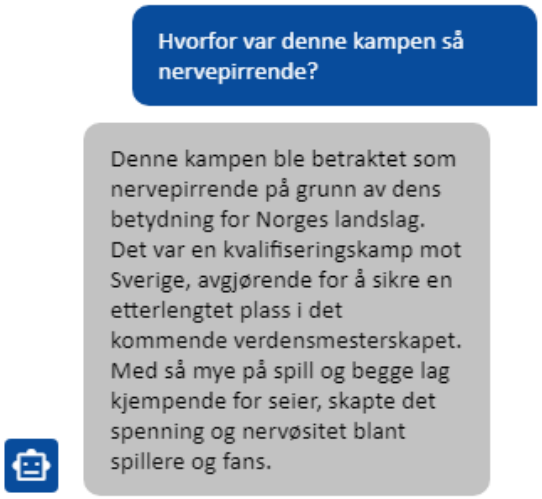


Figur 31: Artikler i samme serie

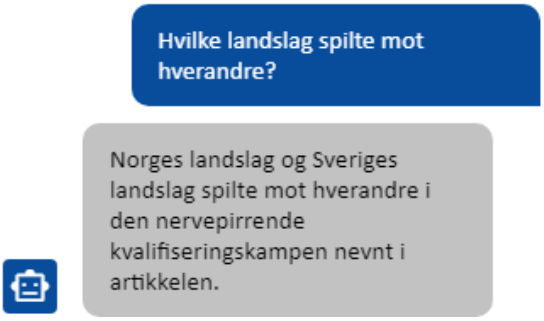
'Lignende artikler'-knappen søker gjennom databasen etter artikler med tags som matcher de i den aktuelle artikkelen, og rangerer dem etter antall like tags. Chatboten presenterer deretter de tre mest relevante artiklene til brukeren.

4.3.9 Chatting med artikkelboten

Chatfeltet i artikkelboten byr på de samme grunnfunksjonene som forsideboten, men inkluderer ytterligere funksjoner tilpasset artikkelinnhold. Brukerne kan stille spesifikke spørsmål om artikkelen og få tilpassede svar fra GPT-modellen.



(a) Spørsmål fra bruker om artikkelen eksempel 1



(b) Spørsmål fra bruker om artikkelen eksempel 2

Figur 32: Eksempler på brukerspørsmål om artikler

4.4 Design

Vi har investert betydelig tid og energi for å sikre at vår chatbot ikke bare er like tiltalende og brukervennlig som de beste chatbotene på norske nettsider, men i mange tilfeller, forhåpentligvis enda bedre. Design er en avgjørende komponent i suksessen til en chatbot; det påvirker direkte brukernes vilje til å engasjere seg med verktøyet. Gitt at chatboten representerer et viktig tiltak fra Sunnmørsposten for å øke leserengasjementet, har vi gjennomført flere designendringer gjennom dens utvikling for å sikre at den er så visuelt tiltalende og funksjonell som mulig. I de kommende avsnittene vil vi gå inn på de spesifikke designvalgene som er gjort og hvorfor disse er viktige for brukeropplevelsen.

4.4.1 Inspirasjon og overordnede designprinsipper

Før vi dykker dypere inn i de spesifikke designbeslutningene for prosjektet, er det verdt å nevne at mye av chatbotens design er inspirert av 'boost.ai', en ledende norsk leverandør av konversasjonell KI. Eksempler på kjente norske nettsteder som bruker chatboter fra boost.ai inkluderer 'Posten.no' og 'Helsenorge.no.'

4.4.2 Visuelle designelementer

Chatvinduets proporsjoner Gjennom flere utkast har vi kommet fram til at det beste høyde-til-bredde forholdet for et chatbot-vindu er cirka 1,7. Denne proporsjonen kombinert med en mild avrunding i kantene gir et profesjonelt og behagelig utseende.

Avrunding i kantene Chatvinduet, knappene, tekstfeltet og send-knappen har alle samme avrunding på kantene, noe som skaper en behagelig og harmonisk estetikk som forbedrer brukeropplevelsen. Meldingsboksene har en noe større avrunding for å etterligne utseendet til en snakkeboble, men de er nøye tilpasset slik at de fortsatt harmonerer godt med de andre designelementene.

Fargekontraster Fargepaletten i chatboten består hovedsakelig av blått, hvitt og grått. Blåfargen er hentet direkte fra Sunnmørspostens logo og nettsidedesign, noe som naturlig gjorde den til hovedfargen for chatboten.

Figur 33: Sunnmørsposten logo

Botens bakgrunn er hvit, noe som gir en klar kontrast til den mørkere gråfargen brukt i meldingsboksene, samt den mildere gråfargen som dominerer Sunnmørspostens nettside.

4.4.3 Interaktivitetsdesign og brukerengasjement

Responsivitet og feedbackmekanismer Når en bruker trykker på en knapp, gir knappen visuell tilbakemelding ved å vises som om den blir 'trykket inn'. I tillegg skifter knappen til en mørkere blåfarge når brukeren holder musepekeren over den, noe som forenkler valget av riktig knapp.

(a) Hover over knapp

(b) Feedbackmekanisme på inntrykket knapp

Figur 34: Eksempel på knapp som trykkes inn

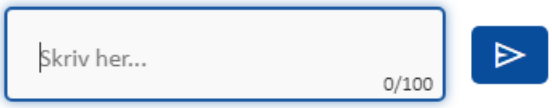
Interaksjonsbegrensninger Under Prosessering Når brukeren har trykket på en knapp, eller sendt en melding blir knappene og tekstfeltet deaktivert. Knappene blir svakere i fargene, og pekeren får et 'forbudt'-symbol når den hovrer over dem. Tekstfeltet blir lysegrått og får også en 'forbudt'-peker over seg hvis brukeren hovrer over.

(a) Svake farger og 'forbudt'-symbol under prosessering

(b) Tekstfelt forbudt

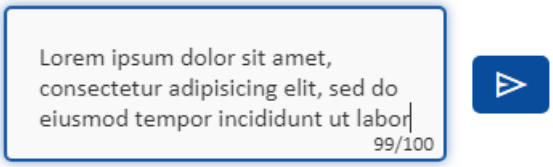
Figur 35: Eksempel på interaksjonsbegrensninger

Markering av Tekstfelt Tekstfeltet får en tiltalende blå kant når det er aktivt, som blir synlig gjennom en myk animasjon på 0,3 sekunder. Dette tydeliggjør for brukeren at tekstfeltet er markert og klart til å ta imot tekst.

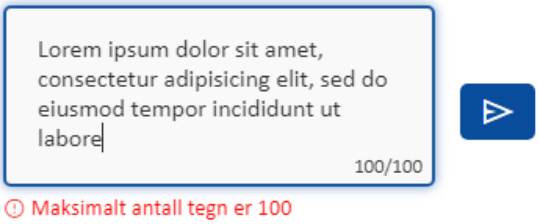


Figur 36: Tekstfelt markert

Dynamiske elementer Tekstfeltet er designet for å være responsivt; det ekspanderer automatisk med én linje hver gang brukerens melding overstiger én linje. Hvis meldingen overstiger grensen på 100 tegn, vil en glidende advarselsanimasjon informere brukeren om at de har nådd maksimalt antall tillatte tegn. Brukeren kan overvåke antallet brukte tegn i sanntid gjennom en teller plassert nederst til høyre i tekstfeltet. Når grensen på 100 tegn er nådd, forhindres ytterligere inntasting automatisk.



Figur 37: Ekspandert tekstfelt



Figur 38: Ekspanderende tekstfelt med maksimalt antall tegn

4.5 Backend teknologi

Introduksjon Backenden til vårt prosjekt er designet for å støtte en rekke funksjonaliteter nødvendig for frontenden gjennom definerte 'endepunkt'. Disse endepunktene håndterer GET-, og POST-forespørsler som er sentrale for å hente og prosessere data, særlig artikler, fra databasen. Valget av teknologier og strukturering av backenden reflekterer behovet for effektivitet og integrasjon med Sunnmørspostens eksisterende digitale infrastruktur.

4.5.1 Serverinitialisering

- **Importering av moduler:** Vi bruker ES-modulsyntaks for å importere alle nødvendige biblioteker. Dette inkluderer biblioteker for Express, body-parser, CORS, og andre avhengigheter som er beskrevet i 'Programmering og språk 3.4.5.'
- **Konfigurering av miljøvariabler:** Ved å kjøre dotenv.config(), lastes konfigurasjonsinnstillinger fra .env-filen inn i process.env. Dette trinnet er nødvendig for å administrere sensitiv informasjon som API-nøkler for OpenAI og Supabase på en sikker måte, og støtter vår backend i å operere effektivt under ulike miljøer.
- **Initialisering av Express:** Vi initialiserer en Express-applikasjon ('app'), som danner grunnlaget for vår server. Denne Express-applikasjonen blir brukt i hvert eneste endepunkt i backenden.
- **Middleware-konfigurasjon:** Applikasjonen konfigurerer nødvendig middleware for optimal drift:

  bodyParser.json() for å tolke innkommende JSON-data.

  cors() for å aktivere forespørsler på tvers av domener.

  express.static("public") for å tilby statiske ressurser, som forbedrer brukertilgjengeligheten og ytelsen til frontenden.
- **Supabase klient-initialisering:** Med Supabase API-nøkkelen lagret fra miljøvariablene, initialiseres Supabase-klienten ved å kalle createClient. Dette forbereder backenden for å utføre dataoperasjoner, som å hente artikler fra den tilknyttede databasen.
- **Serverlytting:** Serveren er konfigurert til å lytte på port 3000, et standard utviklingsport for lokale servere. Denne porten brukes når serveren starter, som markerer at initialiseringen er komplett og serveren er klar til å motta forespørsler.

4.5.2 Databasekonfigurasjon

Databasen er konfigurert ved hjelp av `'createClient'`-funksjonen fra Supabase. Konfigurasjonen starter med importering av nødvendige biblioteker:

```
import { createClient } from '@supabase/supabase-js';
```

Følgende kommando laster inn miljøvariablene:

```
dotenv.config();
```

Dette skjer i applikasjonens rotkatalog hvor filen '.env' holder sensitive nøkler skjult fra offentligheten. Deretter initialiseres Supabase-klienten som følger:

```
const supabaseUrl = process.env.SUPABASE_URL;
const supabaseKey = process.env.SUPABASE_ANON_KEY;
const supabase = createClient(supabaseUrl, supabaseKey);
```

Denne klienten er avgjørende og benyttes i nesten alle API-endepunktene for å utføre databasemanipulasjoner som:

- Å hente og filtrere artikler basert på kategorier eller tagger.
- Oppdatere 'chatHistory' for å forbedre kvaliteten på samtaler.
- Bruk av asynkrone funksjoner som 'async/await' for effektiv databehandling.

4.5.3 Database

Name	Description	Data Type	Format
id	unique article indentifier	bigint	int8
title	the title of the article	text	text
author	the author of the article	text	text
content	the contents of the article	text	text
tags	the tags related to the article	text	text
category	the category of the article	text	text
publication_date	the publication date of the article	timestamp with time zone	timestamptz
viktighetsgrad	how important is the article. (1-5) (low-high)	smallint	int2
url	the url to the article	text	text
series	what series of articles does this belong in (story)	text	text

Figur 39: Database kolonner

På bildet presenteres de forskjellige kolonnene som bidrar til at chatboten kan sortere og velge riktige artikler for brukeren. Videre vil vi forklare noen av kolonnene som ikke er helt selvforklarende:

- Tags: Dette feltet representerer ulike temaer relatert til artikkelen. En utgiver kan tilføye flere tags til en artikkel, noe som øker sannsynligheten for at chatboten foreslår denne artikkelen til brukeren. Dette fordi funksjonen 'Lignende artikler' prioriterer artikler som deler mange tags med brukerens aktuelle søk eller interesse.
- Viktighetsgrad: Dette er en kolonne som Sunnmørsposten har i sine systemer, hvor utgiveren av artikkelen angir en verdi fra én til fem, hvor fem indikerer høyest viktighet på artikkelen. Denne kolonnen er nødvendig for for funksjonene 'Relevante artikler', og 'Viktigste artikler' da systemet fremhever artikler med høyere viktighetsgrad.
- Series: Denne kolonnen er inspirert av Sunnmørspostens 'Story-tag'-funksjon, som binder sammen artikler relatert til samme tema eller hendelsesforløp. Denne funksjonen er designet for å gjøre det enklere for brukeren å forstå bakgrunnen for en artikkel. 'Series'-kolonnen blir benyttet av funksjonalitetene 'Bakgrunn kort fortalt' og 'Artikler i samme serie', for å tilby en sammenhengende leseropplevelse.

4.5.4 API-endepunkter og funksjonalitet

- *GET-forespørsler:* Ber om data fra en spesifisert ressurs.
- *POST-forespørsler:* Sender data til en server for å opprette en ressurs.

Nedenfor er en liste over alle API-endepunktene i vår backend:

GET Endepunkter

1. **/relevantArticles**
   Henter de 20 mest nylige artiklene og returnerer de tre mest relevante basert på viktighetsgrad.
2. **/articlesUngdom**
   Henter artikler merket med "Ungdom", begrenset til de tre nyeste.
3. **/categories**
   Henter og returnerer unike kategorier fra artikler.
4. **/Articles/:id**
   Henter en spesifikk artikkel basert på dens ID.
5. **/Articles/:category**
   Henter alle artikler innen en spesifikk kategori.
6. **/Articles/:category/latest**
   Henter de tre nyeste artiklene innen en spesifikk kategori.
7. **/Articles/:category/important**
   Henter de tre viktigste artiklene basert på viktighetsgrad innen en spesifikk kategori.
8. **/Articles/:category/random**
   Henter en tilfeldig artikkel innen en spesifikk kategori ved hjelp av en database-funksjon (RPC).
9. **/summarizeArticle/:id**
   Henter innholdet til en spesifikk artikkel og oppretter en kort sammendrag av innholdet med OpenAI GPT-3.5-turbo.
10. **/articlesInSeries/:id**
    Henter alle artikler som er del av samme serie som den gitte artikkelen, basert på artikkelens ID.
11. **/similarArticles/:id**
    Henter artikler som har lignende tagger som en spesifikk artikkel, sortert etter antall overlappende tagger.

POST Endepunkter

1. **/ask**
   Håndterer brukerforespørsler ved å først søke etter svar i FAQ eller tagger i meldingen. Deretter søkes det etter relaterte artikler. Hvis ingen artikler blir funnet, spør den OpenAI om hjelp.

2. **/askOpenAIForResponse**
   Sender brukerens melding direkte til OpenAI GPT-modellen for å få et svar uten å gå gjennom andre trinn.

3. **/summarizeMultipleArticlesBackstory**
   Oppretter en kort sammendrag av flere artikler basert på sammensatt innhold fra forespørselen.

4.5.5 Integrering og behandling av data i API-endepunkter

I dette avsnittet vil vi forklare hvordan Supabase-klienten er integrert i et spesifikt API-endepunkt for å hente ut artikler fra vår database. Vi vil demonstrere prosessen for å sortere og filtrere dataene ved hjelp av /relevantArticles-endepunktet.

Trinn 1: Henting av Artikler Det første trinnet involverer en forespørsel til 'Articles'-tabellen i databasen, hvor vi benytter Supabase-klienten til å utføre en SELECT-spørring som henter alle kolonner (*) fra tabellen. Vi sorterer resultatene etter 'publication_date' i synkende rekkefølge og begrenser utvalget til de 20 nyeste artiklene. Dette sikrer at vi fokuserer på de ferskeste artiklene tilgjengelig i databasen.

```
app.get('/relevantArticles', async (req, res) => {
  try {
    const { data: recentArticles, error: recentError } = await
        supabase
        .from('Articles')
        .select('*')
        .order('publication_date', { ascending: false })
        .limit(20);
```

Trinn 2: Filtrering Etter at de innledende artiklene er hentet, gjennomgår vi en filtreringsprosess hvor artiklene sorteres etter deres 'viktighetsgrad' for å identifisere de mest betydningsfulle stykkene. Dette gjøres ved å bruke JavaScripts innebygde sort()-funksjon, som sammenligner 'viktighetsgrad' av hver artikkel. De tre øverste artiklene blir valgt ut basert på denne vurderingen, og disse blir deretter presentert for brukeren som dagens mest relevante artikler. Det er verdt å merke at ikke alle endepunkter krever en slik filtreringsprosess, men for /relevantArticles-endepunktet er det nødvendig for å kombinere de nyeste og viktigste artiklene.

```
    if (recentArticles && recentArticles.length > 0) {
        const topArticles = recentArticles
            .sort((a, b) => b.viktighetsgrad - a.viktighetsgrad)
            .slice(0, 3);
        res.json({
            message: "Dette er de siste og mest relevante artiklene
                for i dag:",
            articles: topArticles
        });
```

Denne totrinnsprosessen fra henting til filtrering illustrerer hvordan vi bruker Supabase-klienten effektivt for å hente data og hvordan JavaScripts innebygde funksjoner brukes til å sortere dataen før de mest relevante artiklene sendes til frontend for behandling som variabelen 'articles'.

4.5.6 Håndtering av brukerinteraksjoner og databehandling

Chat-historikk 'chatHistory' er en avgjørende variabel i backenden som spiller en nøkkelrolle i funksjonaliteten til vår AI-chatbot. Denne variabelen holder styr på alle interaksjoner i chatten, både fra brukeren og fra GPT-modellen. Den inneholder også viktig informasjon om artikler som tittel, forfatter og hovedinnhold. 'chatHistory' bidrar også til å definere og opprettholde chatbotens 'personlighet' ved oppstart. Dette initialiseres med følgende prompt:

```
let chatHistory = [
  {
    role: 'system',
    content: `You are a helpful AI chatbot for Sunnmørsposten that
    answers questions with a smile. Communicate briefly and
    precisely in Bokmål Norwegian.

    Follow these guidelines:

    1. Always base your answers on verified information.
    Avoid making up or assuming information that is not
    explicitly provided or available in documents you have read.

    2. If you mention specific individuals,
    such as journalists, ensure to use only information
    from the document you have been assigned, without adding
    or altering the information.

    3. Be clear and concise in your responses, and avoid
    lengthy explanations. Aim to provide the information
    that is necessary and relevant to the user's question.

    4. Use friendly and approachable language, but keep the
    focus on facts and provided information.

    5. If a user asks to recommend an article, you will encourage
    the user to write tags they are interested in.

    6. Otherwise make sure to recommend to the user to utilize
    the buttons above, as they are very helpful.`
  }
];
```

Denne prompten sikrer at GPT-modellen oppfører seg korrekt og konsistent gjennom hele brukeropplevelsen. Modellen vil alltid følge disse retningslinjene.

Kode-snuttene nedenfor viser hvordan brukerens meldinger og chatbotens svar håndteres:

```
const userInput = req.body.message;
chatHistory.push({ role: 'user', content: userInput });
```

```
let responseText = openAIResponse.data.choices[0].message.content
    .trim();
chatHistory.push({ role: 'system', content: responseText });
```

Videre blir artikkelinformasjon integrert i 'chatHistory' slik at chatboten kan inngå en dialog med brukeren om artikkelens innhold:

```
      if (data) {
    // Update chat history with article's title, author, and
       content
    chatHistory.push({
      role: 'system',
      content: `Article Title: ${data.title}`
    });
    chatHistory.push({
      role: 'system',
      content: `Article Author: ${data.author}`
    });
    chatHistory.push({
      role: 'system',
      content: `Article Content: ${data.content}`
    });
```

Ved å lagre chat-historikken som chatboten har med brukeren, kan chatboten og brukeren ha en mer naturlig dialog. Videre er integreringen av artikkelinformasjon i 'chatHistory' avgjørende for at chatboten skal være et nyttig verktøy som brukeren faktisk vil benytte.

4.5.7 Dynamisk innholdshåndtering

Chatboten tilbyr to hovedmetoder for å foreslå artikler til brukeren: direkte gjennom interaktive knapper og basert på brukerens inntastede meldinger.

I den første metoden kan brukeren navigere gjennom ulike kategorier og temaer ved å klikke på interaktive knapper. Disse knappene er forhåndsprogrammert til å foreslå de mest relevante artiklene innenfor hver kategori, som gir en intuitiv og brukervennlig måte å utforske innhold på.

```
app.get('/articlesUngdom', async (req, res) => {
  const tag = 'Ungdom';
  const { data, error } = await supabase
    .from('Articles')
    .select('*')
    .ilike('tags', `%${tag}%`)
    .order('publication_date', { ascending: false })
    .limit(3);

  if (error) {
    console.error('Error fetching Articles tagged with Ungdom:',
       error);
    return res.status(500).send('Error fetching Articles');
  }

  if (data.length === 0) {
    return res.status(404).send('No articles found with the tag
       Ungdom');
  }

  res.json(data);
});
```

Når en bruker trykker på for eksempel 'Artikler for Ungdom'-knappen, initieres en 'GET'-forespørsel til et dedikert API-endepunkt i backenden. Backend-serveren behandler forespørselen ved å hente relevant informasjon fra databasen, som demonstrert i kode-snutten ovenfor.

I den andre metoden analyserer chatboten tekst inntastet av brukeren for å identifisere nøkkelord eller 'tags'. Denne tekstanalysen gjør det mulig å tilpasse artikkelforslagene til brukerens individuelle interesser, og skaper en personlig brukeropplevelse. Nedenfor er en kode-snutt fra /ask-endepunktet.

```
    const tags = await findTagsInMessage(userInput);
```

Denne funksjonen, `findTagsInMessage()`, kaller på `getAllTags()` for å sjekke om brukerens melding inneholder 'tags' som finnes i databasen.

```
async function findTagsInMessage(userInput) {
  const allTags = await getAllTags();
  const inputLower = userInput.toLowerCase();
  const foundTags = allTags.filter(tag => inputLower.includes(tag))
  return foundTags;
}
```

```
async function getAllTags() {
  let { data: tags, error } = await supabase
      .from('Articles')
      .select('tags');
  if (error) {
      console.error('Error fetching tags:', error);
      return [];
  }
  const allTags = new Set();
  tags.forEach(article => {
      article.tags.split(',').forEach(tag => allTags.add(tag.trim()
          .toLowerCase()));
  });
  return Array.from(allTags);
}
```

Hvis en relevant 'tag' identifiseres, og det finnes artikler assosiert med denne, vil brukeren få følgende respons:

```
    "Jeg fant noen artikler med tags som matcher meldingen din.
     Ønsker du at jeg foreslår dem?"
```

Brukeren får da muligheten til å velge 'Ja' for å se artiklene eller 'Nei' for å fortsette samtalen. Dette valget styrer om artikler blir foreslått direkte eller om brukerens forespørsel sendes til GPT-modellen for videre behandling.

/ask-endepunktet og dynamisk innholdsgenerering med OpenAI I forrige avsnitt så vi på hvordan '/ask'-endepunktet håndterer meldinger som inneholder en tag. I dette avsnittet utforsker vi hvordan endepunktet opererer når meldinger ikke inneholder tags. Først ser vi på startprosessen for endepunktet, illustrert nedenfor:

```
app.post('/ask', async (req, res) => {
  const userInput = req.body.message;
  chatHistory.push({ role: 'user', content: userInput }); // Log user input to chat history

  let response = [];
```

Endepunktet initieres ved å legge brukerens melding til i chatte-historikken. Deretter initialiseres en tom array kalt 'response'. Denne arrayen fylles dynamisk med svarinnhold basert på evalueringen av brukerens forespørsel.

Dersom endepunktet ikke finner noen tags i meldingen, som vist tidligere, vil meldingen bli sendt videre til OpenAI's GPT-modell:

```
      if (response.length === 0) {
        await askOpenAIForResponse(userInput);
      }
```

Funksjonen som håndterer dette kallet ser slik ut:

```
async function askOpenAIForResponse() {
  try {
    const openAIResponse = await openai.createChatCompletion({
      model: 'gpt-3.5-turbo',
      messages: chatHistory,
      max_tokens: 300,
    });
    let responseText = openAIResponse.data.choices[0].message.content.trim();
    response.push({ type: 'text', content: responseText });
    chatHistory.push({ role: 'system', content: responseText });
  } catch (error) {
    console.error("Error with OpenAI:", error);
    let errorMessage = "Sorry, I encountered an error processing your request.";
    response.push({ type: 'text', content: errorMessage });
    chatHistory.push({ role: 'system', content: errorMessage });
  }
}});
```

createChatCompletion Funksjonen gjør et asynkront kall til OpenAIs API ved hjelp av metoden createChatCompletion. Metoden genererer et svar basert på den konteksten den mottar.

I klammeparantersen under `createChatCompletion` finnes følgende felt:

```
    model: 'gpt-3.5-turbo',
    messages: chatHistory,
    max_tokens: 300,
```

- **Model: 'gpt-3.5-turbo'** er modellen vi bruker. Vi valgte denne modellen fordi den er rask og tilstrekkelig kraftig for våre behov. Selv om det finnes kraftigere modeller, gir 'gpt-3.5-turbo' en optimal balanse mellom ytelse og hastighet for vårt bruk.
- **messages: chatHistory** gir modellen nødvendig kontekst for å opprettholde samtaleflyten og generere et passende svar. Som nevnt tidligere, inneholder chatHistory en samling av tidligere meldinger og artikkelinformasjon.
- **'Max tokens: 300'** er en grense for hvor lang en responsmelding kan være. Med en verdi på '300' tilsvarer det cirka 40-50 ord, som er det maksimale antall ord vi ønsker at boten skal kunne gi i en respons.

Behandling av svaret Når svaret returneres fra OpenAI, henter funksjonen svarteksten fra responsobjektet:

```
    let responseText = openAIResponse.data.choices[0].message.
        content.trim();
```

'choices[0]' velger den første responsen og bruker 'trim()' for å fjerne eventuelle overflødige mellomrom eller linjer.

Nå som responsen er generert og gjort klart i arrayen 'responseText', blir det pushet til arrayen 'response' som et objekt av typen 'text'. Det er teksten som skal sendes tilbake til brukeren. Responsen legges også til i chatHistory:

```
    response.push({ type: 'text', content: responseText });
    chatHistory.push({ role: 'system', content: responseText });
```

4.5.8 Oppsummering av artikkel med ChatGPT

Nå vil vi utforske en kjernefunksjon i chatbot-prosjektet: Evnen til å generere strukturerte artikkelsammendrag. Funksjonen er implementert gjennom endepunktet /summarizeArticle.

Først utføres en SELECT-forespørsel for å hente innholdet fra den aktuelle artikkelen. Deretter konstrueres en prompt som lagres i variabelen prompt:

```
const prompt =

`You will include this message at the end with ONE line of space:
    <strong>NB: Denne oppsummeringen er generert av ChatGPT</
   strong>. Write an ULTRA-SHORT summary in norwegian about this
   article in ONLY THREE bullet points that are SEPERATED WITH
   PARAGRAPHS using the symbol •: \n\n${article.content}`;
```

Bruken av blokkbokstaver i prompten sikrer at modellen følger retningslinjene nøyaktig. Inkluderingen av en advarsel om at sammendraget er generert av ChatGPT sikrer etiske formål og gir åpenhet for leseren.

```
try {
  const openAIResponse = await openai.createChatCompletion({
    model: 'gpt-3.5-turbo',
    messages: [{role: 'system', content: prompt}],
    max_tokens: 300,
  });

  const summary = openAIResponse.data.choices[0].message.content.
      trim();
  res.json({ summary });
}
```

Prompten sendes deretter til GPT-modellen via `createChatCompletion`-funksjonen, og resultatet returneres som en JSON-strukturert summary som behandles videre i frontend.

4.5.9 Feilhåndtering

Generell håndtering av feil Feilhåndtering er en kritisk del av backenden, særlig når det er interaksjon med eksterne databaser som Supabase. Vi bruker try/catch-blokker for å håndtere potensielle feil i asynkrone funksjoner. Dette gjør det mulig å fange opp og håndtere unntak på en kontrollert måte. Her er et eksempel på hvordan vi håndterer feil når vi henter relevante artikler:

```
1     app.get('/relevantArticles', async (req, res) => {
2       try {...
3       } catch (error) {
4         console.error('Error fetching relevant Articles:', error);
5         res.status(500).send("Det oppsto en feil under henting av
              artikler.");
6       }
7     });
```

I dette eksemplet logger serveren feilen til konsollen og sender en feilmelding tilbake til klienten med HTTP-statuskode 500, noe som indikerer en intern serverfeil. Dette gir nyttig tilbakemelding til klienten om at noe gikk galt, samtidig som det gir verdifull informasjon for debugging og vedlikehold.

Håndtering av spesifikke scenarioer For GET-forespørsler som ikke finner noen artikler tilknyttet en spesifikk tag eller kategori, brukes følgende håndtering:

```
1     if (data.length === 0) {
2     return res.status(404).send('No articles found with the tag
          Ungdom');
```

Her returnerer serveren en HTTP-statuskode 404, som indikerer at de forespurte dataene ikke ble funnet i databasen under de gitte kriteriene. Korrekt bruk av statuskoder er viktig for å klarlegge type feil, da det hjelper klienten med å forstå hva som gikk galt.

Håndtering av manglende innhold Når nødvendig innhold mangler i en forespørsel, som i følgende eksempel med en POST-forespørsel, bruker vi statuskoden 400 for å indikere feil bruk av APIen:

```
1     app.post('/summarizeMultipleArticlesBackstory', async (req, res
          ) => {
2   const { contents } = req.body;
3
4   if (!contents) {
5     return res.status(400).send('No contents provided');
6   }
```

I dette tilfellet sjekker vi om contents, som inneholder teksten som skal oppsummeres, eksisterer i forespørselens kropp. Hvis den ikke finnes, svarer serveren umiddelbart med statuskode '400' og en feilmelding 'No contents provided.'

4.5.10 Oppsummering

Denne underseksjonen har detaljert hvordan backenden til vår KI-chatbot er designet og implementert. Vi har valgt Node.js og Express.js for å integrere smidig med Sunnmørspostens eksisterende systemer, et nødvendig kriterium for å oppnå ønsket effektivitet i et sanntidsprosjekt som en chatbot. Vår beslutning om å bruke Supabase for håndtering av en enkel dummy-database, samt OpenAI sine AI-modeller, har gjort backenden både robust og skalérbar.

Videre har bruk av body-parser sikret effektiv håndtering av JSON-data, mens CORS har styrket sikkerheten gjennom sine policyer. Implementasjonen av dotenv for sikker lagring av miljøvariabler sørger for at sensitiv informasjon forblir beskyttet og uforstyrret av potensielle sikkerhetsbrudd.

Med fjorten implementerte endepunkter, tilbyr vår chatbot et rikt utvalg av funksjonaliteter som forbedrer brukeropplevelsen betydelig. Disse endepunktene garanterer at brukeren kan få artikkelforslag tilpasset sine interesser. Gjennom '/ask'-endepunktet kan brukerne ikke bare innhente informasjon relatert til spesifikke artikler, men også engasjere seg i substansielle dialoger med chatboten, drevet av OpenAIs GPT-modeller.

Dette backend-systemet har ikke bare oppfylt de tekniske kravene til prosjektet, men også lagt grunnlaget for fremtidige utvidelser og forbedringer, noe som åpner for ytterligere innovasjon og tilpasning i takt med teknologisk utvikling og brukernes skiftende behov.

4.6 Frontend

Introduksjon Målene for utviklingen av frontenden var å skape en chatbot som ikke bare forbedrer brukeropplevelsen, men også øker tiden leserne tilbringer på artikler. Dette innebar implementering av to primære funksjoner: en hovedchatbot på nettsidens forside som anbefaler artikler, og en sekundær chatbot på selve artikkelsidene for å engasjere og holde på lesernes oppmerksomhet.

For å realisere dette, utnyttet vi teknologier som JavaScript, HTML og CSS, (Programmering og språk 3.4.3), som sammen danner grunnlaget for chatbotens interaktive og visuelle funksjoner.

4.6.1 Brukergrensesnittdesign

For å sikre at chatboten blir brukt, er det essensielt at den fremstår både visuelt stimulerende og profesjonell. En godt designet chatbot kan øke brukerens tillit og engasjement, og dermed sannsynligheten for at teknologien tas i bruk. Dette har ledet til at vi har lagt ned en betydelig innsats i å perfeksjonere chatbotens visuelle aspekter.

Vår chatbot fremhever Sunnmørspostens ikoniske blåfarge, som ikke bare styrker merkevareidentiteten, men også bidrar til en visuell kohesjon med resten av nettstedet. Vi har valgt fonten Calibri for tekst, da dens rene og klare linjer forbedrer lesbarheten og gir en moderne estetikk. Videre er bruk av emojis integrert for å gjøre interaksjonene mer vennlige og engasjerende, noe som kan dempe eventuell frustrasjon under bruk.

Chatboten benytter seg av en rekke dynamiske elementer for å skape en levende og interaktiv opplevelse. Dette inkluderer interaktive knapper, responsive tekstfelt og subtile animasjoner som sammen bidrar til å gi brukeren en følelse av en sofistikert og teknologisk avansert tjeneste.

Figur 40: Chatbot design

Interaktive elementer Knappene som introduseres etter at chatboten er initialisert tilbyr fem valgmuligheter, designet med fokus på estetikk og brukerinteraksjon. Når musepekeren holdes over, endrer knappene farge til en mørkere tone som feedback, og ved klikk simuleres en inntrykningseffekt. Tekstfeltet utvider seg dynamisk etter antall tegn i meldingen, uten å skjule eksisterende samtaler. Under behandling av forespørsler blir knapper og tekstfelt midlertidig deaktivert, visuelt kommunisert gjennom en fargeendring og et 'forbudt' symbol på musepekeren. Detaljer om designvalg og deres betydning for brukeropplevelsen utforskes i 'Design 4.4.3.'

4.6.2 Dynamiske funksjoner

Tenkeanimasjon For å oppnå en konsistent og troverdig brukeropplevelse med vår chatbot, er det viktig å jevne ut responstidene, særlig fordi henting av informasjon kan variere betydelig i tid. Derfor har vi valgt å implementere en tenkeanimasjon som balanserer tidsforskjellene. Denne animasjonen består av tre prikker som animeres for å symbolisere at boten prosesserer informasjonen før den presenterer svaret. Dette gir inntrykk av at chatboten nøye vurderer sitt svar, og bidrar til å forsterke brukerens tillit til at de mottatte svarene er gjennomtenkte og pålitelige.

Figur 41: Tenkeanimasjon

Nedenfor er et utdrag av koden som illustrerer hvordan tenkeanimasjonen implementeres:

```
function showTypingAnimation() {
    const chatbox = document.querySelector(".chatbox");
    const typingLi = document.createElement('li');
    typingLi.id = 'typing-animation';
    const typingAnimationContainer = document.createElement('div');

    for (let i = 0; i < 3; i++) {
        const dot = document.createElement('div');
        dot.classList.add('typing-dot');
        typingAnimationContainer.appendChild(dot);
    }

    typingLi.appendChild(typingAnimationContainer);
    chatbox.appendChild(typingLi);
}
```

Denne funksjonen `'showTypingAnimation'` aktiveres før chatboten sender et svar. Den legger til en visuell animasjon i brukergrensesnittet som indikerer at boten tenker". Når botens respons er klar til å vises, fjernes animasjonen for å gjenopprette grensesnittet til normal bruk. Dette hjelper

med å standardisere ventetiden mellom brukerens spørsmål og botens svar, noe som gir en mer flytende og naturlig samtaleflyt.

Når chatboten har ferdigstilt sendingen av en melding, trigges funksjonen `'hideTypingAnimation'` for å fjerne skriveanimasjonen fra brukergrensesnittet. Nedenfor er koden som styrer dette:

```
    function hideTypingAnimation() {
        const typingLi = document.getElementById('typing-animation'
            );

        if (typingLi) {
            typingLi.remove();
        }
    }
```

Håndtering av meldinger Behandlingen av brukermeldinger styres av funksjonen generateResponse i filen MessageProcessing.js. Denne omfattende funksjonen, som strekker seg over 200 linjer, navigerer gjennom flere lag av kondisjonelle tester for å bestemme passende handlinger eller endepunkter basert på brukerens input. De første fem if-setningene er dedikert til de initielle knappene på brukergrensesnittet. Når en bruker klikker på en knapp som Kategorier", simuleres et skjult input "hvilke kategorier", som trigger funksjonen fetchAndDisplayCategories for å prosedere forespørselen. Et eksempel på kode fra generateResponse-funksjonen vises nedenfor:

```
const generateResponse = (userMessage) => {
  const userMessageLower = userMessage.toLowerCase();
  const chatbox = document.querySelector(".chatbox");

    if (userMessageLower.includes("hvilke kategorier")) {
        fetchAndDisplayCategories()
        }
    }
```

For meldinger som ikke matcher noen forhåndsdefinerte kriterier, videresendes de til '/ask-'endepunktet:

```
    else {
      let endpoint;
        fetch(`http://localhost:3000${endpoint}`, {
          method: endpoint === '/ask' ? 'POST' : 'GET',
          headers: {
            'Content-Type': 'application/json',
          },
          body: endpoint === '/ask' ? JSON.stringify({ message:
              userMessage }) : null,
        })
```

Respons fra /ask-endepunktet, som beskrives detaljert i backend-seksjonen av rapporten, kan returnere data i en av tre formater: 'text', 'article' eller 'confirm'. Når responsen er av typen 'text', håndteres det slik:

```
    if (item.type === 'text') {
    chatBubble = createChatLi(item.content, "incoming");
```

Her blir botens respons sendt til createChatLi-funksjonen, som oppretter et HTML -element for å representere chatmeldingen visuelt.

Når responsen er en 'article', prosesseres artikkeldata gjennom formatArticleMessage-funksjonen for å formatere presentasjonen før det sendes til 'createChatLi':

```
    else if (item.type === 'article') {
        console.log("Article data received:", item);
         const articleMessage = formatArticleMessage({
          title: item.title,
          author: item.author,
          summary: item.summary,
          url: item.url
        });
        chatBubble = createChatLi(articleMessage, "incoming");
```

Hvis responsen er av typen 'confirm', betyr dette at systemet har gjenkjent relevante 'tags' i brukerens melding og spør om brukeren ønsker å se relaterte artikler. Brukeren presenteres da med to valgmuligheter, en "Jaknapp og en Neiknapp.

Ved å trykke på "Ja"aktiveres funksjonen processArticles.

```
    yesButton.onclick = () => {
        processArticles(item.articles, 0, chatbox);
    };
```

Denne funksjonen er ansvarlig for å kalle formatArticleMessage for å formatere artiklene, og deretter presenterer den artiklene for brukeren én om gangen ved å bruke createChatLi. Mellom hver artikkel er det en forsinkelse på 1500 millisekunder for å sikre en jevn og naturlig visning. Alle artikler som vises, går systematisk gjennom denne prosessen:

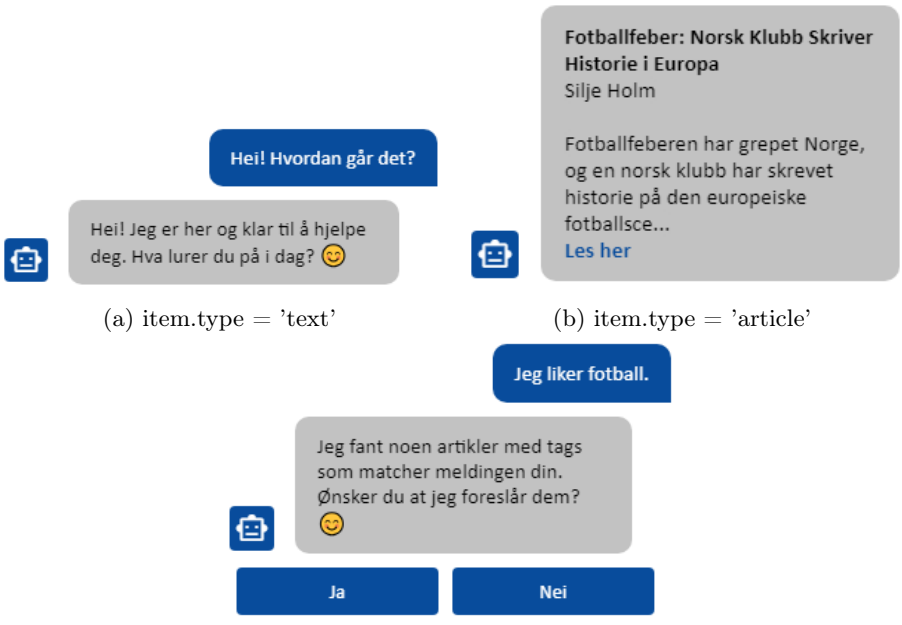
```
function processArticles(articles, index, chatbox) {
    if (index < articles.length) {
        showTypingAnimation();

        setTimeout(() => {
            hideTypingAnimation();
            const article = articles[index];
            console.log("Article at index", index, article);
            const formattedArticle = formatArticleMessage(article);
```

```
10            const articleLi = createChatLi(formattedArticle, "
                  incoming");
11            chatbox.appendChild(articleLi);
12
13            processArticles(articles, index + 1, chatbox);
14        }, 1500);
```

Hvis brukeren velger Nei", sendes meldingen til endepunktet '/askOpenAIForResponse' som direkte interagerer med GPT-modellen.

```
1     noButton.onclick = () => {
2         fetch('/askOpenAIForResponse', {
3             method: 'POST',
4             headers: {
5                 'Content-Type': 'application/json',
6             },
7             body: JSON.stringify({ message: userMessage }),
8             })
```

Nedenfor vises eksempler på hvordan ulike typer respons presenteres i chatbotens grensesnitt.



(a) item.type = 'text'

(b) item.type = 'article'

(c) item.type = 'confirm'

Figur 42: Responskategoriene i chatbotens brukergrensesnitt

Brukerinteraksjonsflyt Chatboten aktiveres når brukeren klikker på aktiveringsknappen, som typisk er plassert nederst til høyre på nettsiden. Aktiveringsprosessen starter med funksjonen initializeChatbot, som setter i gang flere underfunksjoner for å definere chatbotens oppførsel og brukergrensesnitt.

Funksjonen getArticleIdFromUrl benytter Node.js' innebygde funksjoner for å analysere nettstedets URL. Basert på om en artikkel-ID kan hentes fra URL-en, vil chatboten tilpasse sin oppførsel til enten artikkel-spesifikk eller generell frontside-interaksjon:

```
1      if (articleId) {
2        const articleButtonsContainer = createArticleButtons();
3        chatbox.appendChild(createChatLi(articleButtonsContainer, "
             incoming"));
4      } else {
5        createFaqButtons();
6      }
```

Avhengig av konteksten, tilpasses også velkomstmeldingene brukeren ser ved oppstart:

```
1  const greetingMessage = articleId
2    ? `Velkommen til artikkelen!
3    : "NB: Vær oppmerksom på at dette er en AI-chatbot. Noen
         detaljer kan være unøyaktige.";
```

```
1    const clickButtonMessage = articleId
2      ? "Gjerne spør meg hvis du har spørsmål om artikkelen, eller
           benytt knappene nedenfor."
3      : "Hei! Jeg er Sunnmørspostens Chatbot!";
```

Flere funksjoner sikrer en jevn brukeropplevelse:

- resizeTextarea justerer tekstfeltets størrelse basert på lengden på brukerens melding og tilbakestiller størrelsen når meldingen sendes.
- limitTextInput sørger for at brukerens inndata ikke overskrider 100 tegn, for å unngå overfylling av chatvinduet.
- handleChat administrerer sending og mottak av meldinger i chatgrensesnittet, inkludert å legge til brukermeldinger i chatvinduet og generere svar fra serveren.

```
        function handleChat() {
          let userMessage = chatInput.value.trim();
          if (!userMessage) return;

          chatbox.appendChild(createChatLi(userMessage, "outgoing")
              );
          scrollToBottomOfChat();
          generateResponse(userMessage);
          chatInput.value = '';
          resizeTextarea();
          counter.textContent = '0/100';
          warningMessage.classList.remove('visible');
          counter.classList.remove('active');
      }
```

I funksjonen 'handleChat' brukes også 'scrollToBottomOfChat', som igjen utløser 'smoothScrollToBottom'. Denne metoden sørger for at chatvinduet automatisk ruller ned til den nyeste meldingen gjennom en jevn animasjon over 200 millisekunder. Denne funksjonaliteten forbedrer brukeropplevelsen ved å sikre at samtalen føles sammenhengende og responsiv som bidrar til en følelse av høy kvalitet i chatboten.

Behandling av artikler i frontend I backend-delen av rapporten utforsket vi hvordan et endepunkt, i dette tilfellet, /relevantArticles, henter, sorterer og filtrere artikler fra databasen før den sender artiklene til frontend for behandling. Nå skal vi utdype i hvordan dette ser ut i frontend. Nedenfor er et utdrag av generateResponse funksjonen:

```
if (userMessageLower.includes("fetch relevant articles")) {
          const endpoint = '/relevantArticles';
          fetch(`http://localhost:3000${endpoint}`)
            .then(response => response.json())
            .then(data => {
              if (data.articles && data.articles.length > 0) {
                const introMessage = data.message;
                chatbox.appendChild(createChatLi(introMessage, "
                    incoming"));
                processArticles(data.articles, 0, chatbox);
              }
```

Først blir brukerens melding sjekket om den inneholder den skjulte inputen fra 'Relevante Artikler'-knappen fetch relevant articles", før den kaller på endepunktet /relevantArticles. Om den mottar artikler fra backend vil først meldingen som introduserer artiklerne sendes, før artiklene blir sendt til processArticles som prosesserer artiklene videre

Behandling av ChatGPT sammendrag i frontend Vi undersøkte også, i backend-delen av rapporten, prosessen bak /summarizeArticle-endepunktet, som håndterer forespørsler om å oppsummere en artikkel. Nå skal vi utforske hvordan sammendraget behandles i frontend etter at det er generert. Nedenfor er et utdrag fra runArticle funksjonen fra fila apiHandlers.js:

```
function runArticle(articleId) {
    fetch(`http://localhost:3000/summarizeArticle/${articleId}`)
      .then(response => response.json())
      .then(data => {
        if (data.summary) {
          const sentences = data.summary.split(/(?<=[.!?])\s+/);
          const formattedSummary = sentences.join('<br><br>');

          const summaryLi = createChatLi(formattedSummary, "
              incoming");
          const chatbox = document.querySelector(".chatbox");
          chatbox.appendChild(summaryLi);
        }
```

Denne funksjonen, runArticle, gjør følgende:

- Den henter sammendraget fra API-endepunktet: `/summarizeArticle/articleId`.
- Hvis et sammendrag eksisterer, splittes teksten i setninger for å forbedre lesbarheten.
- Disse setningene formateres med to linjeskift mellom hver setning for å skape visuell avstand, noe som bidrar til en mer leservennlig presentasjon.
- Et listeelement (li) opprettes ved å kalle createChatLi, som deretter legges til i chatboksen ved bruk av appendChild. På denne måten presenteres brukeren for et klart og tydelig sammendrag generert av ChatGPT.

4.6.3 Feilhåndtering

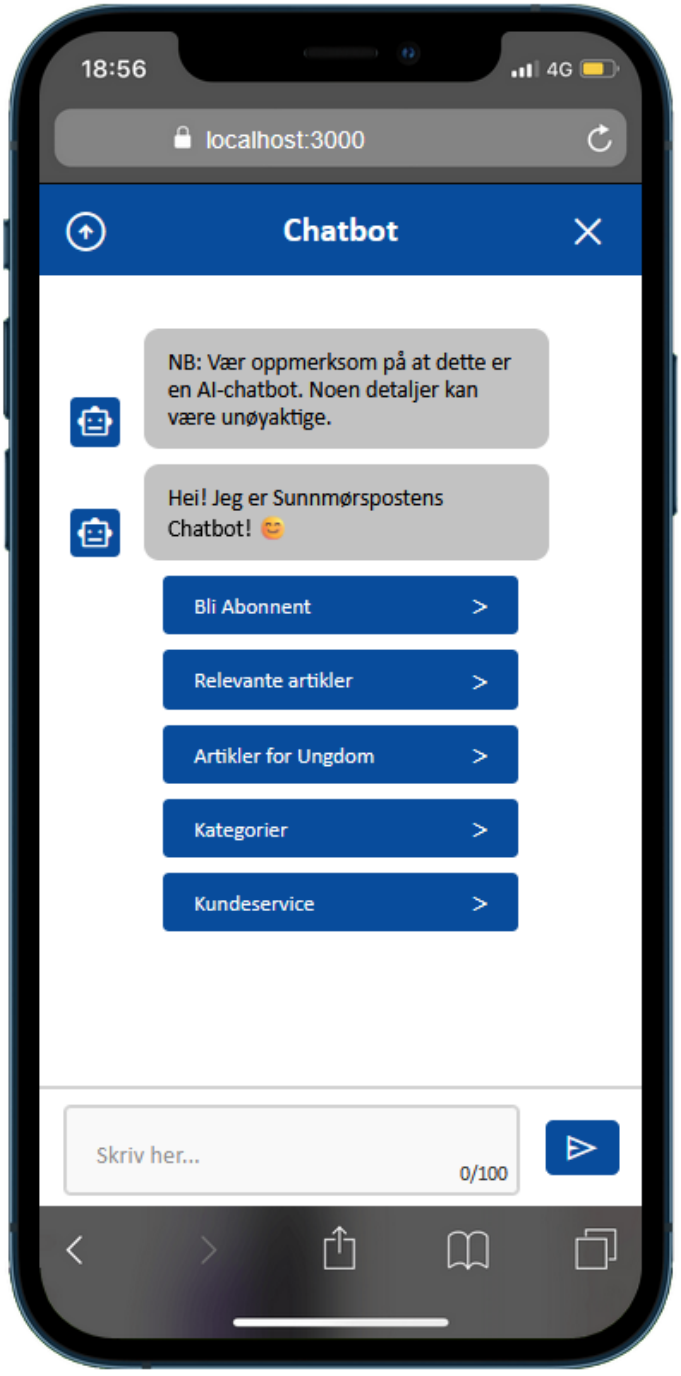
I tilfelle en feil under datainnhenting, vil systemet håndtere det slik som demonstrert i eksempelet fra funksjonen 'Lignende artikler':

```
        .catch(error => {
            console.error('Error fetching similar articles:', error);
            hideTypingAnimation();
            const errorMessage = "Beklager, det oppstod en feil ved
                henting av lignende artikler.";
            const errorLi = createChatLi(errorMessage, "incoming");
            chatbox.appendChild(errorLi);
            scrollToBottomOfChat();
        });
```

I dette eksemplet blir en feil logget i konsollen med en beskrivelse av hva som gikk galt. Funksjonen `‘hideTypingAnimation‘` blir kalt for å fjerne skrivemaskinanimeringen, noe som signaliserer til brukeren at prosessen er stoppet. Deretter genereres en brukervennlig feilmelding som vises i chatvinduet ved hjelp av funksjonen `‘createChatLi‘`, og innholdet skroller automatisk ned slik at feilmeldingen blir synlig. Dette sikrer at brukeren er informert om problemet og at chatvinduet oppdateres korrekt under feilsituasjoner.

4.6.4 Mobilkompatibilitet

Chatboten fungerer godt på mobil, mye på grunn av '@media (max-width: 768px)' reglene i CSS-filen som tilpasser elementenes oppførsel når skjermstørrelsen er 768 piksler eller mindre. Slik ser chatboten ut på en mobil enhet:



Figur 43: Chatbot på mobil

4.6.5 Frontend-struktur

Med mer enn 1300 linjer JavaScript-kode, var det essensielt å strukturere koden i klart definerte filer for å gjøre systemet lettere å navigere for oppdragsgiver. Denne filorganiseringen sikrer en intuitiv forståelse av hvor ulike funksjoner og komponenter er lokalisert:

- **apiHandlers.js** – Inneholder funksjoner for å håndtere API-forespørsler til backend, unntatt de som brukes av funksjonen generateResponse.
- **chatInteractions.js** – Håndterer opprettelsen av chatter i chattevinduet og definerer tenkeanimasjonen.
- **contentManagement.js** – Ansvarlig for administrasjon av knapper, behandling av artikler og oppdatering av artikkelinformasjon til chatboten.
- **csFaqManagement.js** – Administrerer kundeserviceinteraksjoner ved å håndtere spørsmål og svar i chatboten.
- **messageProcessing.js** – Dedikert fil for generateResponse-funksjonen som håndterer behandlingen av brukermeldinger.
- **utils.js** – Samler hjelpefunksjoner som 'smoothScrollToBottom' for jevn scrolling til bunnen av siden og 'formatArticleMessage' for å formatere artikkelmeldinger.
- **script.js** – Hovedskriptfilen som initialiserer og koordinerer chatbotens oppstart og funksjonalitet.

4.6.6 Oppsummering

Dette avsnittet gir en grundig gjennomgang av den teknologiske implementeringen av vår chatbots frontend. Vi utviklet to distinkte chatbot-versjoner: en for forsiden og en annen for artikkelsiden, hver med sine spesifikke funksjonaliteter. Bruken av JavaScript, HTML og CSS sikrer en dynamisk og visuelt behagelig brukeropplevelse.

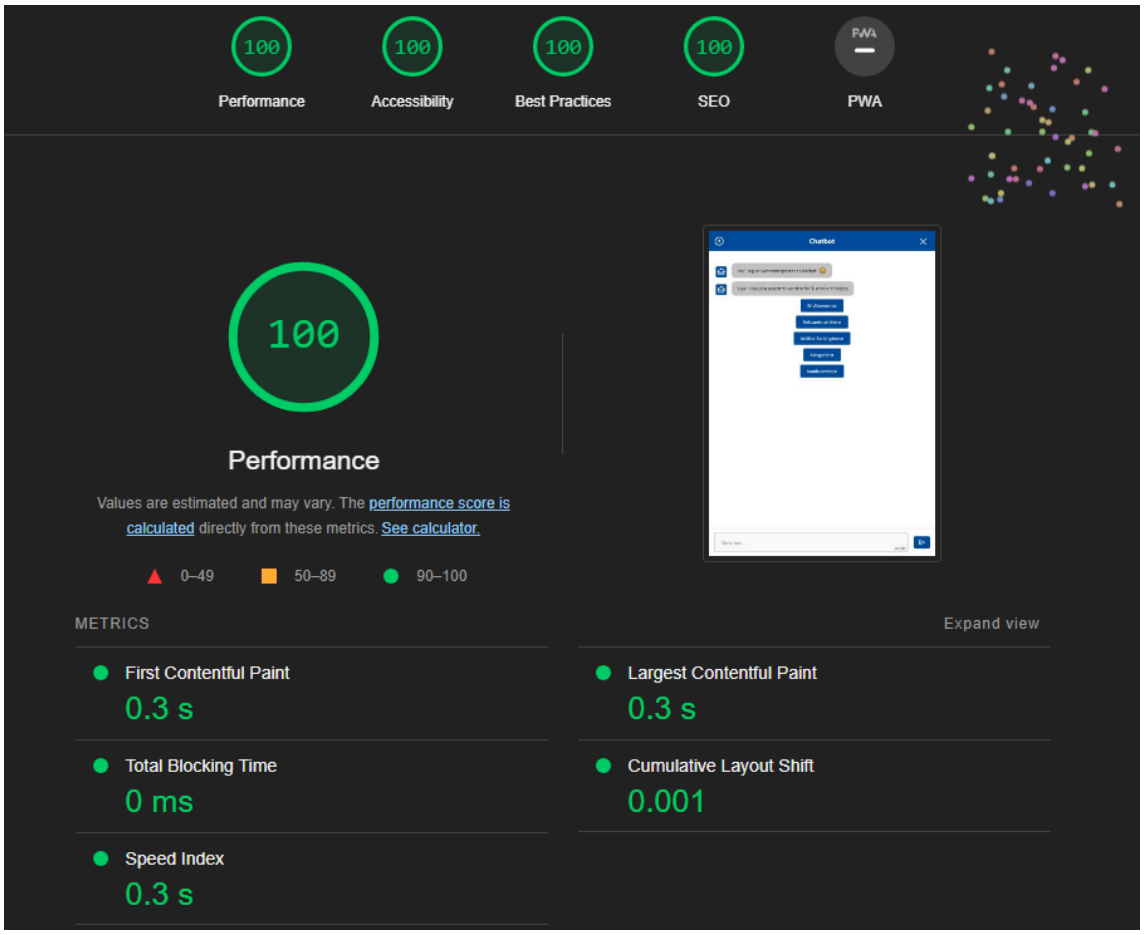
Design og brukervennlighet var sentrale aspekter fra begynnelsen. En høykvalitets chatbot er avgjørende for å engasjere brukere, derfor la vi betydelig tid og innsats i å utvikle funksjoner som dynamiske knapper, et responsivt tekstfelt og velbalanserte tenkeanimasjoner. Disse elementene er designet for å forsterke chatbotens pålitelighet og bygge tillit hos brukeren.

Frontendens struktur er nøye organisert i skriptfiler som kontrollerer ulike aspekter av chatbotens funksjonalitet, noe som bidrar til en ryddig og lett vedlikeholdbar kodebase. Dette gjør det enklere for oppdragsgiveren å forstå og videreutvikle chatboten om nødvendig.

Alt i alt gir dette kapittelet om frontend en omfattende oversikt over utviklingen av vår chatbot, og viser hvordan vi har prioritert både funksjonalitet og brukeropplevelse gjennom hele prosjektet.

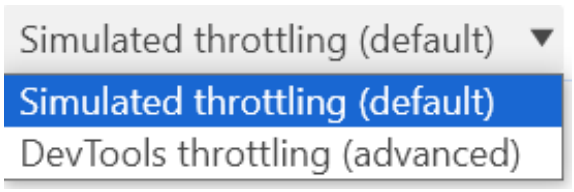
4.7 Lighthouse

Lighthouse-testen vår oppnådde en perfekt score på 100/100. Dette underbygger kvaliteten på vår applikasjon.



Figur 44: Lighthouse resultater

Testen ble utført under både 'Simulert båndbreddebegrensning' og 'DevTools båndbreddebegrensning'. Båndbreddebegrensningrefererer til bevisst reduksjon av datahastighet eller prosessorkraft for å simulere ulike brukssituasjoner og evaluere systemytelse under mindre ideelle forhold.



Figur 45: Lighthouse moduser

Følgende er en forklaring på hva de to modusene innebærer:

- **Simulert båndbreddebegrensning:** Dette alternativet bruker en simuleringsmodell for å anslå hvordan lasteytelsen ville endres under forskjellige nettverksforhold. Det begrenser ikke faktisk nettverket, men bruker prediktiv modellering for å simulere forholdene. Denne

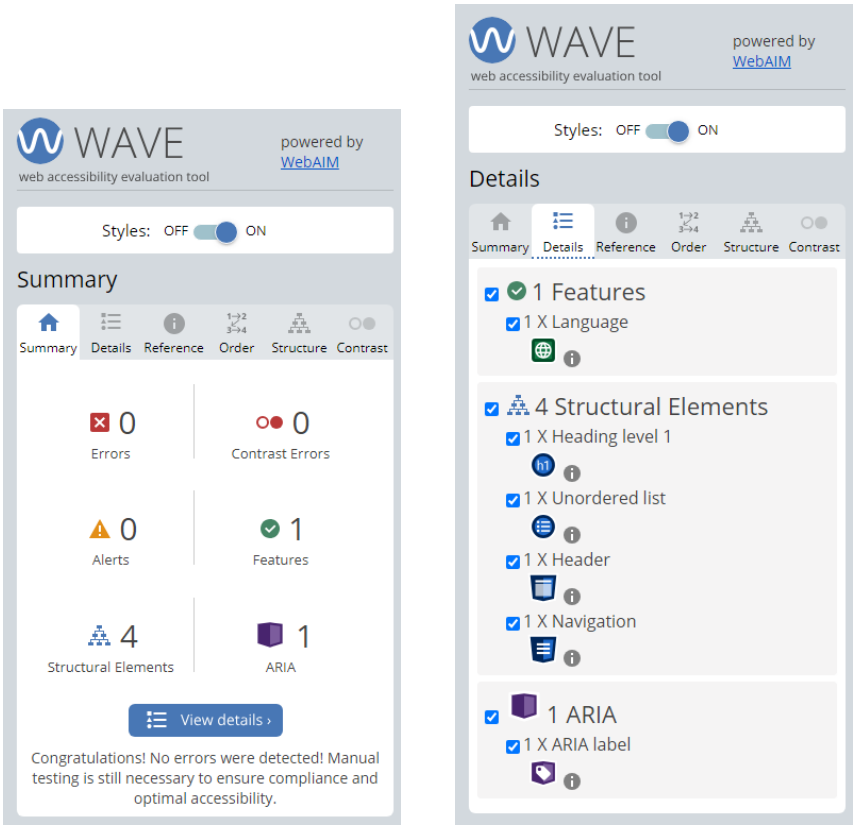
metoden er raskere da den ikke trenger å påføre båndbreddebegrensning i sanntid, og lar tester kjøre raskt.

- **DevTools båndbreddebegrensning:** Denne modusen anvender reell båndbreddebegrensning i nettleseren ved bruk av Chrome DevTools. Den begrenser aktivt nettverksbåndbredden og CPU for å etterligne spesifikke forhold. Denne metoden er generelt mer nøyaktig fordi den gjenspeiler hva som skjer i et virkelig begrenset miljø, men den kan være tregere og mer ressurskrevende ettersom testene kjører under faktisk båndbreddebegrenset forhold.

Vår chatbot oppnådde 100/100 på begge modusene. Under testene deaktiverte vi den intensjonelle forsinkelsen chatboten vanligvis bruker til å sende meldinger, ettersom Lighthouse ikke gjenkjente dette som en brukervennlig funksjon og trakk oss i 'performance'-poeng for det.

4.8 WAVE

WAVE er et evalueringsverktøy designet for å sikre at applikasjoner er tilgjengelige for alle brukere, spesielt de med ulike funksjonsnedsettelser. Som illustrert i bildene nedenfor, oppnådde vår applikasjon utmerkede resultater med ingen registrerte feil, ingen kontrastfeil, og ingen varsler. Detaljbildet viser flere funksjoner som forbedrer tilgjengeligheten, inkludert strukturerte elementer og en ARIA-etikett som er implementert i tekstfeltet i chatboten. Disse funksjonene bidrar til å sikre at alle brukere, uavhengig av deres individuelle behov, kan navigere og bruke applikasjonen effektivt.



(a) WAVE Oppsummering (b) WAVE Detaljer

Figur 46: WAVE Evaluering Resultater

4.9 Miljøfiler

I dette prosjektet har '.env'-filen konfigurert miljøvariabler for å muliggjøre en god kommunikasjon med eksterne tjenester og API-er. '.env'-filen fungerer som en nøkkel som inneholder sensitiv informasjon og konfigurasjonsparametere som er avgjørende for funksjonaliteten til prosjektet.

Én av de mest betydningsfulle oppføringene i '.env'-filen er OPENAI_API_KEY, som gir tilgang til OpenAI-plattformen og dens avanserte AI-tjenester. I tillegg inneholder '.env'-filen SUPABASE_URL og SUPABASE_ANON_KEY, som er tilknyttet Supabase, en plattform for utvikling av applikasjoner og databaseadministrasjon. Disse parametrene gir tilgang til Supabase-infrastrukturen og tillater operasjoner som lagring, henting og administrasjon av data gjennom bruk av Supabase API.

Det er viktig å merke seg at '.env'-filen inneholder sensitiv informasjon, inkludert API-nøkler og URL-adresser, som må behandles med forsiktighet for å forhindre uautorisert tilgang og potensielle sikkerhetsrisikoer. Derfor er det nødvendig å implementere sikkerhetsprotokoller og beste praksis for å sikre at disse nøklene ikke blir kompromittert. Gruppen har imidlertid ikke brukt sensitiv informasjon fra oppdragsgiver i prosjektet.

5 Drøfting

I dette kapitlet skal vi se nærmere på prosjektets fremgang og de beslutningene som har blitt tatt under prosessen. Vi vil undersøke utviklingen av prosjektet, resultatene vi har oppnådd, og hvilke utfordringer vi har møtt på veien.

5.1 Generelle resultat

Gjennom dette prosjektet har vi utviklet en fungerende chatbot for Sunnmørsposten, som integrerer moderne teknologier som OpenAI's GPT-modeller. Chatboten er designet for å forstå og respondere på brukerforespørsler på en måte som etterligner menneskelig kommunikasjon. Den genererer relevante svar og anbefalinger med knapper og basert på brukerens tekstprompt. Den er utviklet til å kunne integreres med eksisterende nyhetsartikler og teknologi til oppdragsgiver, tilbyr funksjonalitet for å forklare artikler, oppsummerer innhold og foreslå relaterte artikler.

Prosjektet hadde flere overordnede mål:

1. **Forlenge lesetiden på artikler:** Ved å implementere en chatbot som kan engasjere leserne gjennom dialoger om artiklene, har vi skapt en funksjonalitet som vi mener bidrar til å holde leserne lengre på siden. Dette trenger testing og analyse verktøy til å fastslå påstanden.
2. **Forbedre brukeropplevelsen:** Chatboten er utviklet til å forstå og respondere på brukerforespørsler på en måte som etterligner menneskelig kommunikasjon. Den genererer relevante svar og anbefalinger basert på brukerens prompt, noe som forbedrer interaktiviteten og tilgjengeligheten av innholdet til nyhetsplattformen.
3. **Tilby relevante artikkelanbefalinger:** Chatboten kan foreslå relevante artikler basert på navigering med funksjonalitetsknapper, eller ved å analysere nøkkelord i brukerens melding og deretter foreslå artikler basert på hvilken emneknagg de tilhører i databasen.
4. **Kan forklare bakgrunnen for en sak basert på tidligere artikler:** Chatboten er i stand til å sammenfatte og forklare konteksten av nyhetssaker ved å referere til, og oppsummere tidligere artikler i samme 'story'. Dette gir bedre forståelse og overblikk over situasjonen.
5. **Foreslå relevante spørsmål og tilby lenker for ytterligere informasjon:** Chatboten kan svare på relevante spørsmål og foreslå lenker til mer informasjon.
6. **Implementere chatboten på smp.no eller som en nettleserutvidelse:** Selv om dette målet ikke er oppnådd enda, er vi godt forberedt på å realisere det i nær fremtid. Implementeringen vil kreve tilpasning til Sunnmørspostens systemer, noe som dessverre ble nedprioritert til fordel for å ferdigstille produktet.
7. **Sikre at chatboten støtter universal design:** Vi har sørget for at chatboten skal være estetisk og støtter universell utforming. Imidlertid trengs det testing med en mer variert brukergruppe til å vurdere om den er tilgjengelig for brukere med nedsatt funksjonsevne.
8. **Bevise 'proof of concept' til en KI-dreven chatbot på nyhetsnettside:** Prosjektet har demonstrert engasjement og effektiviteten til en KI-drevet chatbot på en nyhetsplattform gjennom brukertester og dialog med testerne.

Prosjektet har demonstrert hvordan avansert teknologi kan brukes til å forbedre brukeropplevelsen på digitale nyhetsplattformer ved å tilby dynamiske og avanserte brukerinteraksjoner. Til tross for at det ikke har vært mulig å gjennomføre fullskala testing med virkelige brukere, har vi gjennomført flere brukertester under utviklingen. Resultatene indikerer en forbedring i brukerengasjement og en positiv mottakelse av chatbotens funksjonaliteter av brukere.

Læringsmålene for prosjektet inkluderte å få erfaring med utvikling av KI-drevne løsninger, anvendelse av naturlig språkprosessering (NLP) og webdesign. Prosessmålene fokuserte på å bruke SCRUM-metodikken for prosjektledelse og sikre en iterativ utviklingsprosess med kontinuerlig tilbakemelding fra brukere og oppdragsgiver.

5.2 Forventninger og resultat

Basert på de forventningene vi etablerte sammen med veilederen og oppdragsgiver, er vi fornøyde med å ha implementert alle hovedfunksjonene og oppnådd et innbydende design. Hovedmålet for Sunnmørsposten var å forlenge lesetiden på artikkelsidene og forbedre brukeropplevelsen på nettstedet. Ved å integrere ChatGPT, kan leserne engasjere seg i dialoger om artiklene og enkelt få tilgang til relevant kontekst. Brukere får også anbefalinger om artikler basert på nøkkelord og kan navigere gjennom nettstedet med assistanse fra chatboten. Tilbakemeldingene fra den første brukertesten indikerer at slike funksjoner vil forbedre brukeropplevelsen betydelig.

5.2.1 Oppdragsgivers systemer

Som nevnt tidligere har vi ikke hatt direkte eller indirekte tilgang til oppdragsgivers systemer, noe som har vært en betydelig utfordring i utviklingen av applikasjonen. Vi har imidlertid mottatt nok informasjon til å kunne gjøre informerte antagelser om systemets oppbygning, noe som har muliggjort videre utvikling. Vi har kunnet observere systemets funksjoner fra utsiden og mottatt begrenset informasjon om det.

Vi ville satt stor pris på om oppdragsgiver tidlig i prosjektet hadde informert oss om at systemene ville være utilgjengelige, slik at vi ikke kunne forvente konkret og direkte tilgang til deres databaser. Oppgavebeskrivelsen fastslår klart at det skal legges til rette for at studentene får tilgang til alle nødvendige systemer for å løse oppgaven. Mangelen på forhåndsinformasjon krevde en betydelig tilpasning fra vår side, spesielt da vi lenge hadde forventet å motta mer detaljert informasjon.

5.2.2 Møter

I løpet av prosjektet arrangerte vi møter med oppdragsgiver ved milepæler eller når vi hadde spesifikke elementer å demonstrere eller diskutere. Denne tilnærmingen gjorde det mulig for oss å motta direkte tilbakemeldinger fra oppdragsgiver på kritiske tidspunkter i prosjektet. I ettertid ser vi at en fast møtestruktur, for eksempel annenhver uke, ville ha vært mer gunstig for vår gruppe. En slik fast møterytme ville ha forbedret kommunikasjonen og forenklet samarbeidet ved å skape en mer forutsigbar og kontinuerlig dialog. Dette ville ha sikret løpende innsikt og veiledning fra oppdragsgiver. Hvis det ikke var fremskritt å presentere til et planlagt møte, kunne vi ha valgt å avlyse det. Ved å implementere en slik fast møtestruktur ville vi ha kombinert behovet for å demonstrere konkrete fremskritt med fordelen av regelmessig og strukturert kommunikasjon.

5.2.3 Opprinnelig plan

I prosjektets opprinnelige fase hadde vi planlagt å anvende kunstig intelligens på en annen måte. Målet var å trene en GPT-modell ved hjelp av OpenAIs 'fine-tuning'-program til å besitte omfattende kunnskap om Sunnmørspostens tjenester, artikler og funksjonaliteter, og lære chatboten hvordan den skulle håndtere forespørsler og situasjoner.

Primært var tanken at brukere skulle kunne innhente all informasjon om Sunnmørsposten direkte fra vår fin justerte GPT-modell via tekstfeltet. Modellen skulle trenes med artikler fra oppdragsgivers database. Etter ytterligere forskning på chatboters funksjonalitet og detaljer rundt OpenAI's API, besluttet vi å avstå fra å trene vår egen modell, da dette var unødvendig for å oppnå den ønskede funksjonaliteten og også medførte potensielle etiske og sikkerhetsmessige bekymringer.

Som et resultat valgte vi i stedet å benytte OpenAI's NLP-teknologi (Natural-Language-Processing 2.2.2) for å utføre avanserte funksjoner, og dermed beholde kunstig intelligens som en kjernekomponent i prosjektet. Utfordrende funksjoner som tekstoppsummering og forståelse av brukerinnsendt tekst, som krever avansert NLP, viser seg å være teknisk krevende, men med dagens teknologi, som er brukervennlig og tilgjengelig, kan vi utføre komplekse oppgaver med relativt enkel innsats.

5.2.4 Chattefunksjoner

Vi kan forbedre våre chattefunksjoner for å ytterligere forbedre brukeropplevelsen og effektivisere håndteringen av forespørsler. Et sentralt tiltak ville være å implementere avanserte NLP-teknikker for å bedre fange opp og forstå brukernes hensikter. For øyeblikket, når chatboten møter på skrivefeil eller uklarheter, sender den oftest tekstprompten til ChatGPT. Selv om denne kan håndtere samtaler, kan den ikke foreslå relevante artikler direkte. Derfor veileder den i stedet brukeren ved å foreslå at de bruker navigeringsknappene eller omformulerer spørsmålet sitt med stikkord. For å gjøre dette mer målrettet, kunne en løsning være å utvikle en feiltolerant algoritme som identifiserer og korrigerer vanlige skrivefeil eller misforståelser før prompten sendes til ChatGPT. Dette vil tillate en mer nøyaktig og relevant artikkelforespørsel basert på brukerens faktiske intensjoner.

Fremover ønsker vi også å implementere en funksjon som muliggjør direkte søk og referanse til artikler rett fra chatvinduet, uavhengig av om brukeren befinner seg på den relevante artikkelsiden. Ved å utvide chatbotens evne til å håndtere forespørsler uavhengig av brukerens nåværende side, kan vi potensielt forbedre både tilgjengeligheten og anvendeligheten av chatboten. Denne innsikten ble styrket under brukertest 2 (Brukertest: 2, 3.10.2), hvor en deltaker forsøkte å innhente informasjon om artikler og bakgrunn umiddelbart etter å ha aktivert chatboten.

5.2.5 Knappbaserte funksjoner

Vår nåværende implementering av visuelle og knappebaserte grensesnitt har vist seg å være svært effektiv. Inspirert av de tradisjonelle, knappebaserte chatbotene som allerede er i bruk på mange norske nettsider, har vi, gjennom oppgavebeskrivelsen og samtaler med oppdragsgiver, utviklet en løsning som integrerer denne tilnærmingen. For fremtidig arbeid planlegger vi å fokusere på å optimalisere og utvide disse løsningene for å tilby enda mer intuitive og brukervennlige veiledninger. Vi sikter mot at alle funksjoner som er tilgjengelige gjennom knappene også skal være tilgjengelige gjennom chat, for å skape en sømløs og brukervennlig opplevelse for alle.

5.3 Proof of Concept

Selv om chatboten utviklet for Sunnmørsposten kanskje ikke settes direkte i full produksjon, utgjør dette prosjektet et vesentlig 'proof of concept' som bekrefter både teknologisk gjennomførbarhet og konseptuell gyldighet. Ved å utvikle og teste en AI-drevet chatbot har vi demonstrert hvordan avanserte teknologier som kunstig intelligens kan benyttes for å forbedre brukeropplevelsen av nyhetsformidling. Prosjektet har anerkjent at selv om vi nærmer oss et ferdig produkt, kreves det ytterligere optimaliseringer, mer testing og integrasjon med Sunnmørspostens eksisterende systemer før en fullstendig lasering kan realiseres. Disse trinnene anser vi som nødvendige etter prosjektets avslutning. Prosjektet og utviklingsprosessen har gitt oss en bedre forståelse av AI's muligheter i medieindustrien og åpner opp for potensiell videre forskning og utvikling av tilsvarende teknologier.

5.4 SCRUM og Jira

Valget om å benytte SCRUM som arbeidsmetodikk ble anbefalt av vår veileder, og med gratis tilgang var vi ivrige etter å ta i bruk både SCRUM og Jira. Jira bidro til en utmerket oversikt tidlig i prosjektet og hjalp oss med å fordele arbeidsoppgaver effektivt. Det innebar imidlertid en betydelig læringskurve for å forstå dybden av SCRUM og den beste måten å anvende metodikken på. Tilpasningen til et nytt verktøy som Jira var tidskrevende, og til tider opplevdes det som et unødvendig tillegg til arbeidsbyrden. Vi tror at en tidligere introduksjon til dette programmet i vårt studieløp kunne ha gjort oss bedre rustet til å utnytte dette verktøyet effektivt til bacheloroppgaven.

Implementeringen av SCRUM-metodikken ble mer utfordrende da vi, som nevnt i metodeseksjonen 3.1.2, delte gruppen, noe som førte til at vi ble redusert til to personer. Til tross for denne endringen var vi motiverte for å fortsette med det etablerte systemet. Det falt naturlig at den ene brukte mesteparten av tiden til utvikling, mens den andre tok på seg rollen som produktansvarlig, i tråd med SCRUM-rollene.

Samlet sett er vi fornøyde med å ha lært om og anvendt SCRUM, ettersom det viste seg å være en effektiv arbeidsmetodikk for små utviklingsgrupper. Vi er overbevist om at hvis vi skulle starte et nytt prosjekt, ville vi dra stor nytte av å fortsatt bruke SCRUM og verktøy som Jira og Confluence.

5.5 Etiske aspekter

Ved integrasjonen av kunstig intelligens i en chatbot for en nyhetsside oppstår flere etiske spørsmål som krever nøye vurdering. Det er avgjørende at nyhetsinnholdet er nøyaktig og pålitelig. Selv om ChatGPT kan generere svar fra en omfattende kunnskapsbase, kan denne informasjonen noen ganger være utdatert, unøyaktig eller ufullstendig. Derfor må det etableres regelmessig overvåkning for å sikre at informasjonen som formidles er korrekt og unngår rask spredning av feilinformasjon, som kan ha alvorlige konsekvenser.

Videre kan algoritmer innebære skjevheter, vanligvis stammende fra deres treningsdata. For en nyhetsside er det essensielt at chatboten behandler emner og nyheter balansert og uten fordommer mot bestemte grupper, meninger eller temaer. Selv om ChatGPT er designet for å minimere partiskhet, er dette ikke en absolutt løsning. Klare prosedyrer for ansvarlighet må være på plass for å håndtere feil når de oppstår, og det må være enkelt for brukere å rapportere problemer.

Det er avgjørende å klarlegge ansvar ved feil, enten dette ligger hos nyhetssiden, AI-utviklerne eller begge parter. Transparens rundt hvordan chatboten trekker sine konklusjoner er essensielt for å bygge brukertillit og styrke informasjonens troverdighet. En integral del av etisk AI-utvikling er evnen til å forklare systemets avgjørelser på en måte som er forståelig for brukerne. Ved oppstart vil vår chatbot eksplisitt informere brukerne om at den opererer med kunstig intelligens og inkludere en notis, "NB: Oppsummeringen er generert av ChatGPT," når den lager oppsummeringer eller forklarer bakgrunnen til en sak. Når produktet settes i drift, vil vi implementere omfattende tiltak for å sikre etisk bruk og effektivt forhindre spredning av feilinformasjon gjennom nøye overvåkning og ansvarlig drift, noe som er av største betydning for et nyhetsnettsted.

5.6 Rammeverk

Valget mellom å bruke ren JavaScript og et JavaScript-rammeverk avhenger av flere faktorer som prosjektets størrelse, chatbotens kompleksitet og teamets ferdigheter. Selv om vårt team var kjent med flere rammeverk, sto vi overfor et vanskelig valg. Til tross for teamets erfaring med både React og Angular, benyttet oppdragsgivers systemer primært Vue. Gitt den betydelige innsatsen det krever å sette seg inn i et nytt rammeverk, søkte vi råd fra ChatGPT. Den foreslo:

"I mange tilfeller er det effektivt å starte med ren JavaScript for å utvikle en prototype, og så vurdere å overgå til et rammeverk etter hvert som kravene i prosjektet øker. Dette tilbyr en god balanse mellom kontroll, effektivitet og fleksibilitet."

På bakgrunn av dette rådet startet vi med en enkel prototype i ren JavaScript. Vi bestemte oss for at hvis det viste seg at arbeidet med ren JavaScript ble mer omfattende enn å bytte til et rammeverk, ville vi revurdere. Vår chatbot, selv om den kjører på ren JavaScript, er designet for enkel integrasjon med oppdragsgivers systemer og antas å kreve minimale tilpasninger.

5.7 Database

Oppdragsgiveren var ikke villig til å gi oss tilgang til databasen de har i produksjon. Derfor opprettet vi vår egen 'dummy'-database med tjue artikler for å simulere en ekte artikkeldatabase. Gjennom møter med Sunnmørsposten fikk vi innsikt i hvilke databasestrukturer som var essensielle. Vi gjenskapte disse kolonnene så nøyaktig som mulig i vår egen database. Selv om vi ikke hadde eksakt innsyn i deres faktiske databasestruktur, var målet å sikre at produktet vårt kunne integreres sømløst i deres systemer ved overlevering.

For vår prosjektprototyp valgte vi å ikke opprette en omfattende database med tusenvis av artikler, noe som ville være typisk for en ekte nyhetsside. I stedet genererte vi tjue fiktive artikler ved hjelp av ChatGPT, fordelt jevnt over de kategoriene vi hadde definert. Dette ga oss en funksjonell, men forenklet database som var tilstrekkelig for å teste funksjonaliteten til vår chatbot, ettersom hovedfokuset var på chatbotens utvikling.

Vår 'dummy'-database inneholder kolonnene 'id', 'author', 'content', 'tags', 'category', 'publication_date', 'viktighetsgrad', 'url' og 'series'. Disse ble ansett som tilstrekkelige for å effektivt sortere og presentere artikler. Under et nylig møte med Sunnmørsposten ble det diskutert at vi ikke hadde inkludert kolonner for 'bilder' eller 'bildetekster'. Vi vurderte at dette ikke var nødvendig for chatbotens primære funksjoner, som er å sortere og anbefale artikler basert på innhold, kategorier,

tagger, publiseringsdato og viktighetsgrad. Derfor valgte vi å utelate disse fra vår database. Likevel betyr dette ikke at chatboten ikke kan håndtere artikler med bilder; den vil bare ikke bruke bildene eller bildetekster som kriterier for å anbefale artikler til brukeren.

5.8 Funksjonalitet

Sunnmørsposten ga oss stor frihet til å tolke og definere chatbotens funksjonalitet. De var fleksible med hensyn til ønskede funksjoner, og punktene i oppgavebeskrivelsen var åpne og greie å arbeide med. I møter med Sunnmørsposten diskuterte vi detaljer rundt disse punktene, men de ga oss hovedsakelig ansvar for å selv bestemme hvilke funksjoner som ville være mest effektive for en chatbot basert på nyhetsartikler.

5.8.1 Forsidebot

Vi startet med å plassere en 'Bli abonnent'-knapp som den første interaktive funksjonen i chatboten, siden tilgang til mye av innholdet på Sunnmørsposten krever et abonnement, og det å tiltrekke nye abonnenter er en prioritet.

Den neste knappen er 'Relevante kategorier'. Denne funksjonen er strategisk plassert for å umiddelbart tilby brukeren de nyeste og mest relevante artiklene ved et enkelt trykk, noe som er ideelt for brukere som ønsker rask tilgang til lesemateriale.

Videre har vi 'Artikler for ungdom'-knappen. Denne funksjonen er spesielt designet for å øke engasjementet blant unge lesere, en gruppe som tradisjonelt sett engasjerer seg lite med aviser.

'Kategorier'-knappen er plassert nærmere bunnen. Å navigere gjennom denne krever litt mer tid og tålmodighet; brukeren må først velge en kategori og deretter en underkategori som 'Nyeste', 'Viktigste', eller 'Tilfeldig'.

Den siste knappen er 'Kundeservice', som finnes helt nederst. Denne skiller seg fra de andre ved at den primært håndterer kundeservicehenvendelser.

I tekstfeltet der brukere kan skrive meldinger til chatboten, har vi implementert en funksjon som foreslår artikler basert på brukerens input. Chatboten analyserer meldingene for nøkkelord og spør deretter brukeren om den ønsker forslag relatert til disse nøkkelordene, noe som effektivt personaliserer interaksjonen.

5.8.2 Artikkelbot

Funksjonen for å oppsummere artikler har vist seg å være svært vellykket. Til tross for vår usikkerhet omkring implementasjonen, overrasket det positive resultatet oss. Vi har sørget for å gi chatboten en konsistent og effektiv prompt som styrer denne funksjonen.

'Bakgrunn kort forklart' var et annet område vi var usikre på. Sunnmørsposten ønsket at chatboten skulle kunne gi brukerne kontekst rundt artikler. Vi løste dette ved å la chatboten tilby oppsummeringer av artiklene innenfor samme 'story'.

For å adressere oppgavebeskrivelsens krav om at chatboten skal tilby lenker til ytterligere informa-

sjon, fant vi ut at den mest effektive metoden var å lenke til andre artikler innen samme 'story'. Det vil være lenkene til artiklene som blir oppsummert i 'bakgrunn kort forklart'.

Vi inkluderte også en funksjon for å foreslå lignende artikler. Dette er enkel å implementere, og har stor verdi ved å forlenge brukerens engasjement på nettstedet. Hvis en leser finner en artikkel interessant og trykker på 'lignende artikler'-knappen, vil denne funksjonen anbefale andre artikler med lignende tema.

I chatten kan brukeren stille spørsmål direkte til chatboten om artikkelinnholdet, og chatboten kan føre en samtale basert på innholdet i artikkelen. Imidlertid har chatboten for øyeblikket ikke kapasitet til å oppgi informasjon om andre artikler uten å være inne på selve artikkelen. Dette er noe vi ser på som en mulig forbedring for fremtidige oppdateringer.

5.9 OpenAI og kunstig intelligens

Måten vi bruker OpenAI's modeller på, er at vi gir dem en prompt ved oppstart som bestemmer hvordan de skal oppføre seg. Opprinnelig planla vi å lage vår egen modell ved hjelp av OpenAI's 'fine-tuning' program, som beskrevet i 'Opprinnelig plan 5.2.3.' Denne planen ble imidlertid skrinlagt da de eksisterende modellene viste seg å være mer enn gode nok for våre behov. Som beskrevet i 'Dynamisk innholdshåndtering 4.5.7' benytter vi modellen 'GPT-turbo-3.5'. Dette er den raskeste modellen som oppfyller våre krav i tilstrekkelig grad.

For å bruke OpenAI's modeller måtte vi abonnere på deres tjenester. Dette abonnementet gir tilgang til de fleste OpenAI-modellene samt en API-nøkkel som kan brukes til å kalle en modell gjennom IDE-en vår.

Som forklart i 'Funksjonalitet 4.3.6', har kunstig intelligens kun ansvar for tre funksjoner i vårt prosjekt. De fleste funksjonene innebærer å hente artikler fra databasen ved hjelp av GET-forespørsler med Supabase-klienten. Kunstig intelligens brukes kun i de to oppsummeringsfunksjonene til artikkelboten, og for å engasjere seg i hjelpsomme samtaler med brukerne.

5.10 Tester

I denne seksjonen vil vi drøfte funnene fra brukertestene 3.10.

5.10.1 Brukertest 1

Brukertest 1 ble gjennomført ved å diskutere tre forskjellige konsepter. Målet var å forstå hvordan typiske lesere oppfatter ideene for prosjektet, samt å identifisere hva brukerne anser som viktigst. Vi forventet også at testen ville gi oss verdifulle innblikk før oppstarten av utviklingen. Testen ga oss innsikt i brukerbehov og preferanser, og skapte en god basis for videre utvikling.

Brukerne viste stor interesse for bruken av KI og ChatGPT for å forbedre brukeropplevelsen. Funksjoner som oppsummering av artikler og forklaring av kontekster ved hjelp av chatboten ble ansett som svært nyttige. En av brukerne foreslo å bruke KI til å personalisere nyhetsinnhold, noe som også kunne integreres i chatboten. Sunnmørsposten har vist interesse for skreddersydd funksjonalitet basert på brukernes behov. Vi mener at dette er en nyttig funksjon som kan utvikles

i fremtiden, men på grunn av tid og ressurser velger vi å holde det utenfor prosjektets nåværende omfang.

Generelt mente brukerne at en godt utformet chatbot ville være et nyttig verktøy som de ville bruke, både av nysgjerrighet og for praktisk hjelp. De foretrakk en chatbot integrert i artiklene eller en tradisjonell chatbot som er konstant tilgjengelig.

Brukertest 1 ga flere viktige innsikter som kan forme utviklingen, hovedinntrykkene var:

1. **Brukerinteresse:** Brukerne viste stor interesse for en chatbot på en nyhetsnettside, spesielt hvis den kunne oppsummere artikler og forklare kontekster. Dette indikerer at en chatbot kan forbedre brukeropplevelsen ved å tilby mer dybde og klarhet i nyhetsinnholdet.
2. **Designpreferanser:** Brukerne mente en integrert chatbot som ikke var påtrengende på hovedsiden, men som kunne aktiveres inne i artikler kunne være gunstig.
3. **Bred anvendelse:** Brukerne trodde at både yngre og eldre lesere ville ha nytte av chatbotten, noe som understreker potensialet for bred anvendelse og aksept blant forskjellige demografiske grupper.
4. **Nysgjerrighet og funksjonalitet:** Brukerne brukte chatbotten både av nysgjerrighet og for praktiske formål. Dette antyder at chatbotten bør utformes med et bredt spekter av funksjoner som kan appellere til ulike behov og interesser.
5. **Brukerengasjement:** Brukerne foreslo at en personalisert anbefalingsalgoritme ville være verdifull, noe som er i tråd med Sunnmørspostens interesser.

Brukertest 1 har vist at det er en klar interesse for og behov for en KI-basert chatbot på en nyhetsnettside. Videre utvikling bør fokusere på å tilby nyttige funksjoner som oppsummering av artikler og kontekstforklaringer for å forbedre brukeropplevelsen og øke engasjementet. Selv om personalisering er en lovende funksjon, vil vi ikke prioritere dette i vårt nåværende prosjekt på grunn av begrensede ressurser.

Formålet med brukertest 1 var å samle inn brukernes tanker om de foreslåtte ideene og få en dypere forståelse av hva faktiske brukere ønsker i en chatbot. Etter å ha analysert tilbakemeldingene fra brukerne, presenterte vi resultatene for vår oppdragsgiver, Sunnmørsposten. Basert på brukernes innsikter og preferanser, hadde vi en konstruktiv samtale med oppdragsgiver om hvordan vi skulle fortsette prosjektet. Samarbeidet hjalp oss med å justere våre design- og funksjonalitetsvalg, slik at utviklingen av chatboten kunne fortsette i tråd med både brukernes ønsker og oppdragsgivers mål.

5.10.2 Brukertest 2

Brukertest 2 ble utført på en lignende måte som brukertest 1, men med mer frihet for brukerne til å utforske chatboten på eget initiativ. Formålet var å identifisere hvilke funksjoner en typisk bruker ville finne nyttige og hvilke utfordringer de kunne møte. Testen inkluderte et sett oppgaver, og merkbare hendelser ble dokumentert.

Oppgave 1: Brukerne ble bedt om å stille chatboten et spørsmål. De spurte om ulike emner, inkludert tilfeldige spørsmål, noe som indikerte behovet for at chatboten skal kunne håndtere et bredt spekter av spørsmål effektivt.

Oppgave 2: Brukerne skulle finne en hvilken som helst artikkel. Denne oppgaven ble løst, men den avdekket behovet for en mer realistisk nyhetsdatabase og forbedrede søkefunksjoner basert på relevans og nyhetsverdi.

Oppgave 3: Brukerne skulle navigere til en artikkel og deretter be om en oppsummering. Flere problemer oppsto, blant annet at chatboten ga for lange oppsummeringer og hadde vanskeligheter med å håndtere forespørsler om kortere oppsummeringer. Brukerne ønsket også å kunne be om oppsummeringer uten å måtte være inne på artikkelsiden. Dette er mulig og kan legges til i fremtidig arbeid, selv om det ikke er prioritert i hovedutviklingsfasen. Brukerne ga positiv tilbakemelding på chatbotens visuelle design og knappene. Det var delte meninger om hvorvidt de foretrakk å bruke chatten eller knappene først. Det understreker viktigheten av å optimalisere begge metodene for interaksjon.

Brukertest 2 avdekket flere viktige områder som kan forbedre chatbotens funksjonalitet og brukeropplevelse:

1. **Forbedret database og søkefunksjoner:** En mer realistisk nyhetsdatabase og forbedrede søkefunksjoner basert på relevans og nyhetsverdi vil gjøre det lettere for brukerne å finne de mest aktuelle artiklene.
2. **Håndtering av tilfeldige spørsmål:** Chatbotten må kunne håndtere et bredt spekter av spørsmål effektivt for å møte brukernes behov.
3. **Videreutvikling av tags-funksjonalitet:** Tags-funksjonen er nyttig for brukerne og bør videreutvikles for bedre å kunne sortere og finne relevante artikler.
4. **Forbedret samtalehistorikk:** For å ha mer naturlige samtaler med chatbotten, bør samtalehistorikken optimaliseres slik at brukeren kan veilede samtalen mer effektivt.
5. **Rettelse av bugs:** De identifiserte bugsene, inkludert språkbytte midt i samtalen og feil artikkelvisning, må rettes for å sikre en jevn og pålitelig brukeropplevelse.
6. **Optimalisering av gamle funksjoner:** Eldre funksjoner må oppdateres for å være på nivå med de nyere funksjonene, slik at brukeropplevelsen blir konsistent.
7. **Balansere chat og knapp-funksjonalitet:** Begge interaksjonsmetodene må fungere effektivt, og brukerne bør lett kunne bytte mellom dem uten problemer.
8. **Støtte for kontekstbaserte spørsmål:** Brukerne bør kunne stille spørsmål om artikler som chatboten foreslår uten å nødvendigvis åpne dem, noe som vil forbedre deres mulighet til å få relevant informasjon raskt og effektivt.

Brukertesten fremhevet behovet for en fleksibel og robust chatbot som kan håndtere et bredt spekter av forespørsler og tilby en sømløs brukeropplevelse. Videre testing og utvikling er nødvendig for å implementere disse forbedringene og sikre at chatboten oppfyller brukernes forventninger og behov.

5.10.3 Brukertest 3

Brukertest 3 avdekket flere viktige innsikter om brukeropplevelsen og funksjonaliteten. Testen var strukturert med 1-2 brukere 5 ganger, og fikk tilgang til chatboten, dens hovedfunksjoner, og et sett med seks oppgaver som de skulle gjennomføre. Formålet var å evaluere hvordan brukerne navigerer og løser oppgavene, samt identifisere eventuelle forbedringsområder.

Oppgave 1: Handlet om å la brukerne stille et valgfritt spørsmål før de startet. Resultatene viste at brukerne hadde forskjellige forventninger og behov. Enkelte brukere forsøkte umiddelbart å finne en artikkel, mens andre ønsket å snakke med en ekte person. Dette viser at det er flere måter for forskjellige brukere å ta chatboten i bruk.

Oppgave 2 og Oppgave 3: Handlet om å finne artikler om henholdsvis kultur og sport. Alle brukere klarte å finne relevante artikler, men majoriteten foretrakk å bruke chatfeltet fremfor knappefunksjonen. Dette antyder at chatfeltet må være robust nok til å håndtere alle funksjoner, ettersom knappene kunne ende opp ute av syn og brukerne måtte bla opp for å finne dem.

Oppgave 4: Involverte å finne informasjon om leveringstid på papiravisen. Her viste det seg at brukerne foretrakk å bruke chatfeltet for å finne kundeserviceinformasjon, noe som understreker behovet for å inkludere slike spørsmål i chatbotens funksjonalitet, i stedet for å kun stole på knappene.

Oppgave 5 og Oppgave 6: Handlet om å navigere til en artikkel og teste chatbotens funksjoner, samt stille spørsmål om artikkelen. Brukerne var generelt imponert over funksjonaliteten, men noen små bugs i layouten ble oppdaget. Et mer alvorlig problem ble identifisert med håndteringen av nøkkelord i chatfeltet. Når en bruker skrev et nøkkelord da de ikke mente å nevne et nøkkelord, spurte chatboten ofte om de ville lese artikler relatert til dette nøkkelordet. Dette kan oppleves som irriterende og forstyrrende når brukeren forsøker å snakke om helt andre tema.

Brukertesten har gitt verdifulle innsikter som kan forbedre chatbotens funksjonalitet og brukeropplevelse:

1. **Forbedret onboarding og klarere instruksjoner:** For å redusere forvirring bør chatboten gi mer tydelige veiledninger til nye brukere, spesielt om bruken av nøkkelord og tilgjengelige funksjoner.
2. **Robust chattefelt:** Siden brukerne foretrakk å bruke chatfeltet fremfor knappene, er det viktig at alle funksjoner også kan nås gjennom chatfeltet. Dette vil kreve en mer robust og fleksibel chatfunksjonalitet.
3. **Inkludering av kundeservicespørsmål:** Chatboten bør ha innebygd støtte for vanlige kundeservicespørsmål direkte i chatfeltet, slik at brukerne ikke må navigere tilbake til knappene for å få hjelp.
4. **Håndtering av nøkkelord:** Chatbotens nåværende håndtering av nøkkelord kan være forstyrrende, spesielt med en stor database. Det er nødvendig å finne en balanse mellom å tilby relevante artikler og å unngå å overvelde brukeren med forslag.

5. **Layoutbugs:** Små layoutfeil bør rettes for å sikre en jevn og problemfri brukeropplevelse.

Samlet sett har testen bidratt til å identifisere flere forbedringsområder som når de blir adressert, vil kunne gi en mer intuitiv og tilfredsstillende opplevelse for brukerne. Videre testing, spesielt med en realistiske systemer og database, vil være nødvendig for å validere disse forbedringene og sikre at chatboten møter brukernes behov effektivt.

5.10.4 Diskusjon rundt brukertester

Brukertestene ga oss mange verdifulle innsikter, men det er viktig å reflektere over både hva vi gjorde bra og hvilke områder vi kan forbedre oss på. Ved å analysere testene kan vi identifisere hva vi burde ha gjort annerledes, samt planlegge for fremtidig arbeid og testing.

Hva vi gjorde bra:

Problemanerkjennelse: Brukertestene var svært nyttige og ga oss verdifulle tilbakemeldinger som formet utviklingen av chatboten. Brukertest 1 viste tydelig interesse for en chatbot blant brukerne og ga oss viktige innspill om hva de ønsket seg. Dette bekreftet at produktet har potensial til å være gunstig for brukere når det settes i produksjon.

Tilpasning basert på tilbakemeldinger: Etter Brukertest 2 fokuserte vi på å forbedre funksjonaliteten i chatfeltet ved å øke funksjonaliteten med nøkkelord (tags) og optimalisere prompten og chathistory-funksjonen. Vi eliminerte også fundamentale feil som ville ødelegge brukeropplevelsen. Brukertest 3 viste at vi hadde tatt til oss de fleste innsiktene, og brukerne hadde en betydelig bedre opplevelse.

Hva vi kunne gjort bedre:

Tidligere testing: Vi skulle ønske vi hadde planlagt og gjennomført Brukertest 2 og 3 tidligere. Dette ville gitt oss rom for en ekstra brukertest i løpet av prosjektet. Tidligere tester ville også gitt oss muligheten til å teste Sunnmørspostens systemer, noe som ville resultert i en mer realistisk test med brukere og oppdragsgiver. Dette kunne ha gjort produktet klart for full produksjon tidligere.

Forbedring av chatfunksjonalitet: Selv om vi gjorde forbedringer i chatfeltet etter Brukertest 2, skulle vi ønske at vi hadde gjort mer. I de første testene brukte brukerne chatfeltet mer enn knappene, noe som betyr at vi burde ha hatt enda større fokus på å forbedre chatten for å øke tilgjengeligheten av funksjoner. Etter Brukertest 3 ble det klart at vi ikke kunne stole fullt ut på at GPT-modellen kunne besvare alle uforutsette spørsmål. Vi kunne ha jobbet mer intensivt med å forbedre denne delen av chatboten etter Brukertest 2.

Bedre forberedelser: Vi kunne hatt et mer systematisk opplegg og bedre forberedelser til brukertestene, spesielt Brukertest 2. Dette tok vi imidlertid til betraktning i Brukertest 3, hvor vi opplevde en mer strukturert og effektiv gjennomføring.

Testing i reelle systemer: Det ville vært gunstig å ha utført brukertester i Sunnmørspostens systemer. Dette kunne gitt oss mer presise data om hvordan chatboten fungerer i et reelt produksjonsmiljø. Fremtidig testing bør inkludere intensive tester i Sunnmørspostens systemer med store brukergrupper for å sikre at produktet ikke bare fungerer teknisk, men også oppfyller brukernes behov og forventninger. Begrenset tilgjengelighet og ressurser har dessverre hindret oss i å gjøre dette i løpet av vårt prosjekt.

5.10.5 Konklusjon av testene og fremtidig arbeid

Brukertestene har vist en klar interesse for en KI-basert chatbot på en nyhetsnettside og har gitt oss verdifulle innsikter som har forbedret utviklingsprosessen. Selv om vi har gjort betydelige fremskritt, er det klart at ytterligere arbeid er nødvendig for å sikre optimal funksjonalitet og brukeropplevelse.

For å fortsette med dette prosjektet i fremtiden, bør vi prioritere intensive tester i Sunnmørspostens systemer. Dette vil gi oss mer presise data om hvordan chatboten fungerer i et reelt produksjonsmiljø og tillate oss å gjøre nødvendige justeringer basert på tilbakemeldinger fra brukere. Det vil også hjelpe oss å finjustere produktet for å møte både brukernes og oppdragsgiverens behov.

Videre bør vi fokusere på å forbedre chat-funksjonaliteten for å redusere avhengigheten av ChatGPT sine svar. Selv om GPT-modellen gir relevante og nøyaktige svar på uforutsette spørsmål, kan vi ikke stole fullt og helt på det. Å integrere flere funksjoner i chatfeltet vil øke tilgjengeligheten og brukervennligheten av chatboten.

Til slutt bør vi gjennomføre en omfattende evaluering av hele systemet etter implementeringen i Sunnmørspostens miljø for å sikre at alle funksjoner fungerer som forventet og at produktet gir en positiv brukeropplevelse. Siden vi ikke har testet produktet i oppdragsgivers systemer, er det usikkert hvordan de vil fungere sammen. Selv om vi forventer at denne prosessen vil gå fint, kan det oppstå uforutsette problemer som krever justeringer.

Gjennom disse tiltakene kan vi videreutvikle chatboten til et ferdig produkt som er klart for full produksjon og som møter de høye standardene både vi og Sunnmørsposten setter.

6 Konklusjon

Gjennom dette prosjektet har vi utviklet en chatbot som bruker kunstig intelligens, spesifikt OpenAI's GPT-modeller, for å forbedre brukerinteraksjonen på Sunnmørspostens digitale plattform. Prosjektet har demonstrert potensialet for AI-teknologi til å formidle kompleks informasjon på en brukervennlig måte, noe som har økt brukernes engasjement og forståelse av nyhetsinnhold.

Vi har møtt flere utfordringer underveis, blant annet begrensninger knyttet til tilgang på ekte data og integrasjon med eksisterende systemer, som har påvirket testing og optimalisering av chatboten. Til tross for disse hindringene, har brukertester vist at chatboten bidrar til en mer interaktiv og engasjerende leseropplevelse, og oppfyller dermed våre opprinnelige mål.

Fremtidig arbeid bør fokusere på å utvide chatbotens funksjoner, forbedre dens evne til naturlig-språkforståelse og integrere mer omfattende databaser for å ytterligere forbedre brukeropplevelsen. Det er også essensielt å fortsette med etiske vurderinger rundt bruk av AI, spesielt i forhold til hvordan informasjonen behandles og presenteres, for å sikre at chatboten opprettholder høy journalistisk integritet og nøyaktighet. Til slutt understreker dette prosjektet viktigheten av brukersentrert design og kontinuerlig testing i utviklingen av AI-drevne verktøy.

Dette prosjektet illustrerer effektivt bruk av kunstig intelligens for nyhetsformidling, og viser hvordan avansert teknologi kombinert med brukerinnsikt kan forbedre digitale tjenester.

Vedlegg

A Vedlegg 1 - KI-deklarasjon

Dette vedlegget inneholder KI-deklarasjonen.

B Vedlegg 2 - Forprosjektsplan

Dette vedlegget inneholder forprosjektsplanen.

C Vedlegg 3 - Standardavtale

Dette vedlegget inneholder standardavtalen.

D Vedlegg 4 - Arbeidskontrakt

Dette vedlegget inneholder arbeidskontrakten.

E Vedlegg 5 - GitHub

Dette vedlegget inneholder en URL til GitHub-repositoriumet.

Referanser

Berni, A. og Y. Borgianni (2021). «FROM THE DEFINITION OF USER EXPERIENCE TO A FRAMEWORK TO CLASSIFY ITS APPLICATIONS IN DESIGN». I: *Proceedings of the Design Society* 1, s. 1627–1636.

Brophy, P. og J. Craven (2007). «Full Access Web Accessibility». I: *Library Trends* 55.4, s. 950–972.

Chowdhary, K. R. (2020). *Fundamentals of Artificial Intelligence*. Springer Nature India Private Limited. URL: https://doi.org/10.1007/978-81-322-3972-7.

Cooper, A. mfl. (2014). *About Face: The Essentials of Interaction Design*. Wiley.

Fauver, S. (2022). «Scrum for Two: How to Use Scrum Principles in a Two-Person Team». I: *BreakFree Solutions*. URL: https://www.breakfreesolutions.com/post/scrum-for-two.

Feng, Di, C. Lu og Shaofei Jiang (2022). «An Iterative Design Method from Products to Product Service Systems—Combining Acceptability and Sustainability for Manufacturing SMEs». I: *Sustainability*.

Jobin, Anna, Marcello Ienca og Effy Vayena (2019). «The global landscape of AI ethics guidelines». I: *Nature Machine Intelligence* 1, s. 389–399. URL: https://www.semanticscholar.org/paper/35ebed28c967acbfe38fa757f4e7023d075beaeb.

Lehmann, J. mfl. (2016). *Story-focused Reading in Online News and its Potential for User Engagement*. Universitat Pompeu Fabra.

Meier, Lukas J. mfl. (2022). «Algorithms for Ethical Decision-Making in the Clinic: A Proof of Concept». I: *The American Journal of Bioethics* 22, s. 4–20. URL: https://www.semanticscholar.org/paper/fa43ebca85750b0f16b8fda7167d7c7235009c0c.

Norman, D. (2013). *The Design of Everyday Things: Revised and Expanded Edition*. New York: Basic Books.

O'Brien, H. L. (2012). «Exploring user engagement in online news interactions». I: *Proceedings of the American Society for Information Science and Technology* 48.1, s. 1–10. URL: https://asistdl.onlinelibrary.wiley.com/doi/full/10.1002/meet.2011.14504801088.

Paredes, Cristian Mauricio Gallardo, Cristian Machuca og Yadira Maricela Semblantes Claudio (2023). «ChatGPT API: Brief overview and integration in Software Development». I: *International Journal of Engineering Insights*. URL: https://www.semanticscholar.org/paper/2e23667158760f5851a8b68649505f84e055de26.

Rogers, Yvonne, Helen Sharp og Jenny Preece (2023). *Interaction Design: Beyond Human-Computer Interaction, 6th Edition*.

Russell, Stuart og Peter Norvig (2016). *Artificial Intelligence: A Modern Approach*. 3rd. Pearson. URL: https://www.pearson.com/store/p/artificial-intelligence-a-modern-approach/P100000252830.

Sachdeva, S. (2016). «Scrum Methodology». I: *International Journal of Engineering and Computer Science* 5.6, s. 16792–16799.

Spinellis, D. (sep. 2005). «Version Control Systems». I: *IEEE Software* 22.5, s. 108–109. DOI: 10.1109/MS.2005.140.

Verschuren, P. og H. Doorewaard (2010). *Designing a Research Project*. The Hague: Eleven International Publishing.

Westin, A. F. (1967). *Privacy and Freedom*. Atheneum.