Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

Joachim Hauso Amundsen
Linda Gjerde Hageselle
Jenny Skjeret Valderhaug

Rustiki

Et wikirammeverk for Rust-utviklere

Bacheloroppgave i Dataingeniør
Veileder: Girts Strazdins
Mai 2022

NTNU
Kunnskap for en bedre verden

Joachim Hauso Amundsen
Linda Gjerde Hageselle
Jenny Skjeret Valderhaug

Rustiki

Et wikirammeverk for Rust-utviklere

Bacheloroppgave i Dataingeniør
Veileder: Girts Strazdins
Mai 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU

Kunnskap for en bedre verden

Oppsummering

Denne rapporten beskriver det teoretiske grunnlaget, utførelsen og løsningen for bacheloroppgaven «Rustiki - Et wikirammeverk for Rust-utviklere».

Oppgaven ble gitt av Ringrev - en bedrift basert i Ålesund som spesialiser seg på markedsføring. Bedriften benytter programmeringsspråket Rust og spesifiserte at prosjektet skulle utvikles med Rust fullstack-rammeverket MoonZoon med databasen ArangoDB. Oppgaven gikk ut på å bruke de nevnte teknologiene for å utvikle et wikirammeverk for Rust utviklere som ønsker å bruke Rust både i front- og backend.

Ringrev er interesserte i å bidra til at flere bruker Rust. Av den grunn var det bestemt at prosjektet skulle fortsette som et open source prosjekt etter bacheloroppgaven var fullført. I skrivende stund eksisterer det ingen andre open source wikirammeverk som er programmerte i Rust. Dette prosjektet blir dermed et nytt bidrag til Rusts utviklingsmiljø.

Metoder som ble brukt til kunnskaps- og erfaringsbygging var hovedsakelig lesing av dokumentasjon og studering av kodeeksempler. Prosjektet ble utført i henhold til agile metoders prinsipper, som vil si at arbeidet ble planlagt og utført i iterasjoner basert på tilbakemelding fra oppdragsgiver. Gruppen valgte interne sprinter med lengde på en uke. Dette førte til en syklus med kontinuerlig utvikling og tilbakemeldinger.

Hovedresultatet av oppgaven er et wikirammeverk som gruppen har navngitt Rustiki. Rustiki er tilgjengelig på Github som et open source prosjekt, der interesserte Rust-utviklere kan ta i bruk rammeverket og bidra til videre utvikling. Prosjektet har også bidratt til Rust fullstack-rammeverket MoonZoon, i form av å legge til og etterspørre funksjonalitet.

Abstract

This report describes the theoretical framework, execution and and solution of the Bachelor's thesis "Rustiki - A Rust Wiki Framework".

The assignment for this thesis was defined by Ringrev - a marketing business local to Ålesund. Ringrev uses the Rust programming language for their systems. They specified that the project is to be developed using Rust, the Rust fullstack framework MoonZoon and the database system ArangoDB. The task at hand was to create a wiki framework for Rust developers who want to use Rust for both front- and backend development.

Ringrev want to contribute to growth within the Rust community. For this reason, this project continues as an open source project after the product is delivered. At this point in time, there are no other existing wiki frameworks built in Rust. That makes this project a contribution to the Rust community.

Methods utilized for building the knowledge and experience needed, were mainly reading documentation and studying code examples. The team used agile methods for software development, consisting of work that was planned and executed in iterations based on feedback from the client. The group worked in internal sprints of one week, with stakeholder meetings every two weeks. This led to a cycle of continuous development and feedback.

The main result of the thesis is a wiki framework named Rustiki. Rustiki is available at GitHub as an open source project, which interested Rust developers can utilize and contribute to. Additionally, the thesis led to improvements in the fullstack framework Moonzoon.

Forord

Denne bacheloroppgaven var skrevet som det siste steget i en bachelorgrad innen dataingeniør ved NTNU i Ålesund.

Oppgaven ble valgt fordi gruppen var interesserte i å lære Rust, og syntes det hørtes spennende ut å bruke Rust innen webutvikling.

Vi ønsker å takke vår veileder Girts Strazdins for hans veiledning. Vi vil også takke de ansatte ved Ringrev, Arnaud Menant og Beate Stavdal Riise, som har vært entusiastiske og hjelpsomme hele veien. Til sist vil vi også takke stifter og hovedutvikler av rammeverket MoonZoon, Martin Kavík, som har vært behjelpelig med å svare på spørsmålene våre.

Oppgavetekst

Oppdragsgivers oppgave til gruppen var å utvikle et wikirammeverk til bruk for Rust-programmerere. Verktøyene som var definert for å oppnå dette er programmeringsspråket Rust, Rust fullstack-rammeverket MoonZoon, og databaseløsningen ArangoDB. Wikirammeverket må ha støtte for artikler, tags, bidragsytere og redigering. Etter gruppen har levert, vil produktet leve videre som et open source prosjekt. For videre detaljer om oppgaven se vedlegg B.

Innholdsliste

Figurer

Tabeller

Forkortelser og terminologi

Agile Manifesto	Retningslinjer for agil utvikling.
Agile metoder	Prinsipper benytter for utvikling av programvare.
API	Application Programming Interface. Programmeringsgrensesnitt
Aria-rolle	Beskrivelse for HTML-elementers roller, egenskaper eller tilstander.
Backend	Inneholder logikken for en applikasjon og koblingen mot databasen.
Backlog	Liste over oppgaver som ikke ble fullført innen gitt tid.
Bibliotek	Gjenbrukbar samling av kode.
Branch	Gren i Git strukturen, enten lokal eller remote på GitHub.
Brukertest	Utført på personer som ikke er involvert i prosjektet.
Code review	Diskusjon av en feature før den inkluderes i develop branch.
Commit	Lagrer endringer i koden til lokal versjonskontroll.
CRUD	Create, Read, Update, Delete.
Deploye	Hoste applikasjonen/nettsiden på en server.
Feature	Funksjonalitet til en programvare.
Fork	Egen kopi av et Git repositorie.
Frontend	Applikasjonslaget som inneholder selve nettsiden.
Fullstack	Backend og frontend.
Hashing	Kryptografisk omgjøring av data ved hjelp av en algoritme.
Header	Et element øverst på en nettside. Inneholder vanligvis logo, navigasjon og søkefelt.
ID	Identifikasjon i form av en unik kombinasjon tegn eller tall.
JWT	JSON Web Token.
Konseptskisse	Grovskisse av utseende på en nettside. Også kalt wireframe.
Kontinuerlig integrasjon	Metode for å hyppig inkludere ny funksjonalitet applikasjonen.
Kvalitative data	Data vanligvis bestående av tekst, som ikke kan fremstilles med tall.
Main branch	Git branch som bare inneholder fungerende kode.

Merge	Flette kode inn i annen kode. For eksempel feature branch inn i develop.
MVP	Minimum Viable Product.
Node	En enhet i en datastruktur.
NoSQL	Not only SQL. Database med alternativer for relasjoner i strukturen.
Push	Git funksjon for å sende kode til repository, gjennomføres etter en commit.
Query	Spørring til databasen.
Repositorie	Lagring av kildekode eks Github, GitLab.
REST	Representional State Transfer
Routing	Velge bane for trafikken – navigere mellom sider.
Schema	Struktur for database.
SCRUM	Rammeverk for å effektivt strukturere arbeid i henhold til agile prinsipper.
String	Datatype for tekst eller binær data.
Sprint	Arbeidsøkt ofte begrenset til en eller to uker.
Token	Bevis på at en økt tilhører en bruker.
Vektor	Rust: Liste av data av samme type. Likner på en Array men kan utvides når ny data legges til.
Versjonskontroll	System for å holde orden på endringer i koden.
VM	Virtuell maskin.

1 Introduksjon og relevans

Dette kapittelet introduserer oppgavens relevans og gruppens problemstilling.

1.1 Relevans

Gruppens oppdragsgiver Ringrev er et markedsføringsselskap. De vil hjelpe sine kunder å bygge kunnskap om hvordan de best mulig markedsfører seg selv på nett. I denne sammenheng ønsker Ringrev å tilby brukerveiledninger til kundene sine. De ønsker derfor å integrere en wiki i sitt system som kan brukes til dette formålet. Her kunne Ringrev valgt å bruke en eksisterende open source wiki, men en sentral faktor til hvorfor Ringrev ønsker å utvikle en helt ny wikiløsning er at de bruker programmeringsspråket Rust i sine systemer. I tillegg er det viktig for Ringrev å bidra til vekst innen Rust, som per nå har et relativt lite utviklingsmiljø. På grunn av dette bidrar Ringrev selv i prosjekter som er open source. Det finnes open source wikier utviklet i forskjellige programmeringsspråk, men ingen i Rust. Ringrev ser derfor muligheten for å tilby et open source wikirammeverk i Rust, som gjør det mulig for hvilken som helst Rust-utvikler å sette opp en egen wiki, og å bidra med ny funksjonalitet til rammeverket. På denne måten vil produktet som Ringrev selv trenger være et innovativt bidrag til Rusts utviklingsmiljø.

1.2 Problemstilling

Hvilke utfordringer kan oppstå når man kombinerer erfaringsbygging i Rust og MoonZoon med å utvikle et wikirammeverk med disse verktøyene, og hvordan kan resultatet bidra til Rusts utviklingsmiljø?

2 Teori og materialer

Dette kapittelet inneholder en oversikt over teori og materialer som er brukt i prosjektet.

2.1 Teori

2.1.1 Wiki

En wiki er en presentasjon av informasjon på nett i form av artikler, som igjen kjennetegnes av muligheten for bidrag til artiklene fra flere forfattere. Artiklene er koblet sammen med hyperlinks, og wikien har en søkefunksjon. Wikier på nett er vanligvis brukt for å presentere informasjon innen ett gitt tema (Wikipedia, 2022). Wikipedia er den meste kjente wikien, med mål om at de skal ha en altomfattende samling av all kunnskapen i hele verden (Wikipedia, 2022).

2.1.2 Rammeverk

Et rammeverk er en gjenbrukbar struktur som brukes som utgangspunkt for å bygge noe mer (Wikipedia, 2022). Innen programmering er et rammeverk en struktur av ferdiglagde løsninger som gjør utvikling raskere eller enklere. Et poeng med mange eksisterende rammeverk for utviklere er at lavnivås funksjonalitet skal være på plass fra starten slik at det kreves mindre oppsett for hvert prosjekt som blir startet (Ranjan, 2021).

2.1.3 Wikirammeverk

Et wikirammeverk er strukturen wikien blir bygd på, som inneholder en enkel måte å tilby forventede funksjoner til en wiki. Mens en wiki (2.1.1 Wiki) er en spesiell samling med informasjon, er altså et wikirammeverk (2.1.2 Rammeverk) strukturen som wikien er bygd på.

2.1.4 Frontend, backend og fullstack

Frontend og backend er en måte å skille mellom oppgavene til presentasjonslaget og det laget som har tilgang til data. I en typisk klient-server-modell vil klienten være det som kalles frontend, og server er det vi kaller backend (Wikipedia, 2022). Frontend står for å vise brukergrensesnittet til bruker og reagerer på interaksjon (Granevang, 2020). Backend kommuniserer med eventuelle databaser, og inneholder den underliggende logikken til en side eller applikasjon. Frontend og backend trenger et API for å kommunisere med hverandre (Granevang, 2020). Fullstack vil si både frontend og backend.

2.1.5 Rust

Gruppen brukte programmeringsspråket Rust til å utvikle produktet. Rust er et kompilert språk designet for høy ytelse og pålitelighet (Rust Team, u.d.). Konseptet eierskap i Rust er noe av det som skiller seg ut mest fra andre programmeringsspråk. Eierskap vil si at hver variabel har en enkelt eier, og at der kun kan finnes en eier til enhver tid. Hvis vi bruker en variabel som argument i en funksjon, blir denne funksjonen den nye eieren av variabelen,

og variabelen har ikke lenger en verdi i sin forrige kontekst. På grunn av eierskap kan Rust garantere minnesikkerhet og slippe å ha en garbage collector (Klabnik & Nichols, u.d.). Eierskap kombinert med sjekk av type tillater at vanlige trådprogrammeringfeil blir oppdaget ved compile-time istedenfor runtime (Klabnik & Nichols, u.d.).

En studie utført i 2017 så på energieffektivitet for forskjellige programmeringsspråk. I den studien slår Rust alle språk med unntak av C når de sammenlignet tids- og energibruk. På minnebruk kom Rust inn på en sjetteplass bak Ada og C++ (Pereira, et al., 2017). Selskaper som bruker Rust i deler av sin kodebase inkluderer blant annet Amazon, Microsoft, Meta, Discord og DropBox (Dreimanis, 2020).

2.1.6 Agile metoder

Agil utvikling (Agile Alliance, u.d.) er et bredt begrep som kan omfatte mange ulike rammeverk og metoder som stammer fra The Agile Manifesto. Dette omfatter eksempelvis sprinter med følgende refleksjon, og bruk versjonskontroll med branching. Det som utmerker seg med agil programvareutvikling er et fokus på menneskene involvert i prosjektet og hvordan de samarbeider for å løse oppgavene. Løsninger kommer da som følge av samarbeid og utvikling av teamet, gjennom konstant læring og deling av kunnskap. Dette er styrken med agile metoder når målet kan endres underveis, eller er av en type med mange ukjente momenter som må løses når de oppstår.

De aktuelle punktene fra The Agile Manifesto for dette prosjektet er:

6: Utveksling av informasjon innad i gruppen skjer best ansikt til ansikt.

7: Fungerende programvare er den beste måling på fremgang.

10: Minimering av unyttig arbeid.

12: Teamet reflekterer over hvordan de kan forbedre arbeidsflyten og benytter dette for å justere arbeidet fremover (Agile Alliance, u.d.).

2.1.7 Versjonskontroll med branching

Når man følger feature branch-modellen, blir features utviklet på en isolert branch i versjonskontrollen med hensikt å forenkle samarbeid mellom flere utviklere (Atlassian, u.d.). Dette betyr at ulike utviklere kan arbeide på en individuell feature uten å være avhengig av at andre utviklere må være ferdig med sin feature først. Dette medfører at main branch alltid inneholder kjørbar og fungerende kode som er sentralt for kontinuerlig integrasjon.

En fordel med denne metoden er code review - når en feature er ferdig blir den vurdert av andre utviklere før den blir merget inn i develop. Formålet med denne prosessen er at utviklere skal diskutere kode for å forbedre prosessen. Dette er dermed en arbeidsmetode som følger prinsippet om samarbeid og deling av kunnskap for å forbedre produktet i henhold til agile metoder (Atlassian, u.d.).

2.1.8 Sprint

En sprint (Drumond, u.d.) er en tidsbegrenset periode hvor planlagt arbeid blir utført. Tidsrammen kan variere fra en uke til en måned avhengig av teamets behov. Den ønskede

effekten er at teamet kan bryte ned et stort prosjekt i mindre deler og dermed sørge for en effektivisert fremgang på oppgaven.

2.1.9 Scrum

For å dokumentere sprinter og arbeidsoppgavene brukes ofte ulike programvarer for samarbeidsrammeverket Scrum. Her er det viktig å vite at agile metoder er prinsipper for utvikling mens Scrum er et rammeverk som følger disse prinsippene. Scrum skal føre til strukturert arbeid med god flyt og gi ledelsen viktig informasjon for progresjonen av prosjektet. Eksempler på dette er at oppgaver i sprinten som ikke blir fullført innen forventet tid blir flyttet til en backlog. Dette gir da mulighet for å analysere hvorfor dette ikke ble oppnådd og hva kan gjøres for å effektivisere prosessen eller disponere mere tid på slike oppgaver i neste sprint (Drumond, u.d.).

2.1.10 Personas

Personas (Agile Alliance, u.d.) er fiktive personer som skapes for å ha noen å utvikle produktet mot. Dette er nyttig når brukeropplevelsen er viktig. Dette skal illustrere at systemet har ulike brukere med ulike mønstre for hvordan de interagerer med systemet. Dette bidrar dermed til å unngå at utvikleren designer produktet for egne preferanser i motsetning til kunden og brukernes behov (Agile Alliance, u.d.).

2.1.11 Konseptskisse av nettside

En konseptskisse eller wireframe (Usability.gov, u.d.) er en illustrasjon av nettsidens brukergrensesnitt. Dette er hovedsakelig størrelsen på de ulike komponentene og prioriteringen de har. Den demonstrerer også forventet funksjonalitet og adferd fra disse funksjonene. En konseptskisse inneholder vanligvis ikke farger, styling eller bilder, men det kan benyttes ulike gråfarger for å vise kontraster (Usability.gov, u.d.). Dette er dermed en effektiv måte å definere de viktige tingene og strukturere designet før man begynner å kode nettsiden. Det er også ideelt med brukertesting av skissene tidlig i prosessen for å skaffe viktig tilbakemelding fra brukeren slik at man raskt kan iterere og ferdigstille de viktigste funksjonene i henhold til agil utvikling.

2.1.12 NoSQL database

NoSQL (MongoDB, u.d.) kom som et resultat av prisfallet på lagringsmedier i perioden 2000-2009. Dette medførte at kostnaden i industrien endret seg til at utviklerne ble den største utgiftsposten. Dette var en av grunnene til at NoSQL hadde som formål å forenkle hverdagen til utviklerne med et enklere system for å lagre data. Interessen for NoSQL økte i samme periode som The Agile Manifesto ble populært i bransjen. Dette var grunnet et økende behov for fleksibilitet og raske endringer under utviklingen av programvare. De vanligste fordelene med NoSQL databaser er:

1. Fleksible schema-filer, ofte generert automatisk.
2. Horisontal skalering ved å legge til nye noder for å håndtere økt belastning. Dette er lettere å gjennomføre sammenliknet med tradisjonelle relasjonelle databaser. Dette er løst ved at samlinger er isolerte og da enkelt kan distribueres over nye noder.

3. Raskere queries til databasen. På grunn av strukturen på samlinger trenger ikke en query å bli med i mange tabeller. Dette reduserer tiden det tar å gjennomføre en query.
4. Enklere bruk for utviklere. Muligheten for å endre schema-filen gjør det mulig å utvikle applikasjonen før databasen er ferdig modellert. Det gir også økt mulighet for å endre strukturen sent i utviklingen (MongoDB, u.d.).

2.1.13 Normans designprinsipper

Donald Norman (Edspresso, u.d.) er kjent for forskning innen brukervennlighet og menneskers interaksjon med datamaskiner. Han har definert seks designprinsipper man bør tenke på når man designer brukergrensesnitt:

1. **Synlighet**
   Brukere skal kunne se hvilke muligheter de har, og forstå hvordan de får adgang til de, bare ved å se på nettsiden. Dette innebærer å ikke ha unødvendig informasjon der den ikke er nyttig.

2. **Respons**
   Brukeren må få respons etter hver handling de foretar, slik de vet om handlingen var suksessfull eller ikke.

3. **Samhandling**
   Det må være en kobling mellom hvordan noe ser ut og hvordan det er brukt.

4. **Mapping**
   Forholdet mellom kontroller og hvordan de brukes eller hvilken effekt de har, som for eksempel at man går nedover på siden når man drar scrollbaren nedover.

5. **Begrensninger**
   Brukeren har begrensede valgmuligheter for å unngå å bli overveldet og for å unngå å ta feil valg.

6. **Konsistens**
   Konsistent gjenbruk av kjente mønster gjør at brukeren lærer og kan navigere (Edspresso, u.d.). Konsistens kan deles opp i intern konsistens og ekstern konsistens (Krause, 2021). Intern konsistens omhandler konsistensen innenfor eget produkt eller en serie produkter, mens ekstern konsistens tar hensyn til at brukere har forventninger til ditt produkt basert på andres produkter. Et eksempel på dette er at man kan trykke på en nettsides logo for å komme tilbake til forsiden (Krause, 2021).

2.1.14 Universell utforming

I Norge er det bestemt med forskrift at både offentlige og private virksomheter må oppnå WCAG 2.0 eller tilsvarende standard på sine nettløsninger (Lovdata, 2013). Kriteriene man må oppfylle er de på nivå A og AA, med unntak av tre kriterier som omhandler lydbeskrivelser og live undertekst. Offentlig sektor må fra februar 2023 møte standarden WCAG 2.1 som del av EUs webdirektiv (Digitaliseringsdirektoratet, u.d.).

Retningslinjer for universell utforming (W3, 2008):

Mulig å oppfatte

1.1: Tekstlig alternativ
Det må finne tekstlig alternativ til alt innhold som ikke er tekst. Unntak til regelen er hvis innhold kun fungerer som dekorasjon, formatering eller er usynlig.

1.2: Alternativ til lyd og video
Der skal være alternativer til lyd og video, og der skal være teksting tilgjengelig for alle forhåndsinnspilte lydopptak.

1.3: Kan tilpasses
Innhold skal kunne presenteres på andre måter uten å miste informasjon eller struktur. Dette betyr blant annet at rekkefølge på innhold skal kunne oppdages i koden, og at instruksjoner for å forstå eller interagere med innhold ikke kun er kommunisert med form, størrelse eller plassering.

1.4: Kan skilles mellom
Det skal være enkelt for brukere å se eller høre innholdet. Innebærer blant annet at farge ikke skal fungere som eneste visuelle måte å vise en handling, at kontrasten mellom tekst og bakgrunn er minst 3:1, og at tekst kan forstørres opptil 200% uten tap av innhold eller funksjonalitet.

Mulig å betjene

2.1: Tastaturtilgjengelighet
All funksjonalitet skal være tilgjengelig fra et tastatur. Det vil si at man ved bruk av tastatur skal kunne navigere mellom ulike komponenter og klikke på det som er klikkbart.

2.2: Nok tid
Brukere må ha nok tid til å lese og bruke innholdet.

2.3: Anfall
Innholdet må ikke bli designet på en måte som kan føre til epileptiske anfall. Innebærer at der ikke må være innhold som blinker mer enn tre ganger per sekund.

2.4: Navigerbart
Det skal være mulig for brukere å navigere, finne innhold og finne ut hvor de befinner seg. Dette innebærer at innholdet skal kunne fokuseres på med tastatur i en logisk rekkefølge. Man skal kunne skjønne poenget med en link ved hjelp av kun linkteksten eller sett i sammenheng med konteksten. Der skal være overskrifter for å vise tema eller funksjonalitet. Indikatoren for elementer man markerer med tastaturet må kunne gjøres synlig.

Forståelig

3.1: Lesbart
Tekst skal være lesbar og forståelig. Det vil blant annet si at språket til nettsiden

skal kunne oppdages i koden. Språket til tekst på siden skal også kunne oppdages i kode.

3.2: Forutsigbart
En nettside skal opptre og vises som forventet. Dette betyr at å markere en komponent ikke forandrer konteksten, at input til en komponent ikke automatisk forandrer konteksten, og at navigasjonselementer som finnes på flere enkeltsider dukker opp i samme rekkefølge hver gang.

3.3: Instruksjoner for input
Man skal hjelpe brukere å rette feil de gjør. Her legges det vekt på å ha instruksjoner til bruker i form av tekst og at man gir tekstlig beskjed når bruker gir feil input.

Robust

4.1: Kompatibelt
Sørge for best mulig kompatibilitet med nåværende og fremtidige brukermidler. Det innebærer at navn og Aria-rolle for alle komponenter kan oppdages i koden og at ID-attributter er unike (W3, 2008).

2.1.15 GDPR

General Data Protection Regulation er verdens mest omfattende lov for personvern og sikkerhet som ble iverksatt 25 mai 2018. Loven har som mål å regulere hvordan bedrifter samler og benytter data i sammenheng med borgere innen EU. Mulige konsekvenser ved brudd av GDPR er svært høye bøter for bedriften. I denne oppgaven er det lagring av data som e-post. passord og annen informasjon som kan brukes for å identifisere et individ som er mest relevant (Proton Technologies, 2020).

Relevant terminologi fra GDPR:

- Personlig data: Informasjon som kan relateres til et individ enten direkte eller indirekte. Eksempler for denne oppgaven er e-post, bidragsgiver og brukernavn.
- Dataprosessering: Handling utført på dataen, dette dekker automatiske og manuelle interaksjoner. Eksempler for denne oppgaven: lagre e-post, strukturere e-post i database, bruk av e-post og sletting av dataen.

En person har i henhold til GDPR disse rettighetene:

1. Retten til å bli informert.
2. Retten til adgang.
3. Retten til utbedring.
4. Retten til å bli slettet.
5. Retten til begrenset databehandling.
6. Retten til flytting av data.
7. Retten til å protestere.
8. Rettigheter i henhold til automatisert beslutningstagning og profilering.

Formålet med disse rettighetene er å gi individet økt grad av kontroll over informasjonen de deler med bedrifter (Proton Technologies, 2020).

2.1.16 Scripting

Et scripting-språk (GeeksforGeeks, 2022) er et språk som blir tolket istedenfor kompilert. Det vil si at hver linje med kode blir lest etter hvert som den kjøres. Kompilert kode blir tolket før den kjøres. Vanligvis vil et kompilert program kjøre raskere enn et tolket program fordi det blir gjort om til lavnivås maskinkode før det kjører. Eksempler på tradisjonelle scripting-språk er JavaScript, Python og PHP. Det går likevel an å lage kompilatorer for disse språkene og bruke de som kompilerte språk (GeeksforGeeks, 2022). JavaScript er det språket som er mest brukt til scripting i nettleser, og alle nettlesere kan tolke dette språket.

JavaScript blir lagt til på sider via HTML-dokumenter og interagerer med DOM (Wikipedia, 2022) – eller Document Object Model, som er en interface som gjør det mulig å behandle HTML eller XML-dokumenter som en trestruktur der hver del av dokumentet er representert via en node (Wikipedia, 2022).

2.1.17 WebAssembly

WebAssembly (Mozilla, 2022) eller WASM er en ny variant kode som kjører i nettleseren. Det er et lavnivå assembly liknende språk i binært format som gir det muligheten for å kjøre med svært god hastighet. Denne teknologien gjør det også mulig å kompilere språk som C/C++, C# og Rust slik at de kan benytte på en nettside. WASM er laget for å fungere som en utvidelse for Javascript og de kan kjøres sammen. Dette gir muligheter for å utnytte ytelsen og effektiviteten av WASM for deler av nettsiden mens man benytter Javascript for fleksibiliteten og store mengden biblioteker som eksisterer på andre deler av nettsiden. Denne funksjonaliteten krever heller ikke kompetanse på å skrive WASM direkte siden dette håndteres av kompilatoren (Mozilla, 2022). WASM brukes vanligvis som et kompilert språk, men kan også brukes som tolket språk (Wikipedia, 2022).

For dette prosjektet er det WASM som gjøre det mulig å benytte Rust på Frontend fordi MoonZoon kompilerer denne delen til en WASM binær fil.

2.1.18 Container images

Et container image er en umodifiserbar statisk fil som inneholder kjørbar kode som kan kjøre i en isolert prosess på en pc, skytjeneste eller annen infrastruktur. Denne filen inneholder alle avhengigheter som programvaren trenger for å kjøre. Den inneholder også konfigurasjoner for å kjøre på plattformer for håndtering av kontainere, eksempelvis Docker. Imaget deler kjernen til operativsystemet det kjører på, det er derfor mulig å spesifisere hvilken type vert som er målet for filen (TechTarget, u.d.).

2.1.19 Open source

Open source er programvare som ligger åpent for alle å lese, ofte på portaler som GitHub. De er dekket av en open source lisensavtale eksempelvis MIT eller Apache. Dette gir brukere fri tilgang til å lese, benytte, modifisere og dele programvaren (Open Source initiative, u.d.). Hvis man ser på open source-prosjekter på GitHub som et utgangspunkt, kan man foreslå ny eller endret funksjonalitet ved å lage en issue i prosjektets repositorie. Det er mulighet for å diskutere issues med andre utviklere, da hver issue har et eget kommentarfelt. Det er vanlig at hvem som helst kan bidra til prosjektet ved å klone eller lage en fork av prosjektet og gjøre endringer i koden. Et klonet prosjekt er fortsatt

synkronisert med repositoriet man klonet, mens en fork ikke blir synkronisert. Etter man har pushet endringer i koden til en egen branch, kan man legge inn en pull-request til open source-prosjektets repositorie. Dette vil si at du ber open source-prosjektet om å merge din kode inn i den eksisterende koden. Deretter kan noen med administratortilgang til open source-repositoriet be om eventuelle forandringer i koden du commitet. Pull-requesten din blir oppdatert når du pusher ny kode til branchen du jobber på. Hvis administrator er fornøyd med koden din, kan de merge koden inn i repositoriet til open source-prosjektet (GitHub, u.d.). Ønsker man å bidra til et open source-prosjekt, er det viktig at man først sjekker om de har regler for hvordan man bidrar – der finnes ofte en «code of conduct».

2.1.20 Sikkerhet

2.1.20.1 HTTPS

HTTP eller HyperText Transfer Protocol er en protokoll for kommunikasjon. HTTPS er en utvidelse av HTTP, hvor kommunikasjonen blir kryptert ved hjelp av TLS (Transport Layer Security). HTTPS brukes til å autentisere nettsiden man er på, og til å beskytte data fra lesing eller endring under overføring. Dette i motsetning til HTTP, der overført data kan leses eller endres av en eventuell mellompart (Wikipedia, 2022). HTTP er en tilstandsløs protokoll og kan ikke huske informasjon. For å løse denne utfordringen kan man bruke en session eller en token (Hsu, 2018).

2.1.20.2 Session

En session er en tidsbegrenset informasjonsutveksling mellom to eller flere enheter eller endepunkt. En vanlig tilnærming til bruk av sessions er at bruker får tildelt en unik session ID når de besøker en webserver (Wikipedia, 2021). Denne blir typisk oppbevart i en cookie i brukerens nettleser, og blir sendt sammen med hver HTTP-request. På nettsider som bruker sessions til brukerautentisering blir brukeren tilsendt en ny session ID etter de logger inn (Hsu, 2018). En session kan bli utsatt for hijacking-angrep der en uautorisert bruker får tilgang til informasjon eller tjenester i et system (Wikipedia, 2022).

2.1.20.3 Token

Det er vanlig å bruke JSON Web Token til autentisering istedenfor sessions. Tokens blir validert ved hjelp av asymmetrisk kryptografi – også kjent som public/private key. Dette vil si at ene parten, vanligvis server, signerer med en hemmelig privat nøkkel, og den andre parten med den korresponderende offentlige nøkkelen (Wikipedia, 2022). Klient oppbevarer deretter JWT-tokenet lokalt, og detter tokenet blir inkludert i headeren til hver request de sender, og server validerer tokenet for hver request (Hsu, 2018).

2.1.20.4 Hashing

For å unngå å oppbevare brukeres passord i klartekst kan man velge å bruke en hash-funksjon, som er en algoritme for å gjøre om data til en representasjon av seg selv – vanligvis i form av en string. Samme algoritmen for hashing vil alltid resultere i samme hash. For å finne ut hva en innholdet av en hashet verdi var i utgangspunktet, må man bruke samme algoritmen.

2.1.21 Kobling og kohesjon

Kobling (GeeksforGeeks, 2021) handler om hvor avhengige moduler eller funksjoner er av hverandre. Hvis to moduler bruker mange av hverandres funksjoner, er der en høy grav av kobling. Man ønsker å ha en lav grad av kobling.

Kohesjon handler om graden av sammenheng innenfor en modul eller funksjon. Det er ønskelig at alle deler innenfor en modul eller funksjon har en begrenset og definerbar oppgave. Man vil helst oppnå en høy grad av kohesjon (GeeksforGeeks, 2021).

2.1.22 HTML

HTML eller HyperText Markup Language er det mest brukte markup-språket for dokumenter i nettleser. HTML beskriver strukturen av et dokument. Det er mulig å definere utseendet av et HTML-dokuments elementer ved hjelp av CSS. Man kan også bruke scripting-språk som for eksempel JavaScript for å få ønsket interaksjon (Wikipedia, 2022). Hvordan HTML blir brukt er viktig for universell utforming på nettsider, da for eksempel skjermlesere bruker denne strukturen aktivt til å best mulig presentere innholdet på en side til en bruker (Mozilla, 2022). For å forklare strukturen enda bedre kan man bruke Aria fra Web Accessibility Initiative til å skrive inn mer betydningsfulle beskrivelser av elementers roller, tilstander og egenskaper (Mozilla, 2022).

2.2 Materialer

2.2.1 Utvikling og testing

2.2.1.1 MoonZoon

MoonZoon er et fullstack Rust-rammeverk som tillot gruppen å bruke Rust både i backend og frontend. Dette er mulig fordi koden blir kompilert til WebAssembly (2.1.17 WebAssembly). MoonZoon vektlegger at man ikke skal trenge å forholde seg til mange forskjellige språk og verktøy for å lage enkle webløsninger. Utvikleren bak rammeverket ønsker at man ikke skal trenge for eksempel JavaScript, CSS, HTML og SQL til å lage et prosjekt, men at brukerne av rammeverket likevel skal ha mulighet til å bruke alt dette dersom de ønsker det (Kavík & Northman, 2021). MoonZoon bruker HTML, CSS, og delvis JavaScript (2.1.16) i bakgrunnen, men brukere av rammeverket trenger ikke forholde seg til dette med mindre de skal utføre spesifikke oppgaver som ikke enda finnes støtte for.

Et MoonZoon-prosjekt er delt inn i fire hoveddeler: backend, frontend, public og shared. Backend og frontend inneholder kode som oppfyller rollene som forklart i kapittel 2 (2.1.4 Frontend, backend og fullstack), mens public vanligvis inneholder stilark og bilder, og shared definerer hva som kan sendes frem og tilbake mellom frontend og backend.

2.2.1.2 ArangoDB (Community Edition)

Gruppen brukte databaseløsningen ArangoDB. ArangoDB er et NoSQL databasesystem som støtter lagring av dokumenter og grafer (ArangoDB, 2021).

2.2.1.3 Aragog
Gruppen brukte Rust-biblioteket Aragog til kommunikasjon mellom koden og databasen. Aragog lar brukeren håndtere dokumenter og grafer for ArangoDB-databaser med forhåndsdefinerte funksjoner for alle CRUD-operasjoner (Aragog, u.d.).

2.2.1.4 Firebase Auth
Firebase Auth ble brukt til registrering og innlogging. Denne løsningen støtter registrering og innlogging av brukere med e-post og passord. Man kan også sette opp verifiseringsepost, resetting av passord og forandring av passord eller epost (Google Developers, 2022). Dette fungerer ved at man enten bruker eksisterende biblioteker for å kommunisere med Firebase Auth, eller at man sender requests til Firebase Auth REST API.

2.2.1.5 Fire Auth
Gruppen brukte Rust-biblioteket Fire Auth til å kommunisere med Firebase. Fire Auth har funksjoner som gjør kall til Firebase Auth REST API, og støtter registrering, innlogging, verifiseringsepost, resetting av passord, fornying av token og oppdatering av epost eller passord (Fire Auth, 2022).

2.2.1.6 IntelliJ
Gruppen brukte IntelliJ som IDE i prosjektet.

2.2.1.7 GitHub
Gruppen brukte GitHub til versjonskontroll for koden.

2.2.1.8 Docker
Docker ble brukt til å bygge et image av ArangoDB.

2.2.1.9 DigitalOcean
Gruppen brukte en Droplet i skytjenesten DigitalOcean til å hoste et Docker image av databaseløsningen ArangoDB.

2.2.1.10 Lighthouse
Gruppen brukte Lighthouse til å sjekke om produktet de utviklet møter standarder for webutvikling på lastetid, universell utforming, unødvendig kode og optimalisering for søkemotorer (Google Developers, 2021).

2.2.1.11 Balsamiq Cloud
Gruppen brukte Balsamiq Cloud til å lage konseptskisser av produktet.

2.2.2 Prosjektstyring og kommunikasjon

2.2.2.1 Jira
Prosjekthåndteringsverktøyet Jira ble brukt av gruppen til å lage og håndtere sprinter, samt å logge timer.

2.2.2.2 Confluence
Gruppen brukte Confluence til å planlegge møter, ta møtenotater og å skrive refleksjoner over sprints. Confluence har wikifunksjonalitet og kan linkes til Jira.

2.2.2.3 OneDrive
Gruppen brukte en delt mappe i OneDrive til samskriving og til å oppbevare alle felles dokumenter.

2.2.2.4 Outlook
Gruppen brukte Outlook til å sende møteinnkallinger og agenda til arbeidsgiver og veileder.

2.2.2.5 Discord
Discord ble brukt til kommunikasjon internt i gruppen og med oppdragsgiver. Arbeidsgiver ønsket møtereferat, deling av ressurser og Rust-spørsmål på deres Discordserver.

3 Metode

I dette kapittelet blir det forklart hvilke forskningsmetoder gruppen brukte for å tilegne seg nødvendig kunnskap til å utføre prosjektet. Kapittelet tar også for seg hvilke valg som ble gjort innen teknologi og utviklingsmetode, og grunnen bak disse valgene.

3.1 Forskningsmetode

3.1.1 Informasjonsinnsamling

Gruppen skaffet seg en bedre forståelse for hva oppgaven innebar ved å spørre utdypende spørsmål til oppdragsgiver. Under prosjektet var Ringsrevs CTO tilgjengelig for Rust eller MoonZoon-relaterte spørsmål gruppen hadde. Dersom gruppen møtte utfordringer som ikke fantes informasjon om på nettet, kunne han spørres. Spørsmål som han ikke kunne svare på kunne tas videre til Martin Kavík, som er hovedutvikleren av MoonZoon.

3.1.2 Litteraturstudium

Siden gruppen ikke hadde tidligere kjennskap til programmeringsspråket Rust, ble en stor og viktig del av prosjektet å lære det som trengtes for å kunne bruke dette språket. Ringrev fortalte gruppen at den viktigste ressursen for å oppnå nødvendig kompetanse var den offisielle boken om Rust (2.1.5 Rust). Denne boken stod for mesteparten av kunnskapen gruppen skaffet seg, supplert med andre kilder tilsendt av Ringrev.

Gruppen satte seg også inn i hva som forventes av en wiki ved å lese om dette og å studere brukergrensesnittet til eksisterende løsninger, som Wiki.js, DokuWiki, og Wikipedia.

3.1.3 Eksempelfølging

Den offisielle Rust-boken inneholder noen eksempelprosjekter man kan sette opp, så gruppen brukte disse som øvelse mens de gikk gjennom Rust-boken. I tillegg gikk gruppen gjennom et Rust-kurs på Udemy og fulgte eksemplene i det kurset (Stocks, 2021).

På grunn av lav grad av dokumentasjon for rammeverket MoonZoon, ble det å studere eksempelprosjekter laget i dette rammeverket grunnleggende for at gruppen skulle vite hvilken funksjonalitet som fantes.

3.1.4 Brukertesting

Gruppen lagde konseptskisser (2.1.11 Konseptskisse av nettside) som illustrerte forståelsen de hadde av hva en wiki skal være som de både presenterte til oppdragsgiver og utførte brukertester på. Dette for å tilegne seg informasjon om hvilke forbedringer som kunne gjøres og få et bilde av hvilke forventninger brukere har til en wiki.

Senere i prosjektet utførte gruppen også brukertesting på selve produktet etter at ønsket funksjonalitet var oppnådd.

3.2 Valg av teknologi og utviklingsmetode

Dette delkapitlet tar for seg valgene gruppen har tatt i forhold til teknologi og metoder, og utdyper tankegangen bak valget.

3.2.1 Utviklingsmetode

3.2.1.1 Agile prinsipper for utvikling

Agil utvikling (2.1.6 Agile metoder) ble brukt til denne oppgaven siden dette er den mest utbredte utviklingsmetoden, og den gruppen har mest erfaring med gjennom studiene. Gruppen spurte oppdragsgiver om typiske og mulige brukere for å få bedre innsikt i hva som trengtes. Gruppen lagde i tillegg personas (2.1.10 Personas) og user stories for å idémyldre videre. Gruppens personas og user stories er vedlagt i kravdokumentasjon (vedlegg B).

3.2.1.2 Scrum

Gruppen brukte prinsipper fra det agile rammeverket Scrum (2.1.9 Scrum) for å sikre god progresjon for prosjektet. Arbeidet var planlagt utført med en ukes sprinter (2.1.8 Sprint) internt, og møter med oppdragsgiver og veileder hver andre uke. Hver sprint ble planlagt og fordelt med epics, stories, tasks og subtasks. Dette ga en strukturert arbeidsfordeling med god oversikt over hva som måtte gjøres til enhver tid. Grunnet uforutsigbarhet på antall timer det ville ta å fullføre oppgavene valgte gruppen å være fleksible på allokeringen av tid på hver oppgave, og hadde forståelse for at backloggen kunne bli stor. Gruppens sprint reports er vedlagt i gruppens prosjekthåndbok (vedlegg D).

3.2.1.3 Utforsking av MoonZoon med kreativ prosess

For et språk eller rammeverk som har mye dokumentasjon, ville det å utforske denne dokumentasjonen være et viktig verktøy for gruppen. Siden MoonZoon har lite dokumentasjon måtte gruppen bruke andre metoder for å lære om rammeverkets funksjonalitet. Av denne grunn ble MoonZoons eksempelprosjekt utgangspunktet for gruppens læring. Da gruppen begynte å se på rammeverket MoonZoon, fantes der 16 eksempelprosjekter som viste fram forskjellig funksjonalitet. Et av de større prosjektene var «pages», som er en nettside som har routing til fire forskjellige sider. Der var også inputfelt og knapper på to av sidene. Dette var viktig funksjonalitet for gruppens prosjekt, så dette eksempelprosjektet ble derfor valgt som et utgangspunkt for utforskning.

Gruppen brukte en uke til å eksperimentere fritt med MoonZoons frontend-funksjonalitet ved å lage hver sin nettside. Dette innebar grunnleggende ting som inputfelt, bilder, struktur med rader og kolonner og andre MoonZoon-relaterte elementer (2.2.1.1 MoonZoon). Gruppemedlemmene arbeidet individuelt med hvert sitt prosjekt, men var samlet for å dele informasjon og diskutere muligheter med rammeverket. Det ble også avtalt med oppdragsgiver å demonstrere prosjektene ved neste møte. Dette skulle vise progresjon med teknologien og mulighet for å spørre Ringrev om råd for videre utvikling.

Denne metoden skulle hjelpe studentene med å gå fra teori og eksempler, til å skrive egne linjer med kode. Dette kan være krevende når man bruker nye språk og rammeverk. Dette var en aktuell utfordring for gruppen fordi MoonZoon er i en tidlig fase av utviklingen av rammeverket, og det fantes begrenset med ressurser å ta utgangspunkt i.

3.2.1.4 Versjonskontroll med feature branches

Gruppen valgte å bruke Git feature branch-modellen for versjonskontroll og utvikling. Som beskrevet i kapittel 2 (2.1.7 Versjonskontroll med branching) bidrar dette til å ha en fungerende versjon av koden på en egen branch, i tillegg til flere forskjellige feature-branches. Slik kan flere utviklere arbeide samtidig uten å være avhengig av fullført arbeid fra andre. Det ble bestemt at gruppen skulle committe ofte, med formål om at det ikke skulle bli for store forskjeller på hver enkelt feature-branch og develop-branchen. Det ble også besluttet at når en feature kompilerte skulle den branchen bli pushet til prosjektets GitHub-repositorie. Når en feature var ferdig eller hadde tilnærmet ferdig funksjonalitet ble den merget inn i develop-branchen. Grunnet gruppens kunnskapsnivå om Rust og MoonZoon var det utfordrende å estimere tidsbruk på å utvikle features. Dette kan føre til større forskjeller enn ønsket i koden på en feature-branch og på develop-branchen. Det ble derfor avtalt i gruppen at når en feature ble lagt til i develop skulle de som arbeidet på andre features dra ned siste endringene til sin feature-branch. Dette for å forhindre store konflikter ved neste merge til develop.

3.2.1.5 Code review

Gruppen bestemte seg for å ha code review før en feature branch ble inkludert i develop-branchen. Dette følger naturlig i Git feature branching-modellen som gruppen benyttet seg av (2.1.7 Versjonskontroll med branching). Det ga også mulighet for å forklare og diskutere ting hvert gruppemedlem oppdaget med MoonZoon. Dette var nyttig da gruppen ikke hadde tidligere erfaring med dette rammeverket eller programmeringsspråket Rust. Evalueringen av koden var planlagt gjennomført enten når en feature var ferdig, eller når den kompilerte ved slutten av en sprint og var nesten ferdig.

Normalt sett skal en code review bli gjennomført med en erfaren utvikler. Dette var ikke mulig for dette prosjektet på grunn av manglende erfaring innad gruppen når det gjelder Rust og MoonZoon. Det ble dermed benyttet for gruppen som en mulighet for å diskutere koden og oppdage bedre løsninger.

3.2.2 Valg av teknologi

3.2.2.1 Rust

En sentral del av oppgaven var kravet om at programmeringsspråket Rust skulle brukes. Dette fordi oppdragsgiveren bruker Rust i sine produkter, og et av formålene med prosjektet er at det skal integreres i deres system.

3.2.2.2 MoonZoon

Fullstack Rust-rammeverket MoonZoon var også et krav fra oppdragsgiver. Det finnes andre frontend-rammeverk for Rust, men Ringrev har valgt å bruke fullstack-rammeverket MoonZoon i sine systemer.

3.2.2.3 ArangoDB

Oppgaven krevde at gruppen brukte databaseløsningen ArangoDB, da oppdragsgiver bruker denne i sine produkter.

3.2.2.4 Aragog
Dette biblioteket for kommunikasjon med ArangoDB ble anbefalt til gruppen av oppdragsgiver - men var ikke et krav å ta i bruk. Fordi gruppens prosjekt fortsetter som open source vil det være en fordel at det er enklest mulig å ta i bruk wikirammeverket. Fordelen med Aragog over alternative biblioteker er at man ikke trenger å skrive inn queryspråk. I tillegg har Aragog funksjoner for alle CRUD-operasjoner. Aragog var et av de få bibliotekene som fortsatt aktivt blir oppdatert. På grunn av disse fordelene tok gruppen i bruk Aragog. Alternativet som ble vurdert var Arangors som har liknende funksjonalitet for å koble til database og bruk av AQL, men var mindre oppdatert og dokumentasjonen var svakere i forhold til Aragog.

3.2.2.5 Firebase Auth
Gruppen valgte å bruke Firebase Auth til registrering og innlogging på grunn av sikkerheten det tilbyr for håndtering av brukerkontoer, og fordi det støtter JSON Web Tokens. For brukere som setter opp en wiki ved bruk av gruppens rammeverk i fremtiden vil Firebase Auth kreve lite interaksjon. De kan også enkelt sette opp verifiseringsepost og resetting av passord hvis de ønsker det. Firebase Auth ble hovedsakelig valgt grunnet kjennskap til systemet fra tidligere prosjekter. Andre alternativer som hadde løst dette på liknende måte er Auth0 og Amazon Cognito.

3.2.2.5.1 Fire Auth
Firebase har ikke et eget bibliotek for Rust, og gruppen ønsket derfor å finne et eksternt bibliotek for dette. Det går an å gjøre kall direkte til Firebase sin REST API, men gruppen ønsket en løsning som gjør det enkelt for fremtidige brukere av wikirammeverket å oppnå mer funksjonalitet enn hva wikien har implementert allerede. Biblioteket Fire Auth ble valgt fordi det har mange funksjoner for interaksjon med Firebase Auth, som blant annet registrering, innlogging, bytte av passord, sletting av bruker og verifisering via e-post. Fire Auth er et av få Rust-biblioteker som tilbyr støtte for Firebase sine autoriseringsfunksjoner, har god dokumentasjon, og i tillegg fortsatt blir oppdatert.

3.2.2.6 Jira og Confluence
Jira og Confluence ble valgt da det ble anbefalt av flere forelesere ved NTNU, og er ofte brukt som prosjektstyringsverktøy i utviklingsprosjekter.

3.2.2.7 Balsamiq Cloud
Balsamiq Cloud ble brukt til å lage konseptskisser. Verktøyet ble valgt fordi gruppen hadde tidligere erfaring med det, og det har god brukervennlighet. Det er enkelt å finne representasjoner av de vanligste elementene man forventer å finne på nettsider eller applikasjoner.

3.2.2.8 IntelliJ
Da gruppen skulle velge IDE, stod det i utgangspunktet mellom IntelliJ fra JetBrains og Visual Studio Code (VSCode) fra Microsoft. De har begge plugins som gir støtte for Rust. På grunn av tidligere erfaring med IntelliJ og dens gode integrasjon av versjonskontroll prøvde gruppen denne IDE-en først. Gruppen fikk opp feilmeldinger på MoonZoon-koden selv om applikasjon kjørte fint. På grunn av feilmeldingene i IntelliJ prøvde gruppen VSCode, hvor

disse feilmeldingene ikke oppstod. Av denne grunn lente gruppen mot å bruke VSCode, men endte opp å spørre oppdragsgiver hva de bruker.

Ringrev anbefalte gruppen å holde seg til en IDE fra JetBrains da deres erfaring er at det fungerer best til MoonZoon. Ringrev og gruppen fant ut at man måtte skru på alle «experimental features» relatert til Rust i IDE-en for å fjerne feilmeldingene. Etter at dette problemet ble løst fortsatte gruppen å bruke IntelliJ. Man kan også fint bruke JetBrains sin IDE CLion, så lenge man installerer samme plugins og bruker samme innstillingene for «experimental features» som i IntelliJ. Det går også fint å bruke VSCode om man ønsker det, med plugin rust-analyzer. Mer om dette i prosjektets readme-fil som er vedlagt prosjektets kildekode (vedlegg G).

3.2.2.9 Docker

Docker ble brukt til å bygge et image av ArangoDB. Gruppen valgte Docker fordi det ville dekke behovet til gruppen, og gruppen så på verktøyet som relevant å ha kunnskap om i arbeidslivet.

3.2.2.10 DigitalOcean

Skytjenesten DigitalOcean ble brukt til å hoste et Docker image av ArangoDB i en VM. Gruppen valgte DigitalOcean fordi det er et verktøy som gjør det enkelt å hoste et Docker image, og det er enkelt å gjøre forandringer mens man tester.

4 Resultater

I dette kapittelet vil det bli presentert hvilke vitenskapelige, ingeniørfaglige og administrative resultater som ble oppnådd gjennom dette prosjektet.

4.1 Vitenskapelige resultater

De vitenskapelige resultatene er basert på brukerstudier fra to forskjellige faser i prosjektet. Det ble utført 2 brukertester, med henholdsvis 4 og 2 deltakere. Disse resultatene er kvalitative data og blir ikke ansett som målbare. Testene er oppsummert i de påfølgende underkapitlene.

4.1.1 Brukertest av konseptskisser

Denne brukertesten ble utført i uke 6, mens gruppen var i planleggingsfasen av produktet. All tilbakemelding fra brukertesten ligger i filen «Brukertest konseptskisser» (vedlegg E). Deltakerne i brukertesten ble først spurt om deres forventninger til en wiki. Deretter ble de vist en og en konseptskisse hvor de skulle gi tilbakemelding på funksjonalitet og plassering av elementer.

4.1.1.1 Forventninger til en wiki

Deltakerne i brukertesten hadde blant annet disse forventningene til en wiki:

- Lett å navigere uten opplæring
- At det er mulig å bidra med informasjon
- Kvalitetssjekk av informasjon
- Preview av artikkel mens man redigerer
- Interaksjon og klikkbarhet på tags, bidragsytere og funksjonalitet for lenker mellom artikler

4.1.1.2 Forbedringer i prosjektet

Forbedringer som ble gjort i prosjektet basert på tilbakemeldingene fra denne testen:

- Gruppen fikk en bedre forståelse for hva som er forventet av en wiki
- Gruppen fikk mange ideer til funksjonalitet som kan legges til som issues på prosjektets GitHub
- Endring i plassering av knapper

4.1.1.3 Resultat av brukertest

Testerne fant lett frem i konseptskissene. Mens kjernefunksjonene var på plass, hadde de forskjellige forventninger som ikke var innfridd. For eksempel at de følte at enkelte elementer lå på litt feil sted – som at listen med bidragsytere lå over artikkelen på artikkelsiden i stedet for under.

4.1.2 Brukertest på MVP

Denne brukertesten ble utført i uke 17-19, mens gruppen var i slutten av utviklingsfasen. All tilbakemelding fra brukertesten ligger i filen «Brukertest MVP» (vedlegg E). Deltakerne i brukertesten ble spurt et sett med spørsmål og ble bedt om å navigere siden.

4.1.2.1 Problemer

Det ble oppdaget noen problemer med løsningen:

- Testeren trykte på søkefelt mens han leste en artikkel, og ble sendt til forsiden idet han skrev en bokstav. Da mistet søkefeltet fokus og han måtte trykke på det på nytt for å skrive videre.
- Testerne klikker to ganger på registrer, for det er ikke tydelig respons på om registreringsprosessen er i gang. De blir ikke registrert to ganger.
- Det går an å lage brukernavn i flere ord, der intensjonen er at det kun skal være ett ord.

4.1.2.2 Forbedringer i prosjektet

Følgende forbedringer har blitt gjort etter tilbakemeldingene fra denne testen:

- Man kommer ikke lenger til forsiden når man skriver i et søkefelt på en artikkelside. Nå går man til forsiden først når man trykker på «Enter» eller søkeknappen.
- De resterende problemene har blitt lagt til som issues på GitHub-en til prosjektet.

4.1.2.3 Resultat av brukertest

Testerne finner frem, og kjenner seg igjen i brukergrensesnittet. Gruppen fikk tilbakemeldinger som førte til forbedringer i prosjektet, samt at de fikk legge til et par gjøremål for fremtidige bidragsytere i form av issues på prosjektets GitHub.

4.2 Ingeniørfaglige resultater

4.2.1 Resultatmål

Resultatmålet som gruppen definerte i forprosjektplanen, bestod av å oppfylle kravene for en MVP. Oppdragsgiver godkjente gruppens forslag for funksjonalitet som skulle være inkludert i MVP.

Funksjonalitet	**Status**
Registrering og innlogging	Fullført
Lagring i database	Fullført
Se artikler i wiki	Fullført
Legge til artikler i wiki	Fullført
Redigere artikler i wiki	Fullført
Slette artikler i wiki	Fullført
Søke etter artikler	Fullført

Tabell 1: Status på resultatmål

Status for punktene i MVP er vist i tabell 1. Gruppen har altså fullført kravene som var definert for å oppnå MVP.

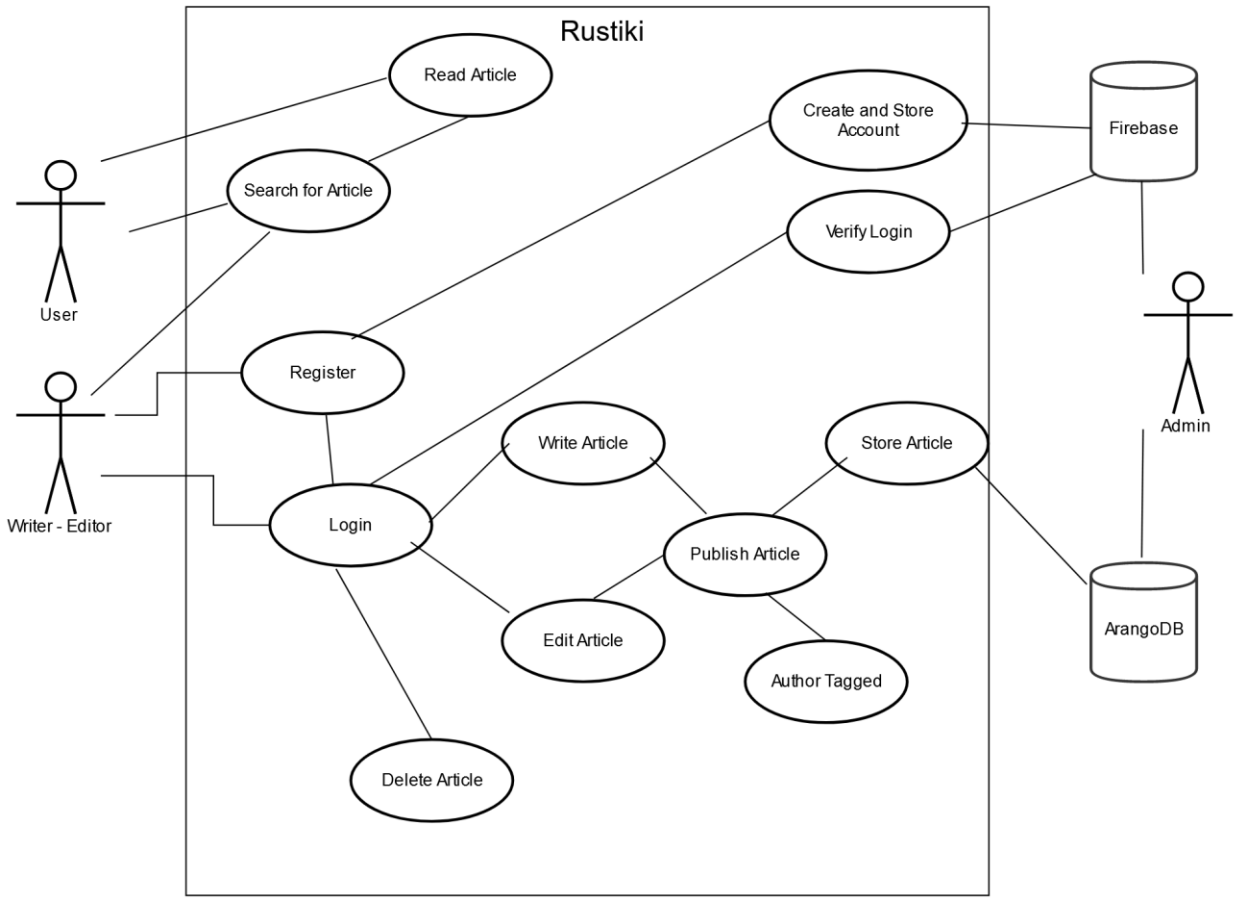
4.2.2 Effektmål

Effektmålet gruppen satte i forprosjektplanen (vedlegg A) var følgende:

> *Målet er å få en god øvelse i større prosjektarbeid som inneholder de viktigste fasene av utvikling av software. Dette innebærer praktisk bruk av agile metoder, planlegging, dokumentering, refleksjon og kommunikasjon med oppdragsgiver. Vi vil forbedre evnen vår til å ta et stort prosjekt for så å strukturert dele det opp i komponentene som kreves for å løse oppgaven.*
>
> *Prosjektet gir oss en innføring i Rust og vi får se hvordan WebAssembly gjør det mulig å bruke programmeringsspråk i frontend som normalt kun brukes i backend. Dette er kompetanse som kan bli verdifull i arbeidsmarkedet de neste årene, basert på trender i bransjen.*
>
> *Prosjektet skal også brukes til å vise til gjennomført arbeid som kan brukes som en demonstrasjon av ferdigheter når vi skal ut på arbeidsmarkedet*.

Gruppen har oppnådd dette målet.

4.2.3 Beskrivelse av produkt



Figur 1: Use case-diagram for Rustiki.

Som nevnt i tidligere i kapittel 4 (4.2.1 Resultatmål) så oppfyller produktet kravene som var definert i MVP. Produktet som gruppen leverer, er et wikirammeverk (2.1.3 Wikirammeverk) som Rust-utviklere kan bruke til å sette opp en egen wiki. Figur 1 viser et use case-diagram over tilgjengelige interaksjoner i Rustiki. For å vise figur 1 som vektorgrafikk, se mappen «Diagrammer» (vedlegg F). For en bruker som ikke er logget inn er det mulig å se en oversikt over alle artiklene, lese artikler og søke etter artikler. For en innlogget bruker er det i tillegg mulig å skrive nye artikler eller bidra på andres artikler. En innlogget bruker kan slette artikler hvis de selv er forfatter av artikkelen. Tags kan legges til eller slettes. For flere detaljer om hvilke operasjoner brukere kan utføre, og en beskrivelse av produktets sider og funksjonalitet, se punkt 2 og 3 i systemdokumentasjon (vedlegg C).

Prosjektet er open source (2.1.19 Open source) under MIT-lisens og ligger åpent på oppdragsgivers GitHub (Hageselle, et al., 2022). Gruppen har utviklet grunnmuren for programvaren i henhold til visjonen for prosjektet, og beholder skrivetilgang som utviklere. For privatpersoner og bedrifter som ønsker å benytte kildekoden til eget bruk er kildekoden fritt tilgjengelig i henhold til lisensieringen. Andre utviklere kan bidra til Rustiki dersom de oppretter en pull-request, og de kan også legge til nye issues med forslag om endringer og ny funksjonalitet. Det er mulig for andre utviklere å redigere kildekoden direkte dersom de blir lagt til som administratorer av Ringrev. Dette kan eksempelvis være en videreføring av prosjektet i form av en bacheloroppgave med formål om å videreutvikle Rustiki.

4.2.4 Designprinsipper

Normans designprinsipper (2.1.13) definerer et sett retningslinjer for design av brukergrensesnitt.

Prinsipp	**Bra**	**Mindre bra**
Synlighet	• Få knapper • Knapper og artikler reagerer på hover	• Kunne eventuelt hatt mer funksjonalitet tilgjengelig fra forsiden for innloggede brukere
Respons	• Beskjed under inputfeltene ved feil input i innlogging og registrering • Dialog som ber om bekreftelse ved «Cancel» og «Delete»	• Ingen beskjed ved tomt søkeresultat • Ingen laste-animasjon når det trykkes «Register», «Publish» eller «Log in»
Samhandling	• Knapper reagerer på hover og kan trykkes på • Inputfelt kan skrives i	• Inne artikkelen er ikke bidragsytere eller tagger klikkbare
Begrensninger	• Ser ikke knapp for å lage, redigere eller slette artikler hvis man er utlogget • Ser ikke knapp for registrering eller innlogging mens man er innlogget	• Brukere som ikke har tillatelse til å slette en artikkel får frem dialog når de trykker på sletteknappen, men sletteknappen kunne vært fjernet for denne typen bruker

Konsistens	• Samme layout går igjen på hver side. Den er lik på registrering og innlogging, og på sidene for å lese, redigere og lage artikler. • Samme header på alle sider • Knappene i header er like hverandre • Knappene på sidene for å lese, redigere og lage artikler er like hverandre • Knapper går igjen i samme rekkefølge • Kan trykke på logo for å komme til forsiden • Knapp for sletting av en tag dukker opp inne i taggen istedenfor utenfor • Knapp for å lage ny artikkel ligger ved siden av eksisterende artikler med et pluss-tegn istedenfor bilde	

Tabell 2: Oversikt over designprinsipper Rustiki oppnår bra og mindre bra.

Tabell 2 viser en oppsummert oversikt over hvilke designprinsipper produktet oppnår bra og mindre bra. Dette blir videre utdypet i kapittel 5 (5.9 Designprinsipper).

4.2.5 Universell utforming

I Norge må private virksomheter oppfylle visse punkter fra WCAG 2.0.

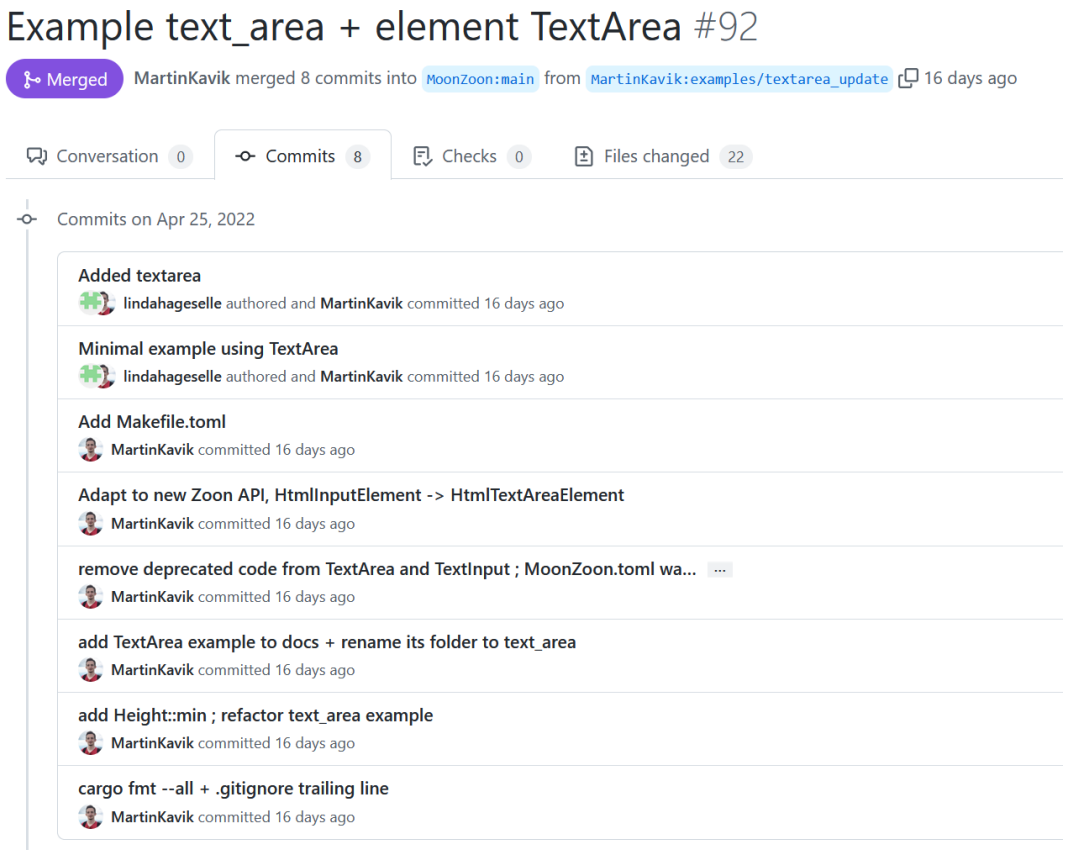
Kriterie	**Wikirammeverket oppfyller kriteriet**
1.1: Tekstlig alternativ	Ja
1.2: Alternativ til lyd og video	Ja
1.3: Kan tilpasses	Ja
1.4: Kan skilles mellom	Ja
2.1: Tastaturtilgjengelighet	Ja
2.2: Nok tid	Ja
2.3: Anfall	Ja
2.4: Navigerbart	Ja
3.1: Lesbart	Ja, men ikke automatisk
3.2: Forutsigbart	Ja
3.3: Instruksjoner for input	Ja
4.1: Kompatibelt	Ja

Tabell 3: Oversikt over Rustikis samsvar med WCAG 2.0.

Tabell 3 oppsummerer produktets oppfyllelse av de lovpålagte punktene fra WCAG 2.0 (2.1.14 Universell utforming). Dette blir videre utdypet i kapittel 5 (5.8 Universell utforming).

4.2.6 Bidrag til rammeverket MoonZoon

Prosjektet har ført til fremgang i fullstack-rammeverket MoonZoon. Denne fremgangen i rammeverket består av det gruppen selv har bidratt med i rammeverkets kode, og endringer gjort av Ringrev og hovedutvikleren av MoonZoon. De endringene ble gjort etter at gruppen pekte ut mangler ved rammeverket og hvilke behov gruppen hadde som utviklere.

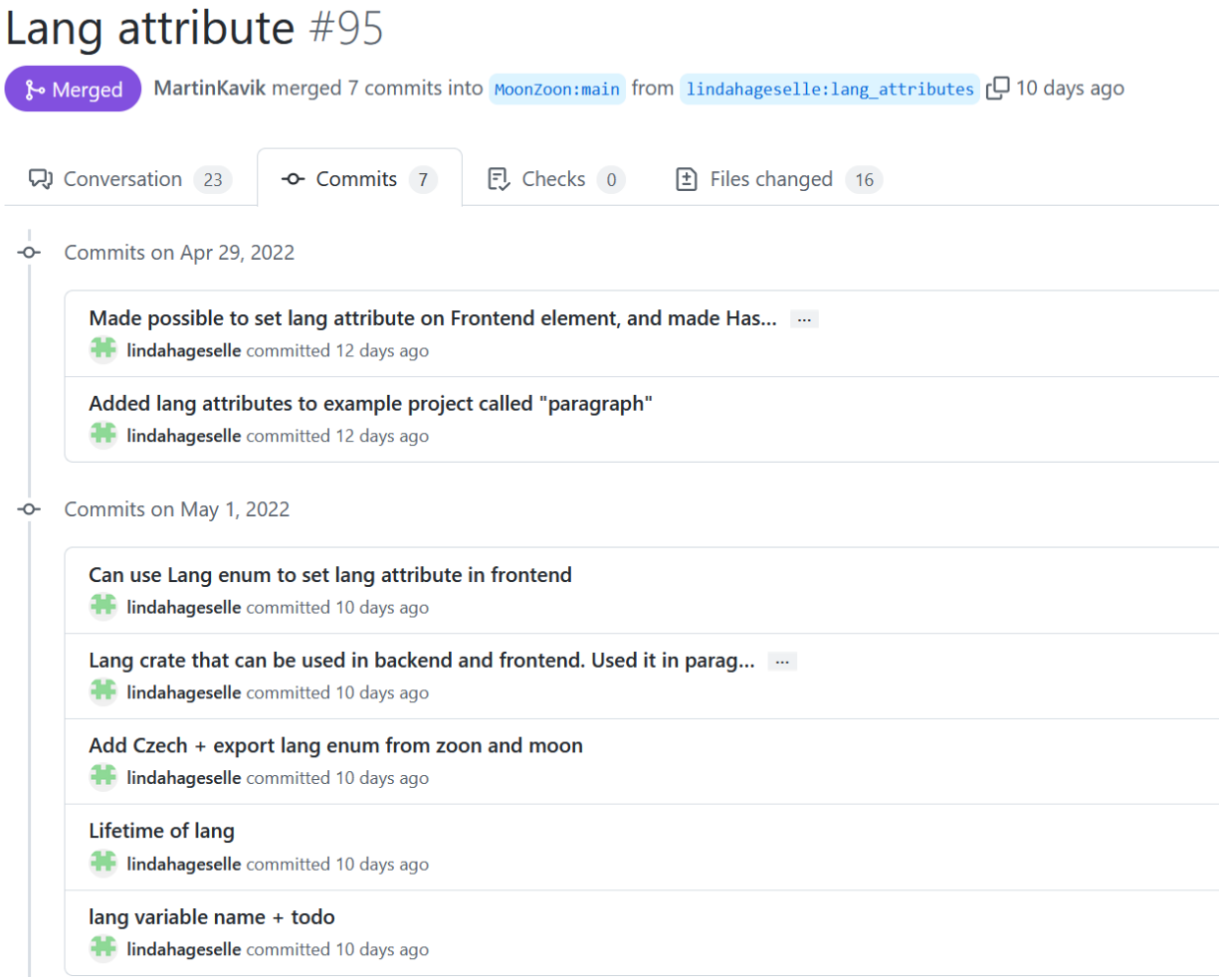


Figur 2: Oversikt over commits før koden ble merget inn i MoonZoon.

Gruppen så at det fantes et behov for en komponent i MoonZoon som støttet input-felt for flere linjer tekst. Ringrev oppfordret gruppen til å opprette en pull-request for å lage denne komponenten selv. Etter at gruppen hadde laget denne komponenten sa hovedutvikleren av MoonZoon at gruppen burde lage et eksempelprosjekt for å kunne teste at komponenten fungerer som forventet. Gruppen lagde deretter et slikt eksempelprosjekt som en pull-request til MoonZoon. På grunn av et par store forandringer i MoonZoon rett før gruppen bidro med eksempelprosjektet, bestemte hovedutvikleren av MoonZoon seg for å merge inn gruppens kode selv, og gjøre nødvendige endringer for å tilpasse den til den nyeste koden i MoonZoon. Figur 2 viser en oversikt over commits som ble gjort av gruppen og MoonZoons hovedutvikler før koden ble merget inn i rammeverket.

Etter gruppen hadde laget komponenten som støttet input-felt for flere linjer tekst, ble det klart at Ringrev etter hvert ønsker muligheten for flere avanserte alternativer for å tilpasse tekst og bilder i en wiki. Selv om komponenten laget av gruppen er nødvendig i MoonZoon og oppnår hva som trengs for gruppens MVP, er det ikke tilstrekkelig for behovet brukerne av rammeverket vil ha i fremtiden.

CTO ved Ringrev fant ut at å ta i bruk JavaScript for denne funksjonaliteten ville være mer produktivt enn andre potensielle løsninger, da det allerede finnes løsninger som kan brukes og tilpasses. Han laget en pull-request der han jobbet med integrasjon av JavaScript-kode i MoonZoon. Dette førte til at Ringrevs CTO og hovedutvikleren bak MoonZoon samarbeidet om et prosjekt som skulle gjøre det enklere for MoonZoon-utviklere å bruke JavaScript-kode i sine prosjekter.



Figur 3: Oversikt over commits før koden ble merget inn i MoonZoon.

Gruppen oppdaget at MoonZoon ikke støttet kriteriet om lesbarhet fra WCAG 2.0. Som nevnt i kapittel 2 (2.1.14 Universell utforming) handler dette om at man kan spesifisere språkattributter på både selve nettsiden og på tekster ellers på nettsiden. Siden dette er et lovpålagt krav lagde gruppen en løsning for dette selv, som deretter ble gjort om til en pull-request i MoonZoon. Gruppen fikk noen forslag til forbedring fra hovedutvikleren av

MoonZoon, utførte disse endringene, og så ble løsningen merget inn i rammeverket av hovedutvikleren. Figur 3 viser commits gjort før koden ble merget inn i MoonZoon. En av endringene MoonZoons hovedutvikler bar om, var at enum-et som definerer forskjellige språkattributter ble gjort om til et eget bibliotek. Dette fordi han mente at dette biblioteket kunne være nyttig for Rust-utviklere generelt – ikke bare utviklere som bruker MoonZoon.

4.2.7 Docker og DigitalOcean

ArangoDB støtter de fleste typer arkitektur, men ikke nye Apple M1. På grunn av dette valgte et gruppemedlem å kjøre ArangoDB i et Docker image. Fordi et Docker image som kjører er knyttet til host-maskinens maskinvare fungerte dette dårlig. Dette ble løst ved å flytte Docker imaget til skyen. Tjenesten som ble valgt grunnet lave driftskostnader og bonuser for nye brukere på kjøretid, var DigitalOcean. Her ble Docker imaget kjørt på en VM med Ubuntu som operativsystem og tradisjonell fastvare for en server. Denne løsningen fungerte godt og stabilt og ble derfor valgt som gruppens felles database under prosjektet. Alternativet som var planlagt som Host for databasen tidligere var NTNUs OpenStack løsning. Gruppen vurderte begge løsninger som gode, men valgte DigitalOcean for å få bedre kjennskap med skytjenester som kan bli aktuelle i arbeidslivet.

Selv om gruppens prosjekt ikke innebærer å deploye en løsning, hadde gruppen i starten av prosjektet ønske om å kunne utforske løsninger for kontinuerlig integrasjon. Dette viste seg å være både komplekst og tidkrevende på grunn av et begrenset utvalg av pipeline-verktøyer med støtte for Rust. I tillegg er ikke rammeverket MoonZoon ment for standalone applikasjoner enda, da prioriteten til rammeverkets hovedutvikler nå er å få opp en egen plattform - MoonZoon Cloud - for hosting av MoonZoon-applikasjoner. Etter hvert planlegger han å legge til støtte for standalone applikasjoner hostet andre plasser enn MoonZoon Cloud (Kavík, 2022). På grunn av nevnte faktorer ble ikke kontinuerlig integrasjon brukt i gruppens prosjekt.

Gruppen har likevel forberedt en Dockerfile til prosjektet, slik at den er tilgjengelig når MoonZoon i fremtiden er tilgjengelig for hosting gjennom en selvvalgt hosting-tjeneste. Denne filen er basert på en Dockerfile i et MoonZoon-prosjekt på Ringrevs GitHub, som var laget av MoonZoons hovedutvikler. Gruppen modifiserte denne Dockerfile-en, og fikk etter noen iterasjoner av filen til å deploye imaget i en container (2.1.18 Container images) på DigitalOcean. I Docker imaget oppstod det problemer i kommunikasjonen mellom frontend og backend, der frontend ikke mottar responsen fra backend. Dette resulterte i at man ser applikasjonens frontend, men all logikk som avhenger av respons fra backend er utilgjengelig – det eneste man får gjøre er å bla mellom forsiden, registeringssiden og innloggingssiden. Applikasjonen ble bare deployet for testing, og gruppen tok ned nettsiden etterpå. Dockerfile-en ligger på en egen branch på Rustikis GitHub til fremtidig bruk. Kort beskrivelse av filen og utfordringer med deployment beskrives i Rustikis readme-fil som er vedlagt prosjektets kildekode (vedlegg G).

4.3 Administrative resultater

Dette delkapittelet presenterer de administrative resultatene oppnådd gjennom prosjektet.

4.3.1 Prosess

Gruppens valg av agil metodologi i prosjektet innebar sprinter som gikk over to uker, med et fysisk møte med oppdragsgiver og veileder etter endt sprint. Resultatet av de fysiske møtene ble en god dialog mellom partene, og lavere terskel for å komme med tanker og innspill enn hva gruppen har erfart ved digitale møter. Kombinasjonen av fysiske møter og oppdragsgivers kompetanse innen Rust og MoonZoon ga rom for klarerende diskusjoner om produktet, og mulighet for at gruppen fikk stille spørsmål om teknologien som ble brukt. Dette var en fordel siden gruppen arbeidet med ny teknologi som har få ressurser. Møtene med oppdragsgiver resulterte også i rask tilbakemelding og mulighet for hyppige iterasjoner under utviklingen, i henhold til agile metoder.

Gruppen valgte å hovedsakelig arbeide fysisk samlet på campus. Dette førte til kontinuerlig deling av kunnskap, og diskusjoner av ideer og tanker rundt prosjektet. Som resultat av denne arbeidsmetoden fikk alle gruppens medlemmer kjennskap til hverandres arbeid, som ga alle en oversikt over samlet progresjon. Gruppetilhørigheten som kom av disse valgene resulterte i bedre utnyttelse av agile prinsipper, spesielt med kort vei for å stille spørsmål og å diskutere funksjonalitet og designvalg på prosjektet og dermed raskt iterere med ny kunnskap. Gruppen besluttet å ikke følge daglige morgenmøter fra Scrum, men valgte å tilpasse dette ved å ha mulighet for spontane møter ved behov. De valgte administrative metodene var sentrale for gruppens oppnåelse av de ingeniørfaglige resultatene beskrevet i kapittel 4.2 Ingeniørfaglige resultater.

4.3.2 Oversikt over tidsbruk

Tidsplanen har blitt nøye fulgt. Dette resulterte i 68 arbeidsdager over 14 sprinter. Gruppen hadde planlagt åtte møter med oppdragsgiver og veileder. Alle møtene ble gjennomført som planlagt. Se gruppens prosjekhåndbok (vedlegg D) for en mer detaljert beskrivelse av planer og gjennomføring av prosjektet.

5 Diskusjon

I dette kapitlet drøftes resultatet av oppgaven. Dette omfatter hvordan deler av oppgaven ble løst i forhold til gruppens forutsetninger, hvordan de utfordringene kunne vært løst annerledes og videre forbedringer på valg som ble tatt. Dette kan være nyttig for videre arbeid på Rustiki, fordi det innebærer gruppens refleksjoner gjennom utviklingen og erfaringer fra hele prosjektet.

5.1 Drøfting av resultatet

Gruppen er tilfreds med at Rustiki leverer funksjonalitet som oversteg kravet for MVP. For det første ble løsningen for roller som forfatter og bidragsyter på en artikkel løst ved automatisk markering av artikkelen. Dette var planlagt som en String som måtte legges inn manuelt av brukeren i MVP. For tekniske detaljer for denne løsningen se punkt 3.6 i systemdokumentasjon (vedlegg C).

For det andre er systemet levert med en høyere grad av universell utforming i forhold til hva gruppen forventet var mulig med et så ungt rammeverk som MoonZoon. Dette kom som et resultat av en systematisk gjennomgang av eksempelprosjektene i rammeverket, der gruppen samlet og tilpasset funksjonalitet som kunne brukes til universell utforming, for så å implementere funksjonalitet for universell utforming i Rustiki.

For det tredje benytter systemer i stor grad funksjonalitet fra MoonZoon. Gruppen er spesielt tilfreds med å ha implementert MoonZoons UpMsg og DownMsg for kommunikasjon mellom frontend og backend. UpMsg og DownMsg er MoonZoons egen abstraksjon av tradisjonelle request og response.

5.2 Kodestruktur

Moduler er inndelt etter hvilke logiske oppgaver de skal løse, og prosjektets kode har på denne måten en høy grad av kohesjon (2.1.21 Kobling og kohesjon). I backend er det egne moduler for å hente objekter fra database, lage nye objekter i database, slette objekter fra database, innlogging og registrering. Det er i tillegg laget separate moduler for hvert av objektene som blir gjenbrukt i backend. I frontend er det egne moduler for hver av sidene til applikasjon, der koden en praktisk inndelt i en modul for logikken til siden og en modul til det visuelle på siden. Koden oppnår en ganske lav grad av kobling, der kun nødvendige moduler importeres inn i en annen modul. For å redusere kodeduplisering, har gruppen i frontend opprettet moduler for visuelle elementer som kan brukes fra andre moduler.

5.3 Avansert tekstredigering

Rammeverket mangler et avansert tekstredigeringsverktøy. Dette var ikke en del av kravet til MVP, men er likevel en forventning personer har til en wiki. Grunnen til at et tekstredigeringsverktøy ikke var et krav er at Ringrev mente det ble vanskelig å bruke Rust-biblioteker for dette, men at det kanskje etter hvert kunne integreres en JavaScript-løsning

i prosjektet. Ringrev så på dette sammen med hovedutvikleren bak MoonZoon, da integrasjon av JavaScript ville være nyttig for andre som bruker rammeverket. I slutten av mars kom tekstredigeringsverktøyet Quill.js som et eget eksempelprosjekt i MoonZoon. Det støtter per nå tekst med bold, italic og underline, samt linker og lister. Det har blitt lagd en egen branch i prosjektet hvor Quill.js er implementert slik dette ligger klart for å bli utvidet senere. Gruppen har lagt til to issues på prosjektets GitHub som må løses før Rustiki kan bytte fra å bruke gruppens selvlagde komponent TextArea til å bruke Quill.js.

5.4 Registrering og innlogging

Oppdragsgiver ønsket at en løsning for registrering og innlogging skulle være på plass, men presiserte at det ikke var noe gruppen trengte å bruke mye tid på å implementere. Ringrev foreslo derfor at løsningen de hadde på sin nettside kunne bli tatt i bruk slik at gruppen skulle slippe å utvikle en løsning fra bunnen av. Det ble tydelig tidlig i denne prosessen at å ta i bruk løsningen til Ringrev ville være omfattende, siden den inneholdt mye ekstra funksjonalitet som ikke trengtes i wiki-løsningen. På grunn av kodens koblinger var det vært utfordrende å få fjernet det som var unødvendig, men fortsatt oppnå ønsket funksjonalitet. I stedet for å implementere Ringrevs løsning, valgte gruppen å bruke Firebase.

5.5 Routing

Som bruker av en wiki er det forventet at man har mulighet for å dele en lenke til en artikkel. Dette er foreløpig ikke støttet av Rustiki grunnet implementasjonen av routing i systemet. Gruppen har opprettet en issue på prosjektets GitHub for å synliggjøre denne mangelen.

5.6 Tokens

Et annet forbedringsområde er løsningen for tokens. JWT blir lagret i frontend knyttet til brukeren, som var en løsning basert på eksisterende løsninger i MoonZoon repositoriet. Gruppen ønsker at dette blir løst ved å lagre JWT i en cookie i brukerens nettleser, fordi dette er ansett som en av de bedre løsningen for tokens (Copes, 2021). Gruppen var kritisk til flere aspekter med sikkerheten (2.1.20 Sikkerhet) av produktet som beskrives i punkt 9 av systemdokumentasjon (vedlegg C). Som nevnt der er mange av gruppens bekymringer planlagte forbedringer i MoonZoon.

5.7 Prosess

Gruppen brukte i starten av prosjektet interne sprints på én uke, selv om sprintmøte var annenhver uke. Dette fungerte fint de ukene gruppen fikk lagt inn fem dager med arbeid, og da kunne se en god fremgang siden forrige sprint. Dette fungerte derimot dårlig de ukene gruppen hadde stor arbeidsmengde ved siden av bachelorprosjektet. Enkelte uker fikk gruppen kun lagt inn to dager i bachelorprosjektet, og fikk da ikke sett særlig fremgang fra forrige sprint. De ukene gruppen visste ville bli korte, kunne sprinten vært forlenget til to uker. Etter noen svært korte sprinter endte gruppen opp med å begynne å tilpasse sprintlengden etter hvor mange arbeidsdager de hadde tilgjengelig i tidsperioden.

I starten av prosjektet brukte gruppen sprints på en dårlig måte der de lagde sprints i Jira uten å starte de, for så å starte de den dagen de skulle avsluttes. I tillegg brukte ikke

gruppen story points. Gruppen forbedret dette ved utgangen av januar, og etter dette har alle sprints en burndown-chart som ikke er flat. Se gruppens prosjekthåndbok (vedlegg D) for flere detaljer om gruppens gjennomføring av sprints.

Da gruppen utførte brukertesting av konseptskisser fikk de mest tilbakemeldinger på forventninger og ønsket ekstra funksjonalitet. Brukertesten kunne blitt forbedret ved å utforme spørsmålene slik at testen resulterte i målbare data. Dette kunne blitt oppnådd ved å stille mer konkrete spørsmål. Der gruppen spurte om forventninger og ønsker, kunne de heller lagt fokus på forbedring av fremvist funksjonalitet og plasseringen av elementer. Gruppens brukertester ligger i mappen «Brukertester» (vedlegg E).

Som nevnt under i kapittel 3 (3.2.1.1 Agile prinsipper for utvikling), utformet gruppen personas og user stories for Rustiki. Gruppens user stories ble ikke brukt til å lage stories i Jira, men fungerte som et verktøy for idémyldring over wikirammeverket skulle brukes av, hvordan de ville bruke det, og hva som måtte implementeres for å innfri brukernes behov. Gruppens personas og user stories ligger i kravdokumentasjon (vedlegg B).

5.8 Universell utforming

Som nevnt i teorikapittelet (2.1.14 Universell utforming) må man som virksomhet i Norge i dag møte standarden for WCAG 2.0, og offentlig sektor må møte WCAG 2.1 innen februar 2023. Offentlig sektor inngår ikke i målgruppen til wikirammeverket. Virksomheter i den sektoren har etablerte plattformer de bruker, og ved behov for en ny plattform vil den velges med vekt på at løsningen møter de lovene og retningslinjene sektoren må forholde seg til.

Målgruppen til wikirammeverket består av utviklere som bruker Rust. Hvis en privat norsk virksomhet som bruker Rust eller vil begynne å bruke Rust trenger en wiki, er gruppens wikirammeverk et godt utgangspunkt. Denne virksomheten er underlagt WCAG 2.0

Wikirammeverket sett i kontekst av kriteriene i WCAG 2.0 som forklart i kapittel 2 (2.1.8 Universell utforming):

1.1: Tekstlig alternativ
Produktet oppfyller dette kriteriet ved å ha tekstlige alternativer til innholdet som ikke er tekst. Dette innebærer blant annet at alle elementer som kan interageres med har en ID som kan oppdages av skjermlesere. Siden MoonZoon er bygget på HTML, er det også umulig å legge til bilder uten tekstbeskrivelse. Bruk av bilder og dets alternativer blir opp til brukeren, siden man i teorien kan sette en tom string som bildetekst.

1.2: Alternativ til lyd og video
Dette kriteriet er ikke relevant for produktet i skrivende stund, siden wikirammeverket hverken har lyd eller video. Ansvaret for innholdet på en side som bruker wikirammeverket ligger på avsender; de må selv sørge for å enten ha en alternativ tekst, eller teksting av lyd i videoen.

1.3: Kan tilpasses
Dette kriteriet oppfylles da innholdet i koden er plassert i den rekkefølgen det skal presenteres for bruker. I tillegg er det ingen funksjonalitet som presenteres kun med én visuell markør. Knappen for å slette tags når man redigerer eller lager en artikkel har ikke

tekstlig instruksjon, men oppnår kriteriet ved å presentere tre visuelle markører; x som et symbol for sletting, x forandrer farge når man holder musepeker over, og knappen er plassert på selve taggen den gjelder.

1.4: Kan skilles mellom
Produktet oppnår dette kriteriet ved å ha god kontrast mellom tekst og bakgrunn, og ved at siden kan forstørres uten at funksjonalitet eller innhold går tapt. Det er heller ingen handlinger i wikien som kommuniseres kun ved bruk av farge.

2.1: Tastaturtilgjengelighet
All funksjonalitet i wikirammeverket kan brukes fra et tastatur. For å muliggjøre dette for artiklene på forsiden måtte de pakkes inn i et Button-element, da MoonZoon ikke har mulighet for å gjøre rader eller kolonner fokuserbare.

2.2: Nok tid
Dette kriteriet er oppfylt for wikirammeverket siden det ikke finnes elementer i wikien som går på tid. Eksempelvis må bruker selv lukke dialogboksene som blir brukt for å gi tilbakemelding til dem.

2.3: Anfall
Produktet har ikke innhold som kan utløse epileptiske anfall. Ansvaret for å ikke laste opp innhold som kan gi anfall faller på brukerne av rammeverket.

2.4: Navigerbart
Produktet oppfyller dette kriteriet ved at elementene kan fokuseres i en logisk rekkefølge, og alt som kan klikkes på med musepeker kan også brukes via tastatur. Overskriftene er forklarende, som f.eks «Create new article» og «Log in», og alle linker har tekst. Det er også mulig for en bruker å gjøre fokusindikatoren synlig. Dette kan gjøres direkte fra nettleser. I Chrome kan man aktivere fokusindikator ved å gå inn på Settings -> Accessibility, og aktivere valget «Show a quick highlight on the focused object».

3.1: Lesbart
Gruppen oppdaget at MoonZoon ikke hadde muligheten til å definere språket til hverken selve nettsiden eller teksten på siden. For å gjøre dette mulig ble det opprettet en pull-request til Moonzoon rammeverket.

Denne pull-requesten har blitt merget inn i MoonZoon og tillater at man kan legge til lang-attributt på hele nettsiden, og på paragrafer. Gruppen har satt språket til teksten for artikler til engelsk, men det er enkelt å bytte ut English med Norwegian i koden hvis man skal ha norsk tekst. I skrivende stund oppfylles kriteriet 3.1 bare dersom innholdet er på engelsk. Dette er noe som utviklere som tar i bruk rammeverket må forandre selv.

Det er også en mulighet å legge til en språk-knapp på siden hvor man lager artikler, i tilfelle noen som bruker wikien ønsker å ha artikler på forskjellige språk. I det tilfellet må man legge til et nytt felt for språk i databaseobjektet for artikler.

3.2: Forutsigbart
Wikirammeverket oppfyller dette kriteriet fordi konteksten ikke forandres når man fokuserer på et element eller gir input til et inputfelt. Knapper som eksisterer på flere sider, har

samme plassering på de forskjellige sidene. Eksempelvis ser siden for å lage ny artikkel og siden for å redigere artikkel like ut, med knappene på samme plass.

3.3 Instruksjoner for input
Alle inputfelt har en placeholder-tekst for å fortelle brukeren hva som skal skrives. Hvis man forsøker å bruke en epost eller et brukernavn som allerede er i bruk, får bruker beskjed om at det ikke er tilgjengelig. Dersom bruker forsøker å logge inn med feil epost eller passord, får de beskjed om at input er feil og at de må prøve igjen.

4.1 Kompatibelt
For at wikirammeverket skal være fungerende for flest mulig brukere, er det satt id eller label på alle elementer som kan interageres med. Aria-rollene for elementer som knapper og inputfelt er satt av MoonZoon, som betyr at for eksempel skjermlesere kan lettere kjenne igjen elementene.

5.9 Designprinsipper

Her vil wikirammeverket bli sett i kontekst av Don Normans designprinsipper som nevnt i teorikapittelet (2.1.13 Normans Designprinsipper):

Synlighet
Wikirammeverket er oversiktlig og har få knapper. En bruker som ikke er innlogget ser enkelt hvordan man leser eller søker etter artikler. En innlogget bruker har i tillegg tilgang til å lage en ny artikkel på forsiden, og tilgang til å redigere og slette en artikkel når de er inne på den.

Synligheten til funksjonene redigering og sletting kunne vært forbedret ved å legge til knapper for de på hver artikkel på forsiden. Grunnen til at dette ikke ble gjort var for at forsiden skulle se ryddigere ut, og ikke overvelde bruker med valg.

Respons
Eksempler på god respons i wikirammeverket er at brukeren får respons ved feil input når de registrerer seg eller logger inn, i form av tekst under inputfeltene. De får respons med en dialog hvis de prøver å legge til en tag som fins fra før. Responsen kunne vært forbedret ved å gi brukeren skriftlig beskjed når de registrerer seg, logger inn og hvis de publiserer, oppdaterer eller sletter en artikkel. I alle disse tilfellene blir bruker sendt til forsiden, uten direkte respons. Brukeren får likevel en indirekte respons. Ved registrering og innlogging blir bruker sendt til forsiden der brukernavnet deres nå vil stå oppe i høyre hjørnet ved siden av knappen for utlogging. Når en bruker publiserer eller redigerer en artikkel, vil nevnte artikkel være den første i listen av artikler når de blir sendt til forsiden. På samme måte vil artikkelen man slettet ha forsvunnet fra listen idet man blir sendt tilbake til forsiden.

Responsen kan også forbedres når ting blir lastet inn, som for eksempel ved registrering og innlogging. Disse operasjonene bruker tilstrekkelig med millisekunder til at bruker gjerne klikker to ganger. Man vil ikke bli registrert to ganger selv om man klikker to ganger, men på andre klikket vil det komme en beskjed om at eposten allerede er tatt - rett før du blir sendt videre til forsiden og logget inn. Der burde ligge en animasjon med betydningen

«lasting pågår» som dukker opp med en gang første klikk blir registrert, slik at bruker ser at siden jobber.

Samhandling
Et eksempel på god samhandling i wikien, er at knapper kan trykkes på og inputfelt kan skrives i. Når man holder musepeker over en artikkel på forsiden forandrer bakgrunnsfargen seg på den artikkelen, som forteller bruker at de kan interagere med komponenten.

Samhandlingen kunne vært forbedret på label-ene som viser forfatter, bidragsyter og tags når du leser en artikkel. Måten en label er utformet på gir brukeren forventning om at de kan klikke på label-en for å søke på frasen den inneholder.

Mapping
Det finnes ingen eksempler på bruk av tradisjonell mapping i wikirammeverket, men heller ingen brudd på mapping.

Begrensninger
Gode eksempler på begrensninger i wikirammeverket er at bruker ikke kan se knapper for å redigere eller slette artikler mens de er logget ut, og at de ikke ser knapper for registrering og innlogging når man er logget inn. Disse begrensningene bidrar til at brukeren ikke blir forvirret, eller får tilgang til funksjonalitet de ikke skal ha tilgang til.

Brukergrensesnittet til produktet kunne vært forbedret når en innlogget bruker enten leser eller redigerer en artikkel. Hvis brukeren som trykker på sletteknappen til en artikkel ikke er forfatteren av artikkelen, får de opp en beskjed om at kun forfatter har lov å slette den. Istedenfor å gi denne beskjeden burde sletteknappen vært utilgjengelig for de som ikke er forfatter. Dette kunne blitt gjennomført på nesten samme måte som wikien bruker for å vise riktige knapper alt ettersom bruker er innlogget eller ikke.

Konsistens
Eksempler på god intern konsistens i wikirammeverket er at de forskjellige sidene har lik eller tilnærmet lik layout, og at knapper for å redigere, slette, publisere eller kansellere alltid ligger nederst på siden. Alle knapper i header er like hverandre. I tillegg er alle knapper for å forandre artikler like hverandre. Knapper som forekommer på flere sider er i samme rekkefølge. Siden for å lage nye artikler og for å redigere artikler er tilnærmet identiske, som gjør at bruker ikke trenger å forholde seg til forvirrende ulikheter.

Et eksempel på god ekstern konsistens er at man kommer til forsiden ved å trykke på logoen. Et annet eksempel på ekstern konsistens i wikirammeverket er at når du legger til en ny tag i en artikkel, så vil det ligge en rød x inne i samme label som taggen – dette gjør det enklere å vite hvilken tag en sletteknapp tilhører når der ligger flere tags ved siden av hverandre. Siden bokstaven x eller et kryss ofte er brukt for å fjerne noe og rød ofte er brukt til sletteknapper, vil bruker anta at man kan trykke på denne for å slette den aktuelle taggen. Et tredje eksempel på ekstern konsistens er at når bruker er logget inn, får de frem en komponent som ligger fremst i artikkel-listen, som man kan trykke på for å lage en ny artikkel. Dette er ment å skape assosiasjoner til en artikkel, men det står «create new article» og har et plusstegn istedenfor bilder for å indikere at knappen ikke sender deg til en ferdig artikkel.

5.10 GDPR

For en uregistrert bruker er det i MVP versjonen av Rustiki mulig å se og lese artikler på nettsiden uten å oppgi personalia eller bli utsatt for cookies.

For å registrere en bruker kreves e-post og passord. For Rustikis MVP håndteres dette via Firebase. Her lagres e-post i synlig tekst for administrator. Passord blir hashet og denne hashen er ikke synlig for administrator. Det genereres også en ID som vi bruker til å knytte brukerobjektet i ArangoDB-databasen til autentisering av brukeren i Firebase. Firebase gjør det mulig å tilby en høy grad av sikkerhet for passordhåndtering i samsvar med GDPR (2.1.15 GDPR) uten mye implementasjon.

En registrert bruker kan ikke slette sin egen konto etter at den er opprettet. Dette gjennomføres av den som har administratortilgang til den aktuelle løsningens Firebase. I henhold til GDPR har en bruker retten til å bli slettet og glemt av bedriften som bruker informasjonen (Proton Technologies, 2022). Ved bruk internt i en bedrift er den nåværende håndteringen av brukerkontoer i samsvar med GDPR. Dette fordi en bruker kan be organisasjonen om slette informasjonen, og dette er mulig å gjennomføre for en administrator. Dersom den som ber om å slettes har et identifiserende brukernavn, må man i henhold til GDPR også fjerne spor av dette brukernavnet fra siden hvis den aktuelle brukeren ønsker det. Dette kan gjøres manuelt i ArangoDB av en administrator.

Selv om det er mulig for en administrator å slette en brukerkonto, kan det være ønskelig å legge til en funksjon for at bruker får slette egen konto, og eventuelt samtidig får brukernavnet sitt fjernet fra artikler de har skrevet eller bidratt på. Dette er en foreslått issue på prosjektets GitHub. Siden biblioteket Fire Auth per nå ikke har en funksjon for å slette en brukerprofil, kan det lages en egen funksjon for dette. Man kan slette en brukerkonto fra Firebase ved å sende en POST-request til Firebase Auth sitt endpoint for sletting av kontoer. Et endpoint er den URL-en vi sender requesten til for å få utføre en operasjon. Etter kontoen i Firebase er slettet, kan man ha en funksjon som fjerner bruker fra ArangoDB-databasen, og til slutt en funksjon som fjerner brukernavnet fra artikler.

For bedrifter som skal bruke Rustiki med Firebase blir det viktig å inkludere brukervilkårene for Firebase i bedriftens brukervilkår dokumentasjon. Dette er inkludert i dokumentasjonen for Rustiki i prosjektets readme-fil.

5.11 Teknologi

5.11.1 Database

Valget av database ble gjort av oppdragsgiver, som nevnt i kapittel 3 (3.2.2.3 ArangoDB). Gruppen hadde ikke kjennskap til NoSQL (2.1.12 NoSQL database) databaser fra før. ArangoDB ble oppfattet som et godt verktøy, da det er brukervennlig og har en god løsning for administrasjon og overvåkning av databasen. Likevel støtte gruppen på et par utfordringer.

Den første utfordringen var koblingen mellom Rust og ArangoDB. Gruppen ønsket en mellomvare som håndterte AQL, query-språket til ArangoDB, fordi det er enklere for fremtidige brukere å forandre på enn queries i form av en String. Det fantes få alternativer

på grunn av lite oppdaterte bibliotekfiler for de aktuelle mellomvarene. Valget falt på Aragog (2.2.1.3 Aragog), fordi det er aktivt utviklet og dekker behovet for dette prosjektet.

Den andre utfordringen var manglende støtte for Apple M1 ARM arkitektur. Som beskrevet i kapittel 4.2.7 Docker og DigitalOcean ble dette løst ved å flytte databasen til en skytjeneste via et Docker image. Mangelen på støtte for ARM oppfattes ikke som en stor mangel med ArangoDB, da det ikke er vanlig å kjøre databasesystemet på denne arkitekturen ennå. Det rammer kun utviklere som må kjøre databasen lokalt på nyere Mac-maskiner.

Det ble også oppdaget problemer med å benytte flere schema-filer som automatisk skapes av Aragog. Dette kan bli et problem for brukere som ønsker å benytte flere databaser i Rustiki. En løsning er at det er mulig å skrive om kildekoden i Rustiki til å benytte andre mellomvarer med støtte for flere databaser.

Med prosjektets mål om å ende opp som et open source prosjekt, mener gruppen at valget av database og mellomvare var fornuftig for oppgaven og fremtidige brukere. Begge teknologiene er godt dokumentert og oppdateres jevnlig i tidsrommet denne oppgaven ble gjennomført.

5.11.2 Søk

Ved levering av wikirammeverket gjøres alle søkene i applikasjonens frontend. Dette gjøres ved at wikien laster alle artikler når applikasjonen blir lastet inn, og et frontend-søk vil da finne alle de eksisterende artiklene som matcher brukerens søkefrase. Gruppen vurderte dette som en god løsning for en wiki som ikke har store mengder brukere eller artikler. Applikasjonen henter innholdet i databasen på nytt hvis bruker publiserer, redigerer eller sletter en artikkel.

På en annen side vil denne løsningen resultere i behov for endringer i wikier som har et stort antall brukere og innhold. I slike tilfeller ønsker man ikke å laste inn alle artiklene fra databasen til frontend når man laster inn nettsiden. Da vil det bli behov for en funksjon som søker etter artikler i databasen heller enn å søke i en vektor i frontend. Dette kan løses ved å lage en ny handler for slik funksjonalitet i backend og å sende input fra søkefeltet til denne handler-en. Dette vil kunne implementeres i lignende stil som den handler-en som oppdaterer artikler etter redigering.

Det er også et spørsmål om hvor mye trafikk til databasen brukeren ser på som akseptabelt. Dette kan variere i stor grad med kapasitet på server, og kan medføre økte kostnader som ikke er ønsket. Søk i database er en issue på prosjektets GitHub, men det foreslås at det er opp til bruker av Rustiki rammeverket om de velger å benytte søk i frontend eller backend.

6 Konklusjon og videre arbeid

6.1 Konklusjon

Målet med prosjektet var å utvikle et wikirammeverk bygget med rammeverket MoonZoon og programmeringsspråket Rust. Det var planlagt fra starten at prosjektet skal fortsette som open source, slik at produktet kan utvikles videre. Dette var spesielt relevant siden rammeverket MoonZoon var i en tidlig fase i sin utvikling. Ringrev ønsket et slikt wikirammeverk for å dele informasjon internt og til sine kunder. Det var også viktig for bedriften at Rust ble benyttet siden dette var foretrukket språk for deres stack, og de ønsket å bidra til Rusts utviklingsmiljø.

Dette var en krevende oppgave for gruppen, da gruppen ikke hadde erfaring med Rust og MoonZoon. En stor del av gjennomføringen av prosjektet gikk ut på å lære og finne ut av disse verktøyene. Den største utfordringen ved prosjektet var hvor nytt rammeverket MoonZoon er, og at det har mangel på dokumentasjon og læringsressurser. På grunn av dette gikk det mye tid til å finne ut av det som er enkelt å få til i HTML, CSS og JavaScript. Oppdragsgiver var klar over utfordringene ved å arbeide med eksperimentell teknologi som rammeverket MoonZoon, og var imponerte over hva gruppen fikk til gjennom prosjektet.

Gruppen lyktes med å imøtekomme kravene for en MVP, som ble godkjent av oppdragsgiver. Dette til tross for perioder med begrenset progresjon grunnet utfordringer med teknologien. Produktet har en høyere grad av universell utforming enn hva gruppen anså som mulig ved bruk av MoonZoon. Gruppen er fornøyd med produktet som er levert.

Siden det ikke finnes et rammeverk for wiki utviklet i Rust fra før, er dette prosjektet innovativt innenfor Rusts utviklingsmiljø. Flere løsninger skrevet i Rust kan føre til lavere terskel for nye personer å ta i bruk Rust. Wikirammeverket gir Rust-utviklere muligheten til å raskt ha et utgangspunkt å jobbe ut fra. Dette kan være erfarne utviklere som vil ha en wiki til seg selv eller innenfor en bedrift, eller nybegynnere som ønsker å ha en løsning de kan eksperimentere med og bygge videre på etter hvert som de får ny kunnskap.

6.2 Videre arbeid

Gruppen har noen anbefalinger til de som skal utvikle prosjektet videre.

Det viktigste er forståelse for egenskapene til Rust. For teori og forståelse anbefales det å begynne med den offisielle Rust-boken «The Rust Programming Language» (Klabnik & Nichols, u.d.). For å lese og skrive Rust kode anbefales «Rustlings» (Rustlings, 2021) og «Rust by Example» (Rust-lang, 2021). Det er også gode kurs på Udemy med prosjekter for å lære språket, men disse er ikke gratis. Rust har en bratt læringskurve, og det kan dermed være lurt å bruke god tid på denne delen før man beveger seg videre til MoonZoon, som bruker Rust på frontend og backend.

Det antas at personer som ønsker å arbeide med dette prosjektet har kunnskap om HTML og CSS. Tidligere erfaring med JavaScript og andre programmeringsspråk er en fordel. Selv

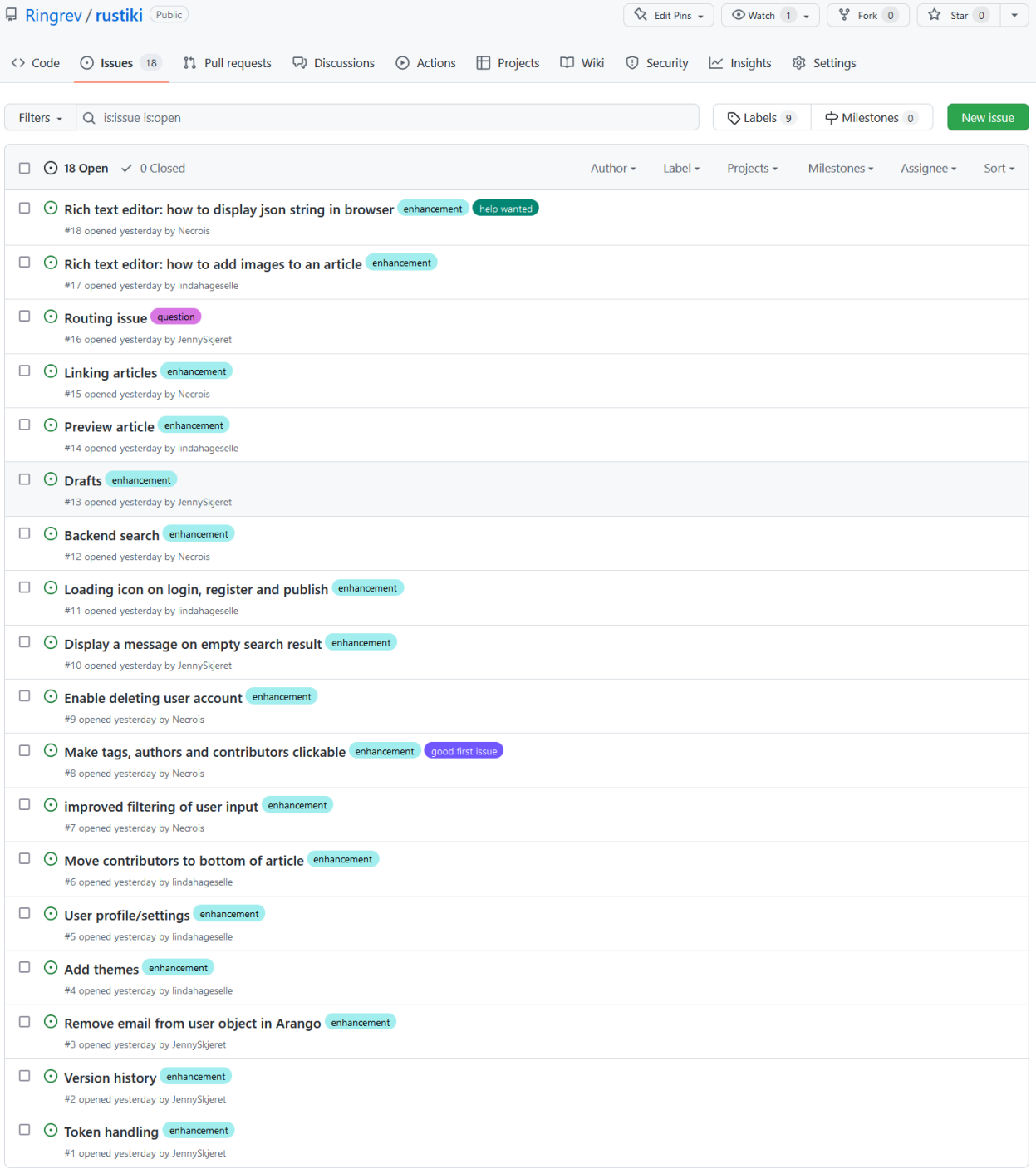
om MoonZoon har som mål å abstrahere bort behovet for å kunne å bruke HTML, CSS og JavaScript, er det likevel mange kjente konsepter fra disse som er brukt i MoonZoon. I tillegg kan man i MoonZoon bruke klassisk HTML (2.1.22 HTML) og CSS i deler av koden dersom man ønsker det.

For å bygge forståelse for MoonZoon anbefales det å utforske GitHub-siden til rammeverket (Kavík, 2022). Der er det mange eksempler, men gruppen anbefaler spesielt «pages» siden dette demonstrerer en nettside med routing, og inneholder en simulert innlogging. Dette var eksempelet som ble studert mest for å utvikle Rustiki. Prosjektene «todomvc» og «time_tracker» ble også studert. Å modifisere frontend-elementer i disse prosjektene er en god måte å bygge kompetanse. Det anbefales også å sette opp egne prosjekter hvor man tar i bruk forskjellige deler av eksempelprosjektene.

Hvis man ønsker å forstå hvordan frontend og backend kommuniserer med hverandre i MoonZoon, kan man man studere hvordan Rustiki har implementert dette. Dette blir utdypet i Rustikis readme-fil, som er vedlagt prosjektets kildekode (vedlegg G).

Rust er strengt på hva som kompileres. Dette resulterer i lav grad av feil relatert til syntaks og kjøretidsfeil. Logiske feil er den største faren og det anbefales en høyere grad av testing på dette. Kompilatoren vil også gi god tilbakemelding på hva som burde forbedres i form av «warnings». Koden vil kjøre med disse, men det anbefales å forbedre koden i henhold til advarslene som skrives ut i terminalen ved kompilering.

Rust og MoonZoon kan være krevende teknologier å jobbe med – hovedsakelig på grunn av en brå læringskurve. Man må ha kontroll på mye for å gjøre lite. Det er derfor viktig å arbeide strukturert og målrettet over en lengre periode og ta seg tid til refleksjon. Kommunikasjon med andre utviklere er også anbefalt – her er det mulig å bli med i kanalene for MoonZoon og Rust på Discord, som ligger linker til på henholdsvis MoonZoons GitHub (Kavík, 2022), og Rusts offisielle nettside (Rust Team, u.d.).



Figur 4: Gruppen har opprettet issues på prosjektets GitHub.

Dette prosjektet er open source og vil få nye utviklere og bidragsytere etter gruppen har levert inn oppgaven. Gruppen ønsker å gjøre det enklere for utviklere å bidra til dette prosjektet, og har derfor laget issues på forbedringer og ny funksjonalitet for Rustiki i prosjektets GitHub, som vist i figur 4. Ved tidspunktet prosjektet leveres, anbefaler gruppen at det fokuseres på issues relatert til tekstredigeringsverktøyet Quill.js – referert til som «Rich text editor» på prosjektets GitHub.

Samfunnspåvirkning

Dette kapittelet handler om samfunnspåvirkningen til prosjektet og verktøyene som er brukt, samt etiske vurderinger relatert til produktet.

Rusts utviklingsmiljø

Gruppens wikirammeverk er i seg selv et bidrag til Rusts utviklingsmiljø, som nevnt i konklusjonen (6.1 Konklusjon). Tidligere måtte man ha laget sin egen løsning helt fra bunnen av for å få en wiki skrevet i Rust. Rustiki passer godt for både nye og erfarne Rust-utviklere som ønsker å ha et utgangspunkt for en wiki.

Som nevnt i kapittel 4 (4.2.6 Bidrag til rammeverket Moonzoon) har gruppens prosjekt ført til utvikling i Rust fullstack-rammeverket MoonZoon, både direkte og indirekte. Gruppen har bidratt med en komponent til MoonZoon som tilsvarer et HTML textarea, og de har bidratt med muligheten til å sette språk-attributt på nettsider og paragrafer av tekst. Hovedutvikleren av MoonZoon har planer om at modulen gruppen laget som inneholder definisjonen av språk-attributtene senere kan bli et offentlig tilgjengelig bibliotek som også kan brukes av Rust-utviklere som ikke bruker MoonZoon. Gruppens prosjekt har inspirert utviklerne og bidragsyterne til å lage eksempelprosjekt på integrasjon av JavaScript, samt lagt til mer dokumentasjon.

Bærekraft i prosjektet

Som nevnt under kapittel 2 (2.1.5 Rust), kom det i 2017 en forskningsrapport som så på energieffektiviteten av forskjellige programmeringsspråk. På de testene som ble utført bruker Rust betydelig mindre energi og ressurser enn de andre språkene som er testet, bortsett fra C og C++. Anderson (2021) skriver at denne studien kan være problematisk siden den ikke tar hensyn til at forskjellige språk kan ha mange forskjellige implementasjoner og kompilatorer, og at noen implementasjoner av funksjonalitet i språk vil være mer effektive enn andre implementasjoner. Det er likevel ingen tvil om at Rust er mer effektivt og mindre ressurskrevende enn mange andre språk.

Når man tar i betraktning hvor mye kode som skrives hvert år og setter det i sammenheng med at den koden bruker energi for å kjøres, vil det å bytte ut kode skrevet i et mer ressurskrevende språk med kode skrevet i Rust ha en positiv innvirkning på verdens ressursforbruk.

Med økende grad av skytjenester og digitalisering satt i sammenheng med energikrise i Europa, kan det argumenteres for nytten av programmeringsspråk med lavere energibehov. Betalingsmodellen til de fleste skytjenester beregnes fra forbruk av serverkapasitet, dermed kan programvare skrevet i språk som Rust bidra til lavere kostnader for bedriften, og i tillegg redusere belastningen på datasentrene. Slik blir kapasiteten bedre utnyttet mellom flere brukere. Her kan kostnadsbildet variere for ulike bedrifter og det vil trolig innebære en forlenget utviklingstid i begynnelsen. Det er dermed viktig å sammenlikne

utviklingskostnader med forventede driftskostnader for å ta en slik vurdering. Sett i dette perspektivet kan det å bruke Rust gi positive resultater i forhold til andre språk.

Etiske vurderinger

Som dataingeniørstudenter er det viktig for gruppen å levere det beste produktet mulig, siden dette er etiske retningslinjer man bør følge som utvikler (Martin, u.d.). Det var krevende å vurdere hva det beste mulige produktet var, da teknologien var ukjent for gruppen før dette prosjektet. Gruppen har lagt vekt på kodestil, og dokumentasjon av det som leveres, samt åpenhet om svakheter funnet ved løsningen. Resultatet av disse vurderingene er dokumentert på prosjektets GitHub.

Valg av metode for samarbeid i prosjektet førte til et resultat gruppen mener samsvarer med skikker for god utvikling.

Universell utforming

MoonZoon støttet i utgangspunktet ikke alle de lovpålagte punktene for universell utforming. Som nevnt i kapittel 2 (2.1.4 Universell utforming) må man kunne sette språk-attributt på både nettsiden og tekster på nettsiden. Selv om det ikke var et krav fra oppdragsgiver at dette var en del av det leverte produktet, ville det vært uetisk dersom gruppen valgte å ignorere problemet eller holde informasjonen for seg selv. Gruppen valgte å fikse dette ved å lage funksjonalitet for å sette språk-attributter og opprette en pull-request til MoonZoon, som forklart i kapittel 4 (4.2.6 Bidrag til rammeverket MoonZoon). Pull-requesten ble merget inn, så MoonZoon er nå tilrettelagt for å kunne oppnå alle de lovpålagte punktene fra WCAG 2.0.

Sikkerhet

Hovedutvikleren av planlegger å fullstendig integrere JWT i MoonZoon, som nevnt i systemdokumentasjon (vedlegg C). Dette for å gjøre det raskere for utviklere å ha et sikkert produkt. Oppdragsgiver bruker selv MoonZoon, og er klar over hva rammeverket tilbyr og hvilken funksjonalitet som ligger lengre frem i tid. Likevel har gruppen dokumentert manglende sikkerhet som issue på Rustikis GitHub, for at utviklere som tar i bruk produktet skal vite hvordan det ligger an. Der har gruppen vist til MoonZoons GitHub for eventuelle oppdateringer i rammeverket.

Referanser

Agile Alliance, u.d. *Personas.* [Internett]
Available at: https://www.agilealliance.org/glossary/personas/
[Funnet 03 05 2022].

Agile Alliance, u.d. *What is Agile?.* [Internett]
Available at: https://www.agilealliance.org/agile101/
[Funnet 04 05 2022].

Anderson, T., 2021. *Can Rust save the planet? Why, and why not.* [Internett]
Available at: https://www.theregister.com/2021/11/30/aws_reinvent_rust/
[Funnet 4 May 2022].

Aragog, u.d. *Aragog.* [Internett]
Available at: https://aragog.rs/
[Funnet 27 04 2022].

ArangoDB, 2021. *ArangoDB: Graph and beyond.* [Internett]
Available at: https://www.arangodb.com/
[Funnet 27 04 2022].

Atlassian, u.d. *Git Feature Branch Workflow.* [Internett]
Available at: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow
[Funnet 29 04 2022].

Copes, F., 2021. *LogRocket.* [Internett]
Available at: https://blog.logrocket.com/jwt-authentication-best-practices/#store-jwts-securely
[Funnet 19 05 2022].

Digitaliseringsdirektoratet, u.d. *EUs webdirektiv (WAD) og WCAG 2.1.* [Internett]
Available at: https://www.uutilsynet.no/webdirektivet-wad/eus-webdirektiv-wad/265#nye_krav_gjelder_for_offentlig_sektor_fra_1_februar_2023
[Funnet 29 04 2022].

Dreimanis, G., 2020. *9 Companies That Use Rust in Production.* [Internett]
Available at: https://serokell.io/blog/rust-companies
[Funnet 27 04 2022].

Drumond, C., u.d. *Scrum.* [Internett]
Available at: https://www.atlassian.com/agile/scrum
[Funnet 29 04 2022].

Edspresso, u.d. *What are Norman's design principles?.* [Internett]
Available at: https://www.educative.io/edpresso/what-are-normans-design-principles
[Funnet 18 04 2022].

Fire Auth, 2022. *Fire Auth.* [Internett]
Available at: https://crates.io/crates/fireauth
[Funnet 28 04 2022].

GeeksforGeeks, 2021. *Software Engineering | Coupling and Cohesion.* [Internett]
Available at: https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/
[Funnet 13 05 2022].

GeeksforGeeks, 2022. *What's the difference between Scripting and Programming Languages?.* [Internett]
Available at: https://www.geeksforgeeks.org/whats-the-difference-between-scripting-and-programming-languages/
[Funnet 11 05 2022].

GitHub, u.d. *Creating an issue or pull request.* [Internett]
Available at: https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/working-with-your-remote-repository-on-github-or-github-enterprise/creating-an-issue-or-pull-request
[Funnet 11 05 2022].

Google Developers, 2021. *Tools for Web Developers: Lighthouse.* [Internett]
Available at: https://developers.google.com/web/tools/lighthouse
[Funnet 27 04 2022].

Google Developers, 2022. *Firebase Auth REST API.* [Internett]
Available at: https://firebase.google.com/docs/reference/rest/auth
[Funnet 28 04 2022].

Granevang, M., 2020. *backend.* [Internett]
Available at: https://snl.no/backend
[Funnet 11 05 2022].

Granevang, M., 2020. *frontend.* [Internett]
Available at: https://snl.no/frontend
[Funnet 11 05 2022].

Hageselle, L. G., Amundsen, J. H. & Valderhaug, J. S., 2022. *Rustiki.* [Internett]
Available at: https://github.com/Ringrev/rustiki
[Funnet 18 05 2022].

Hsu, S., 2018. *Session vs Token Based Authentication.* [Internett]
Available at: https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4
[Funnet 13 05 2022].

Kavík, M., 2022. *Frontend - Zoon.* [Internett]
Available at: https://github.com/MoonZoon/MoonZoon/blob/main/docs/frontend.md
[Funnet 15 05 2022].

Kavík, M., 2022. *MoonZoon.* [Internett]
Available at: https://github.com/MoonZoon/MoonZoon
[Funnet 11 05 2022].

Kavík, M. & Northman, R., 2021. *Philosophy and non-goals.* [Internett]
Available at:
https://github.com/MoonZoon/MoonZoon/blob/main/docs/philosophy_and_non_goals.md
[Funnet 27 04 2022].

Klabnik, S. & Nichols, C., u.d. *The Rust Programming Language.* [Internett]
Available at: https://doc.rust-lang.org/book/title-page.html
[Funnet 13 05 2022].

Klabnik, S. & Nichols, C., u.d. *The Rust Programming Language: Fearless Concurrency.* [Internett]
Available at: https://doc.rust-lang.org/stable/book/ch16-00-concurrency.html
[Funnet 27 04 2022].

Klabnik, S. & Nichols, C., u.d. *The Rust Programming Language: Understanding Ownership.* [Internett]
Available at: https://doc.rust-lang.org/book/ch04-00-understanding-ownership.html
[Funnet 27 04 2022].

Krause, R., 2021. *Maintain Consistency and Adhere to Standards (Usability Heuristic #4).* [Internett]
Available at: https://www.nngroup.com/articles/consistency-and-standards/
[Funnet 13 05 2022].

Lovdata, 2013. *Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger.* [Internett]
Available at: https://lovdata.no/dokument/SF/forskrift/2013-06-21-732
[Funnet 25 04 2022].

Martin, C. R., u.d. *The Programmer's Oath.* [Internett]
Available at: https://deepsource.io/programmers-oath/
[Funnet 18 05 2022].

MongoDB, u.d. *What is NoSQL?.* [Internett]
Available at: https://www.mongodb.com/nosql-explained
[Funnet 03 05 2022].

Mozilla, 2022. *HTML: A good basis for accessibility.* [Internett]
Available at: https://developer.mozilla.org/en-US/docs/Learn/Accessibility/HTML
[Funnet 13 05 2022].

Mozilla, 2022. *WAI-ARIA basics.* [Internett]
Available at: https://developer.mozilla.org/en-US/docs/Learn/Accessibility/WAI-ARIA_basics
[Funnet 13 05 2022].

Mozilla, 2022. *WebAssembly.* [Internett]
Available at: https://developer.mozilla.org/en-US/docs/WebAssembly
[Funnet 03 05 2022].

Open Source initiative, u.d. *licences.* [Internett]
Available at: https://opensource.org/licenses
[Funnet 11 05 2022].

Pereira, R. et al., 2017. *Energy Efficiency across Programming Languages.* [Internett]
Available at: https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf
[Funnet 14 04 2022].

Proton Technologies, 2020. *GDPR EU.* [Internett]
Available at: https://gdpr.eu/what-is-gdpr/
[Funnet 9 5 2022].

Proton Technologies, 2022. *gdpr eu.* [Internett]
Available at: https://gdpr.eu/right-to-be-forgotten/?cn-reloaded=1
[Funnet 04 05 2022].

Ranjan, R., 2021. *What is a Framework in Programming & Why You Should Use One.* [Internett]
Available at: https://www.netsolutions.com/insights/what-is-a-framework-in-programming/
[Funnet 11 05 2022].

Rust Team, u.d. *Rust.* [Internett]
Available at: https://www.rust-lang.org/
[Funnet 27 04 2022].

Rust-lang, 2021. *Rust by example.* [Internett]
Available at: https://doc.rust-lang.org/rust-by-example/
[Funnet 11 05 2022].

Rustlings, 2021. *GitHub Rustlings.* [Internett]
Available at: https://github.com/rust-lang/rustlings
[Funnet 11 05 2022].

Stocks, N., 2021. *Ultimate Rust Crash Course.* [Internett]
Available at: https://www.udemy.com/course/ultimate-rust-crash-course/
[Funnet 03 02 2022].

TechTarget, u.d. *tech target container-images.* [Internett]
Available at: https://www.techtarget.com/searchitoperations/definition/container-image
[Funnet 09 05 2022].

Usability.gov, u.d. *Wireframing.* [Internett]
Available at: Kilde: https://www.usability.gov/how-to-and-tools/methods/wireframing.html
[Funnet 03 05 2022].

W3, 2008. *Web Content Accessibility Guidelines (WCAG) 2.0.* [Internett]
Available at: https://www.w3.org/TR/WCAG20/
[Funnet 25 04 2022].

Wikipedia, 2021. *Session ID.* [Internett]
Available at: https://en.wikipedia.org/wiki/Session_ID
[Funnet 13 05 2022].

Wikipedia, 2022. *Document Object Model.* [Internett]
Available at: https://en.wikipedia.org/wiki/Document_Object_Model
[Funnet 13 05 2022].

Wikipedia, 2022. *Framework.* [Internett]
Available at: https://en.wikipedia.org/wiki/Framework
[Funnet 11 05 2022].

Wikipedia, 2022. *Frontend and backend.* [Internett]
Available at: https://en.wikipedia.org/wiki/Frontend_and_backend
[Funnet 11 05 2022].

Wikipedia, 2022. *HTML.* [Internett]
Available at: https://en.wikipedia.org/wiki/HTML
[Funnet 13 05 2022].

Wikipedia, 2022. *HTTPS.* [Internett]
Available at: https://en.wikipedia.org/wiki/HTTPS
[Funnet 13 05 2022].

Wikipedia, 2022. *JavaScript.* [Internett]
Available at: https://en.wikipedia.org/wiki/JavaScript
[Funnet 13 05 2022].

Wikipedia, 2022. *JSON Web Token.* [Internett]
Available at: https://en.wikipedia.org/wiki/JSON_Web_Token
[Funnet 13 05 2022].

Wikipedia, 2022. *Session (computer science).* [Internett]
Available at: https://en.wikipedia.org/wiki/Session_(computer_science)
[Funnet 13 05 2022].

Wikipedia, 2022. *WebAssembly.* [Internett]
Available at: https://en.wikipedia.org/wiki/WebAssembly
[Funnet 13 05 2022].

Wikipedia, 2022. *Wiki.* [Internett]
Available at: https://en.wikipedia.org/wiki/Wiki
[Funnet 11 05 2022].

Wikipedia, 2022. *Wikipedia.* [Internett]
Available at: https://en.wikipedia.org/wiki/Wikipedia
[Funnet 11 05 2022].

Vedlegg

Vedlegg A: Forprosjektplan

Første vedlegg i dette dokument.

Vedlegg B: Kravdokumentasjon

Andre vedlegg i dette dokument.

Vedlegg C: Systemdokumentasjon

Siste vedlegg i dette dokument.

Vedlegg D: Prosjekthåndbok

Se filen «Prosjekthåndbok» i mappen «Dokumenter» i zip-filen «Vedlegg».

Vedlegg E: Brukertester

Se undermappen «Brukertester» i mappen «Dokumenter» i zip-filen «Vedlegg».

Vedlegg F: Diagrammer

Se mappen «Diagrammer» i zip-filen «Vedlegg».

Vedlegg G: Kildekode

Se mappen «Kildekode» i zip-filen «Vedlegg».

Rustiki
Forprosjektplan

Versjon 1.0

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
26.01.2022	1.0	Første versjon	Joachim Amundsen, Linda Hageselle, Jenny Skjeret
16.02.2022	1.1	Forandring på MVP, inkluderer nå registrering og innlogging.	Linda Hageselle
04.04.2022	2.0	Forandring på effektmål siden vi formulerte oss feil om WebAssembly.	Linda Hageselle

Innholdsfortegnelse

1. Mål og rammer

1.1 Orientering

Den valgte oppgaven var på listen over mulige bacheloroppgaver for datastudenter ved NTNU i Ålesund. Gruppen valgte denne oppgaven fordi alle var enige i at det er en interessant oppgave. Det er også et pluss at oppgaven er tydelig på hva som skal oppnås og med hvilke verktøy. Det er noe rom for tolkning og tilpasning.

1.2 Problemstilling / prosjektbeskrivelse og resultatmål

Oppgavebeskrivelse

Ringrev vil ha et wiki-rammeverk som er ment å brukes av Rust-programmerere. Rust er et relativt nytt språk som mangler noen rammeverk, blant annet et wiki-rammeverk. Oppgaven skal løses ved å bruke fullstack Rust-rammeverket MoonZoon, og ArangoDB skal brukes til database-delen. Wiki-rammeverket skal ha støtte for sider, tags, bidragsytere og redigering. Løsningen er ment å være open source i fremtiden.

Problemstilling

Rust sitt utviklermiljø har ikke et dedikert wiki-rammeverk utviklet i Rust. Hvordan kan vi utvikle et wiki-rammeverk i Rust, med hjelp av blant annet rammeverket MoonZoon og databaseløsningen ArangoDB? Hva er fordeler og ulemper ved å utvikle rammeverket med Rust og nevnte verktøy?

Revidert:

Hvilke utfordringer kan oppstå når man kombinerer erfaringsbygging i Rust og MoonZoon med å utvikle et wikirammeverk med disse verktøyene, og hvordan kan resultatet bidra til Rusts utviklingsmiljø?

Resultatmål

Minimum viable product

- Registrering & Login
- Lagre i database
- Se artikler i wiki
- Legge til artikler i wiki
- Redigere artikler i wiki (string sidenavn/overskrift, string hovedtekst, string tags, string contributors)
- Slette artikler i wiki
- Søke etter artikler

Når prosjektet er ferdig, skal vi ha en forståelse for Rust og full stack-rammeverket MoonZoon. Vi skal ha brukt ArangoDB til database. Vi skal sitte igjen med et fungerende rammeverk for å lage wikier. Det skal være mulig å legge til bidragsytere på en wiki. Man skal kunne legge til sider med en artikkel per side. På artiklene skal man kunne legge til tags eller markere kodeord som skal kunne brukes til å søke etter artikler. Det skal være mulig å redigere sider og tags/kodeord, og bidragsytere.

1.3 Effektmål

Målet er å få en god øvelse i større prosjektarbeid som inneholder de viktigste fasene av utvikling av software. Dette innebærer praktisk bruk av agile metoder, planlegging, dokumentering, refleksjon og kommunikasjon med oppdragsgiver. Vi vil forbedre evnen vår til å ta et stort prosjekt for så å strukturert dele det opp i komponentene som kreves for å løse oppgaven.

Prosjektet gir oss en innføring i Rust og vi får se hvordan WebAssembly gjør det mulig å bruke programmeringsspråk i frontend som normalt kun brukes i backend. Dette er kompetanse som kan bli verdifull i arbeidsmarkedet de neste årene, basert på trender i bransjen.

Prosjektet skal også brukes til å vise til gjennomført arbeid som kan brukes som en demonstrasjon av ferdigheter når vi skal ut på arbeidsmarkedet.

1.4 Rammer

Mulig behov for penger for AWS pipeline, noe uklart hva kostnader blir for dette i skrivende stund.

Rust kurs.

2. Organisering

Gruppa: Joachim Hauso Amundsen, Linda Gjerde Hageselle, Jenny Skjeret Valderhaug

Veileder: Girts Strazdins

Ringrev: Arnaud Menant, Beate Stavdal Riise

3. Gjennomføring

3.1. Hovedaktiviteter

- Lære Rust, Moonzoon og ArangoDB
- Arbeid dokumenteres i Jira/Confluence under tasks. For annen dokumentasjon brukes discord, sharepoint og onedrive, avhengig av typen dokument.
- Arbeid utført kobles til task i en sprint i Jira. Utviklere kan i utgangspunktet velge tasks de selv ønsker, men gruppeavgjørelser kan overstyre egendefinerte valg avhengig av tiden som er tilgjengelig.
- Rapportskriving
- Utvikling

3.2. Milepæler

28. Januar – Innleveringsfrist forprosjektplan

22. April – Fremføring til Girts

06. Mai – Innlevering av førsteutkast til rapport til Girts

20. Mai – Innleveringsfrist Inspera

~20. Mai – Fremføring av prosjekt

4. Oppfølging og kvalitetssikring

4.1 Kvalitetssikring

- Alle gruppemedlemmer må se gjennom dokumenter før dokumentet anses som ferdig.
- Kode må fungere hos utvikler før man kan merge.
- Code review før man merger.

4.2 Rapportering

Gruppa har møte minst 1 gang i uka (vanligvis oftere), og kommuniserer ellers med hverandre på Discord. Code reviews blir gjennomført i gruppa. Refleksjon over sprints tas på et ukentlig møte der vi vurderer fremdrift og planlegger neste sprint.

Annenhver uke har gruppa sprintmøte med oppdragsgiver og veileder.

5. Risikovurdering

Skala: Lav, Middels, Høy

Hendelse	**Sannsynlighet**	**Konsekvens**	**Tiltak**
Kortvarig sykdom som forhindrer personen i å delta	Høy	Kan føre til at enkeltoppgaver blir forsinket	Redistribuere arbeidsoppgaver ved behov
Langvarig sykdom som forhindrer personen i å delta	Lav	Kan føre til at personen som er syk ikke kan delta over lengre tid	Diskutere med vedkommende, koble inn veileder hvis vi er usikre på hva vi skal gjøre videre
Gruppemedlemmer møter ikke opp til avtalt tidspunkt	Høy	Det er forventet at det kan skje misforståelser, forsovelser, eller at noe oppstår som hindrer noen i å møte til avtalt tid. Kan føre til små forsinkelser.	Hvis samme person ofte ikke møter opp til avtalt tid må dette diskuteres i gruppen for å finne en løsning
Oppgaver tar lengre tid enn forventet	Høy	Mindre tid på andre oppgaver	Må beregne lengre tid på lignende oppgaver kommende uker
Sprintmøter blir utsatt	Middels	Kan føre til at arbeid ikke kan bli påbegynt tidsnok pga. mangel på tilbakemelding	Prøve å finne oppgaver som kan startes på tidligere for å unngå å miste produktiv tid

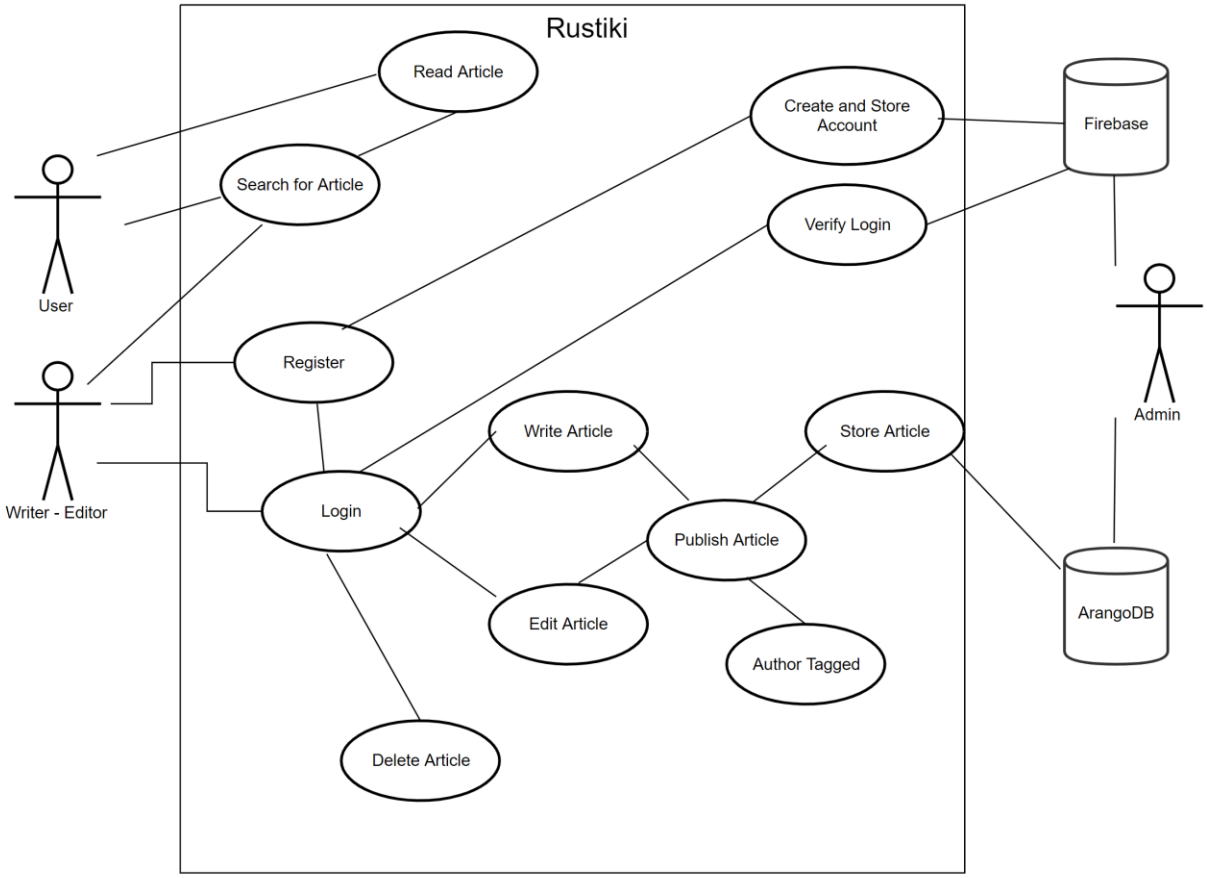
Rustiki
Kravdokumentasjon

Innholdsfortegnelse

1. Introduksjon

Dette dokumentet inneholder gruppens use case diagram, user stories og konseptskisser.

2. Use Case diagram



Figur 1: Use Case diagram av gruppens løsning.

Figur 1 viser et use case diagram av gruppens applikasjon. For å se denne figuren som vektorgrafikk, se filen «Rustiki Use Case Diagram» i vedlagt mappe «Diagrammer».

3. User stories

Følgende user stories ble laget med hensikt å vise de frem til oppdragsgiver, og er utformet på engelsk fordi møtene med oppdragsgiver foregikk på engelsk.

3.1 Rust programmers/Wiki admins

3.1.1 Personas:

- Michael (22) works in customer service. He started learning Rust a little while ago for fun. He wants a website where he as a beginner can compile the most important things he learns.
- Haley (29) works on a team of programmers that uses Rust. She and her team want a wiki for internal use within the team where they can share information with each other.
- Alex (23) works in IT at a marketing company. She wants to set up a wiki on which other departments can

post tutorials that they can send to their clients.

3.1.2 User stories:

- As an admin I want precise setup instructions for the wiki framework so I can set it up without asking for help.
- As an admin I want the code to have comments so that I don't need to spend hours trying to interpret the code.
- As an admin I want to be able to add members to the wiki so they can create and edit content.

3.2 Wiki users

3.2.1 Personas:

- Diane (32) is the manager of a marketing company. She wants a wiki where she can post tutorials that help her clients with their online marketing.
- George (39) is the communication manager of a company that does system development. He wants to link to a wiki article when he posts new social media posts updating their followers on projects.
- Tom (25) has a huge interest in gaming. His friend set up a wiki site for him so he could start posting game reviews.

3.2.2 User stories:

- As a wiki user I want a visible button for creating a new article, so I don't have to look around for it.
- As a wiki user, I want to be able to edit and format the content, so it looks how I want it to.
- As a wiki user I want articles to have an edit button so I can correct errors or contribute to other users' articles.
- As a wiki user I want my username to be added to the list of contributors after I edit an article that is not mine, so other users know I contributed.
- As a wiki user I want to be able to add tags while I create/contribute on an article, so it's easier for others to search for it.
- As a wiki user I want to be able to remove a tag while I'm creating/contributing on an article, in case I make a typo, or a tag needs to be changed.
- As a wiki user I want to be able to delete my own article if it's no longer relevant.
- As a wiki user I want to be able to click cancel when I'm creating or contributing on an article if I change my mind and don't want changes saved.

3.3 Wiki readers

3.3.1 Personas:

- Jim (20) is a university student who is learning Rust. He found a wiki built on Rustiki that posts tips for novice Rust programmers. It's important to Jim that he can search for the subject he is currently curious about.
- Maggie (53) runs a jewellery making business. The marketing consultant she hired sends Maggie a link to their wiki that is built on Rustiki, where they post tutorials for their clients. It's important for Maggie that she can browse several articles.
- Claire (24) is almost done with her degree in computer science. She follows a company on Instagram that posts about their projects linking their wiki which is built on Rustiki. She wants to learn more about the company's projects before applying for their position as junior fullstack developer.

3.3.2 User stories:

- As a wiki reader, I want to see who has written an article, to help decide it the information is trustworthy.
- As a wiki reader, I want to be able to see an overview of several different articles, so I don't have to search for them.
- As a wiki reader, I want to be able to search for tags so I can read articles related to a certain topic.
- As a wiki reader, I want to be able to search for a user so I can read posts authored by or contributed on by that user.

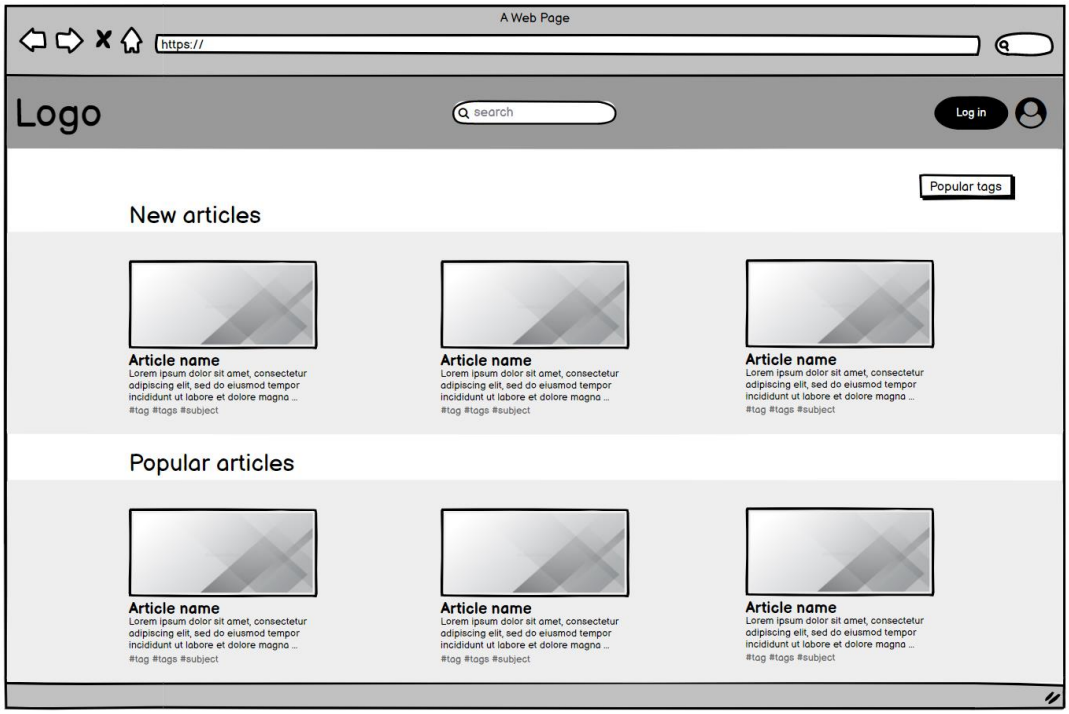
- As a wiki reader, I want to be able to search for keywords from articles' titles, so I can look up the article if I already know its title.
- As a wiki reader, I want to be able to press the logo to go back to the homepage after reading an article, because I am used to this functionality from other websites.

4. Prototyper

Dette kapittelet inneholder konseptskisser (wireframes) som gruppen lagde. Disse ble både vist frem til oppdragsgiver, samt utført brukertester på.

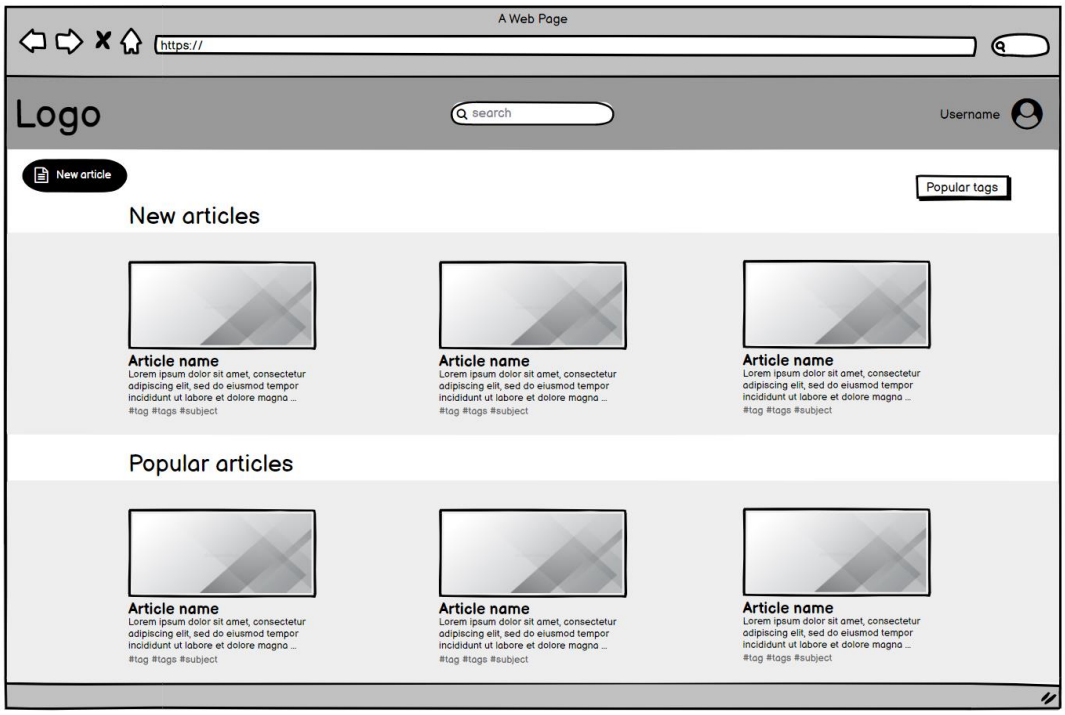
4.1 Konseptskisser

Gruppen lagde konseptskisser i Balsamiq Cloud.



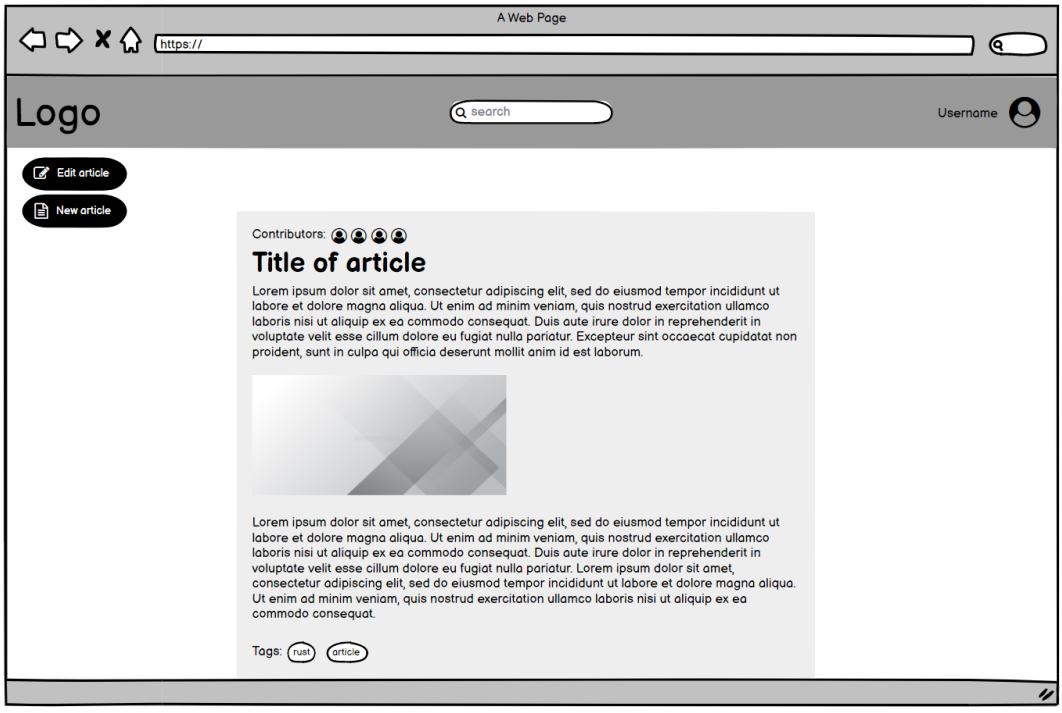
Figur 2: Nettsidens forside for en utlogget bruker.

Figur 2 viser hvordan gruppen så for seg at forsiden kunne se ut for en utlogget bruker.



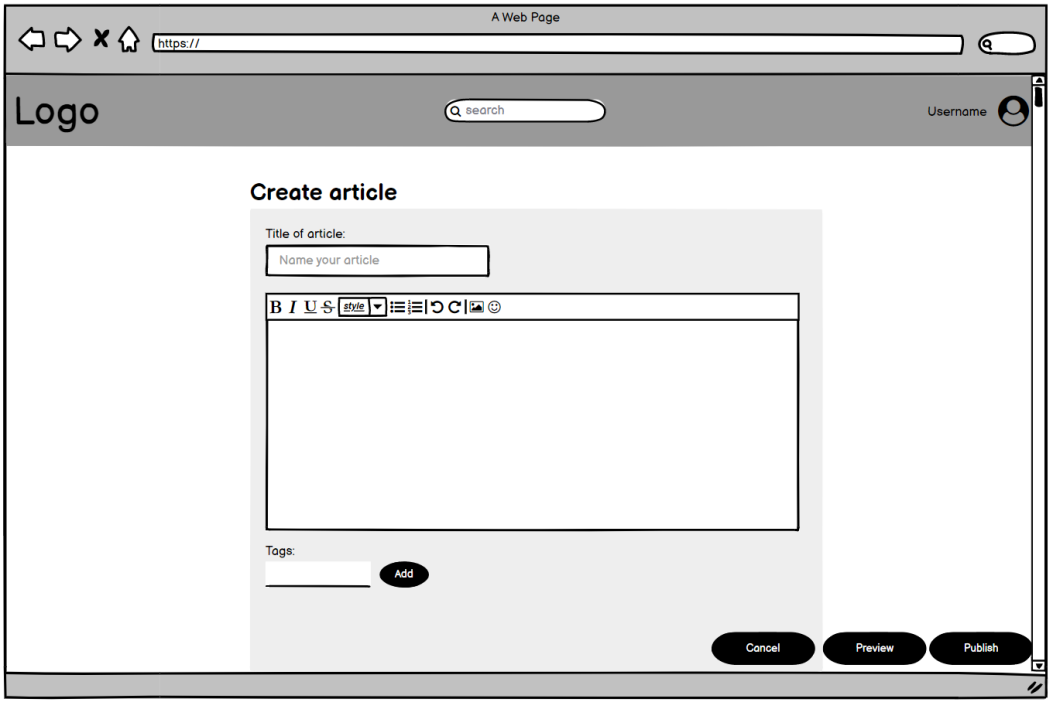
Figur 3: Nettsidens forside for en innlogget bruker.

Gruppens skisse av nettsiden forside for en innlogget bruker er vist i figur 3.



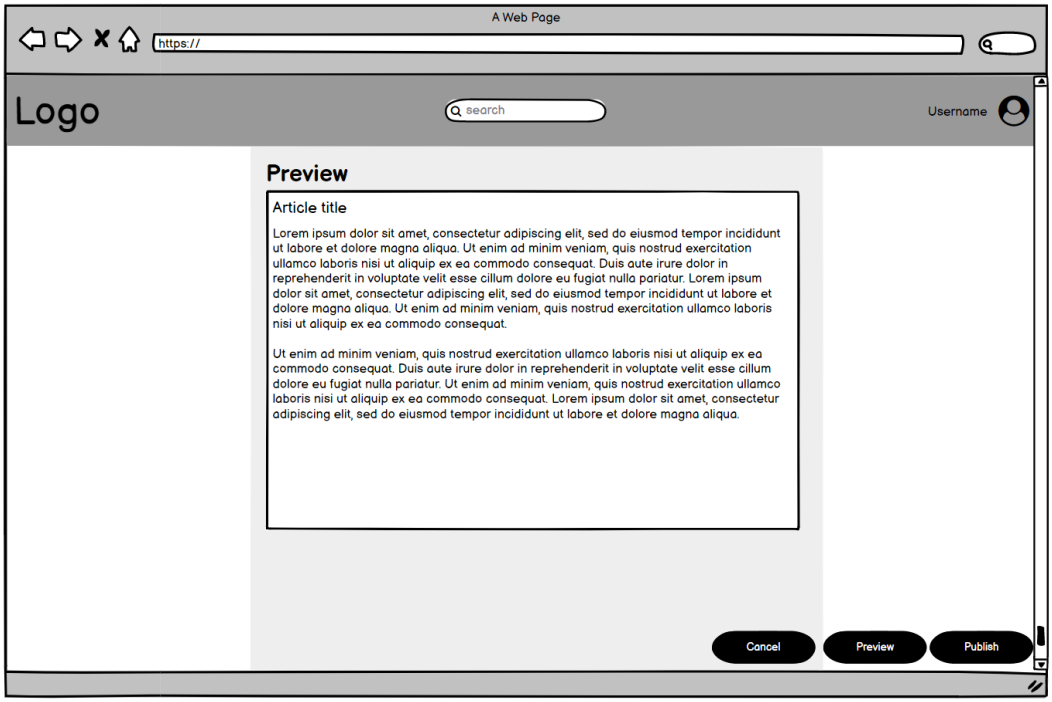
Figur 4: Skisse av visning av en artikkel.

Skissen i figur 4 viser hvordan gruppen så for seg at en artikkel kunne vises.



Figur 5: Skisse av laging av ny artikkel.

Figur 5 viser hvordan gruppen så for seg at lagingssiden for artikler kan se ut.



Figur 6: Skisse av forhåndsvisning av artikkel.

Skissen i figur 6 viser hvordan gruppen tenkte forhåndsvisning av artikkel kunne fungere. Forhåndsvisning ble ikke del av kravene for MVP, så denne ideen er ikke tatt i bruk.

Rustiki
Systemdokumentasjon
<v1.0>

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
18.05.2022	1.0	Ferdigstilte dokument	Linda Gjerde Hageselle, Joachim Amundsen, Jenny Skjeret Valderhaug

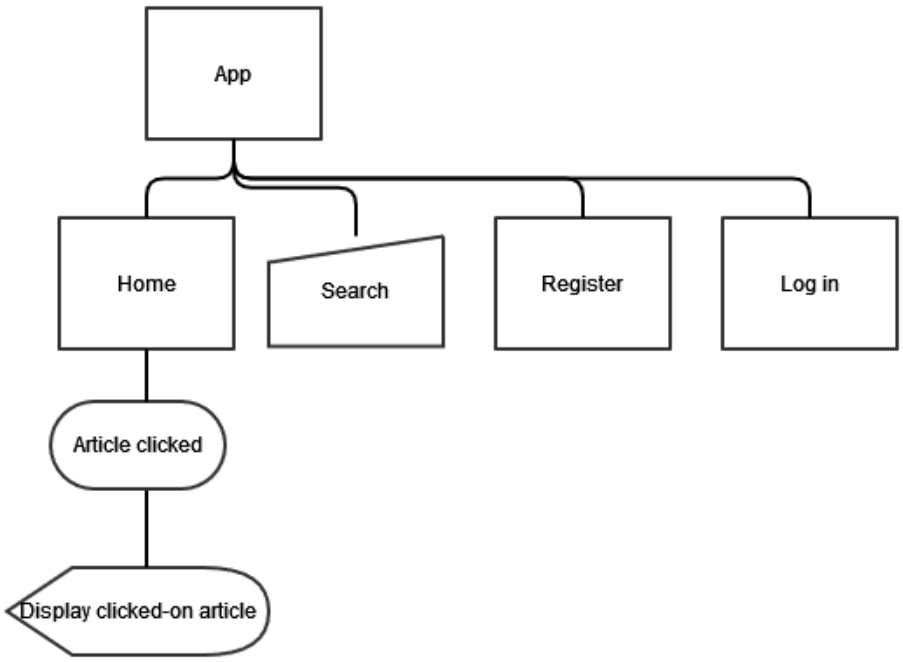
Innholdsfortegnelse

1. Introduksjon

Dette dokumentet inneholder beskrivelse av tekniske konsepter i gruppens prosjekt.

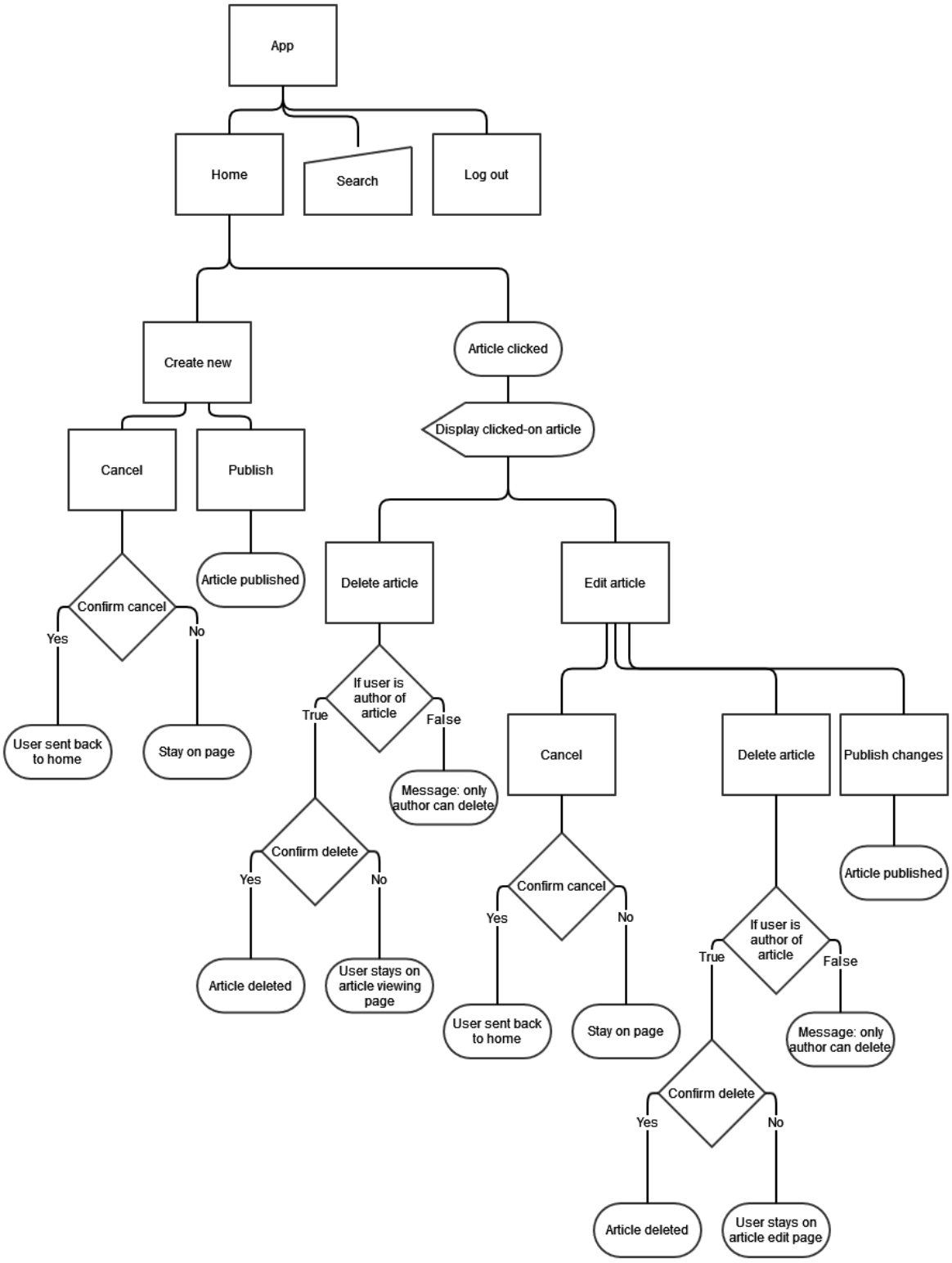
2. Flyt-diagram

I dette delkapittelet illustreres det via flyt-diagrammer hvilke valg som er tilgjengelig for en bruker.



Figur 1: Flyt for en bruker som ikke er logget inn.

I figur 1 kan man se flyten for en bruker som ikke er logget inn.

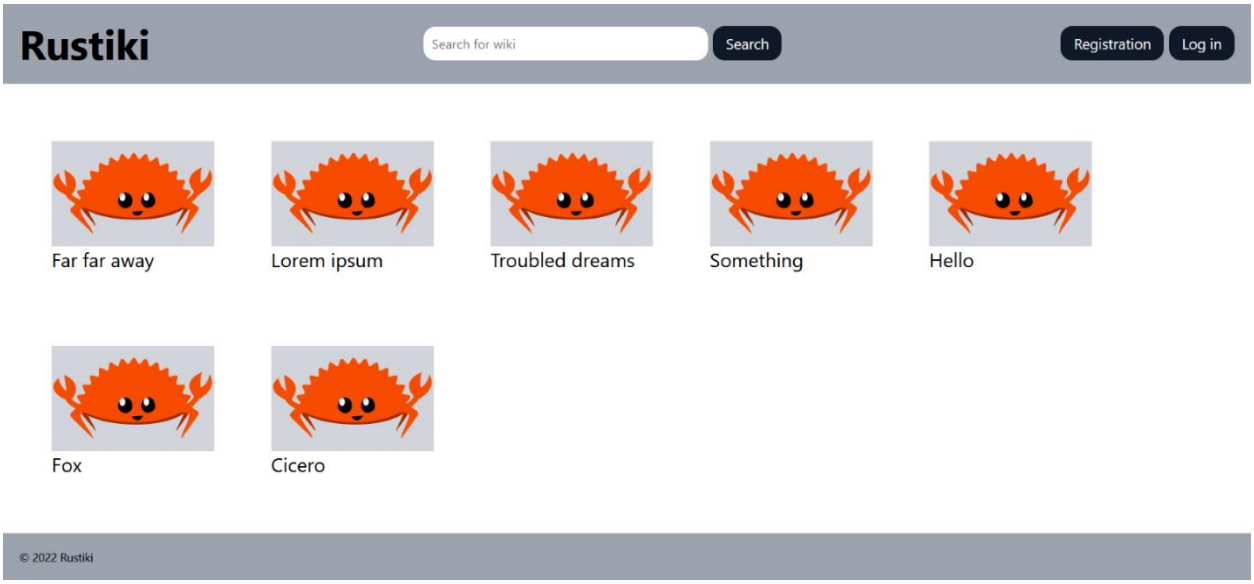


Figur 2: Flyt for en innlogget bruker.

I figur 2 kan man ser flyt for en innlogget bruker. For å se dette diagrammet som vektorgrafikk, se filen kalt «Flowchart Logged In» i mappen «Diagrammer» i innlevert zip-fil.

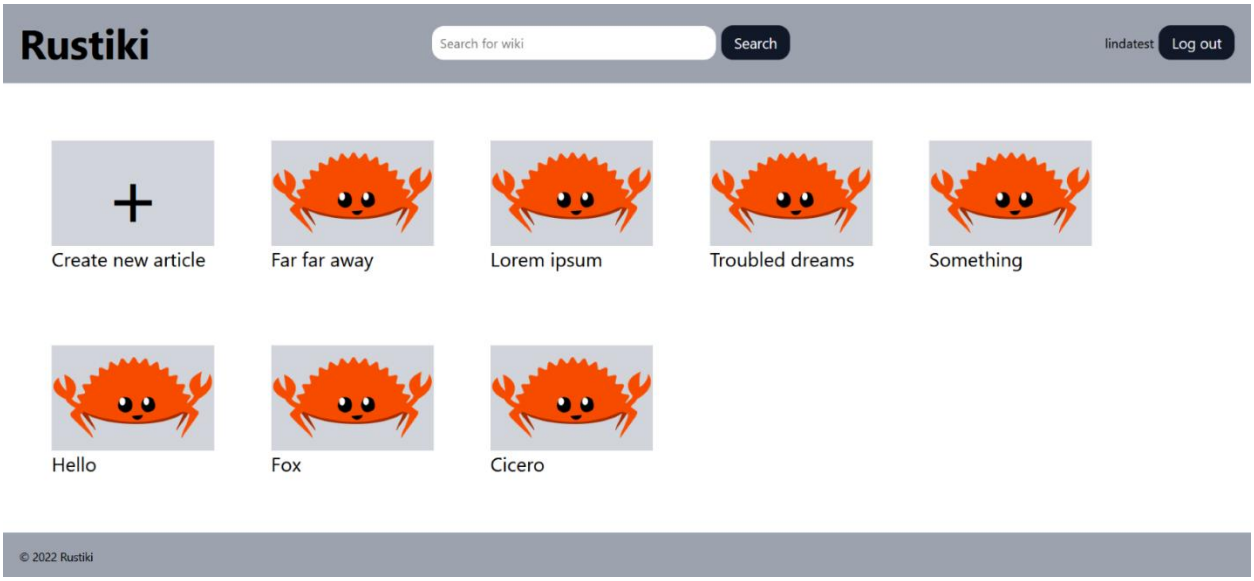
3. Beskrivelse av produktet

3.1 Forside



Figur 3: Produktets forside.

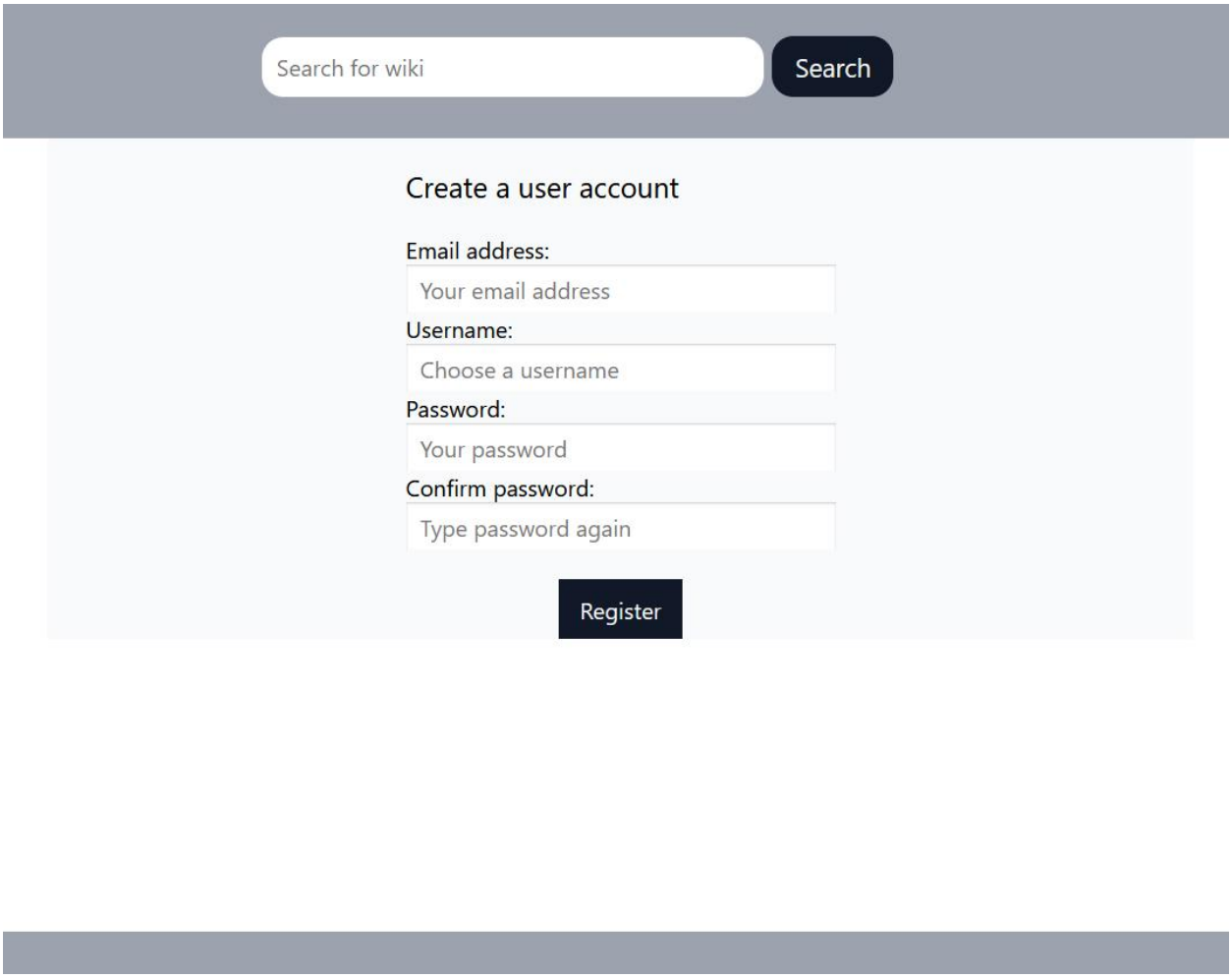
Når man laster inn forsiden blir alle artiklene fra wikiens ArangoDB databasesamling «Article» hentet og lagret i en vektor i frontenden. En vektor tilsvarer det vi kjenner som en liste i for eksempel Java. Representasjoner av hver artikkel vises på forsiden som et bilde med artikkelens tittel under, som vist i figur 3. Bildet som er benyttet som placeholder er unntatt copyright (Tölva, u.d.). Artiklene er sortert etter tidspunktet de sist ble oppdatert. Når man holder musepekeren over en artikkel, blir fargen bak forandret for å indikere at man kan interagere med komponenten. Trykker man på denne kommer man inn på en side som viser hele artikkelen.



Figur 4: Forsiden når bruker er innlogget.

Hvis man er logget inn, får man på forsiden frem en knapp for å lage ny artikkel som første artikkel-komponent. Dette er vist i figur 4. Man kommer inn på en side for å lage ny artikkel når man trykker på denne knappen. For å vise forskjellige knapper om bruker er innlogget eller ikke utføres ved bruk av et signal i koden.

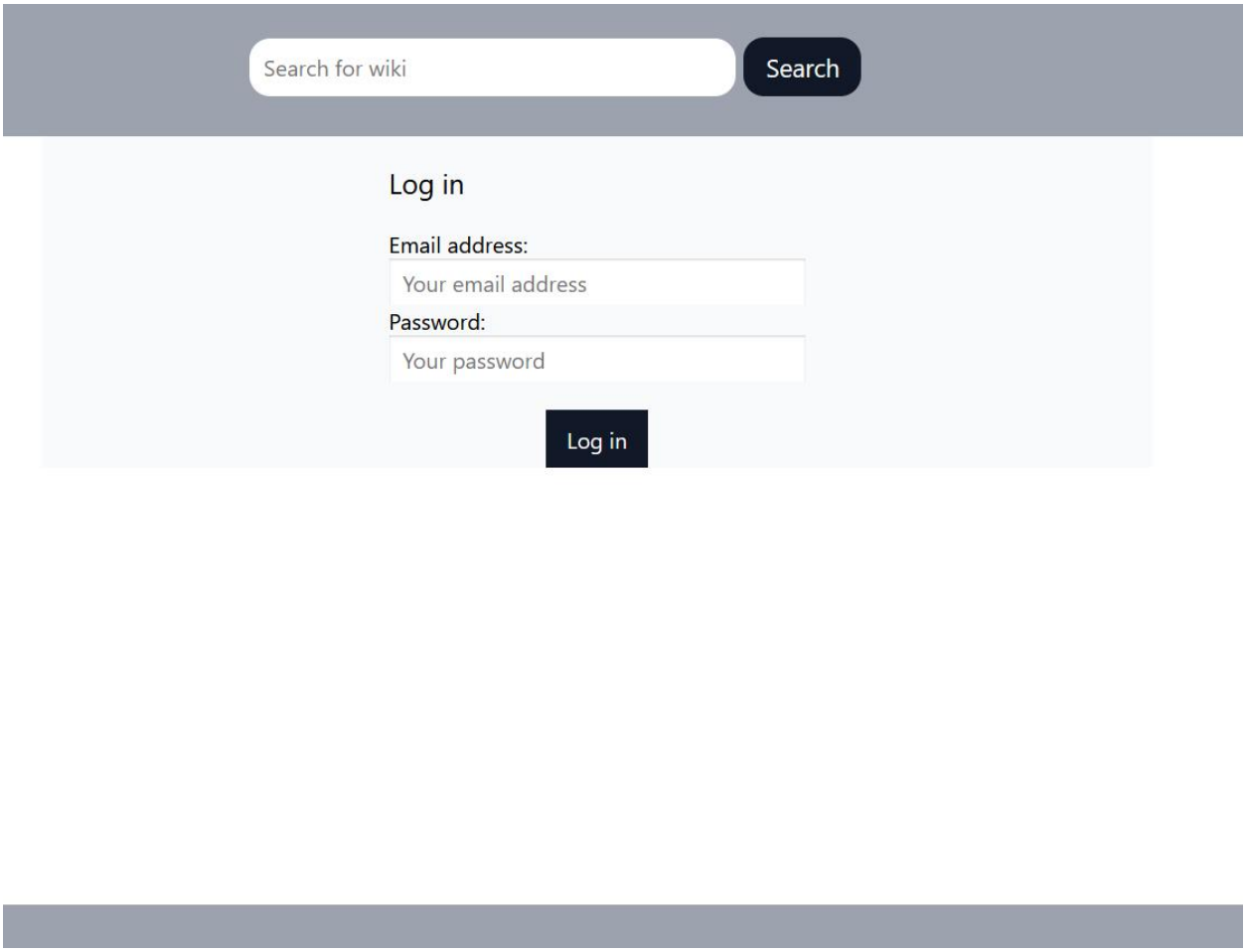
3.2 Registrering



Figur 5: Registrering av en ny bruker.

For å registrere seg som en bruker må man skrive inn epost, brukernavn og passord, som vist i figur 5. Passordet må skrives to ganger for å forsikre at brukeren skrev det de mente å skrive. Hvis passordet er kortere enn 6 tegn vil bruker bli fortalt at passordet må være minst 6 tegn langt. Når bruker trykker «Register» vil det først bli sjekket om dette brukernavnet allerede eksisterer i databasens. Dersom det finnes en bruker med dette navnet får bruker beskjed om at brukernavnet er tatt, og de må velge et nytt. Hvis brukernavnet er ledig forsøkes det å registrere brukeren med denne eposten i Firebase. Dersom eposten allerede er brukt vil registreringen feile og det kommer frem en beskjed under inputfeltene om at epostadressen er ugyldig. Hvis epostadressen er ledig vil Firebase sende tilbake et resultat til backenden av wikien som inneholder en kopi av brukeren. Deretter brukes brukernavn, epost og den unike identifikasjons-stringen generert av Firebase til å lagre denne brukeren i ArangoDB-samlingen «User». Etter registrering blir bruker automatisk logget inn.

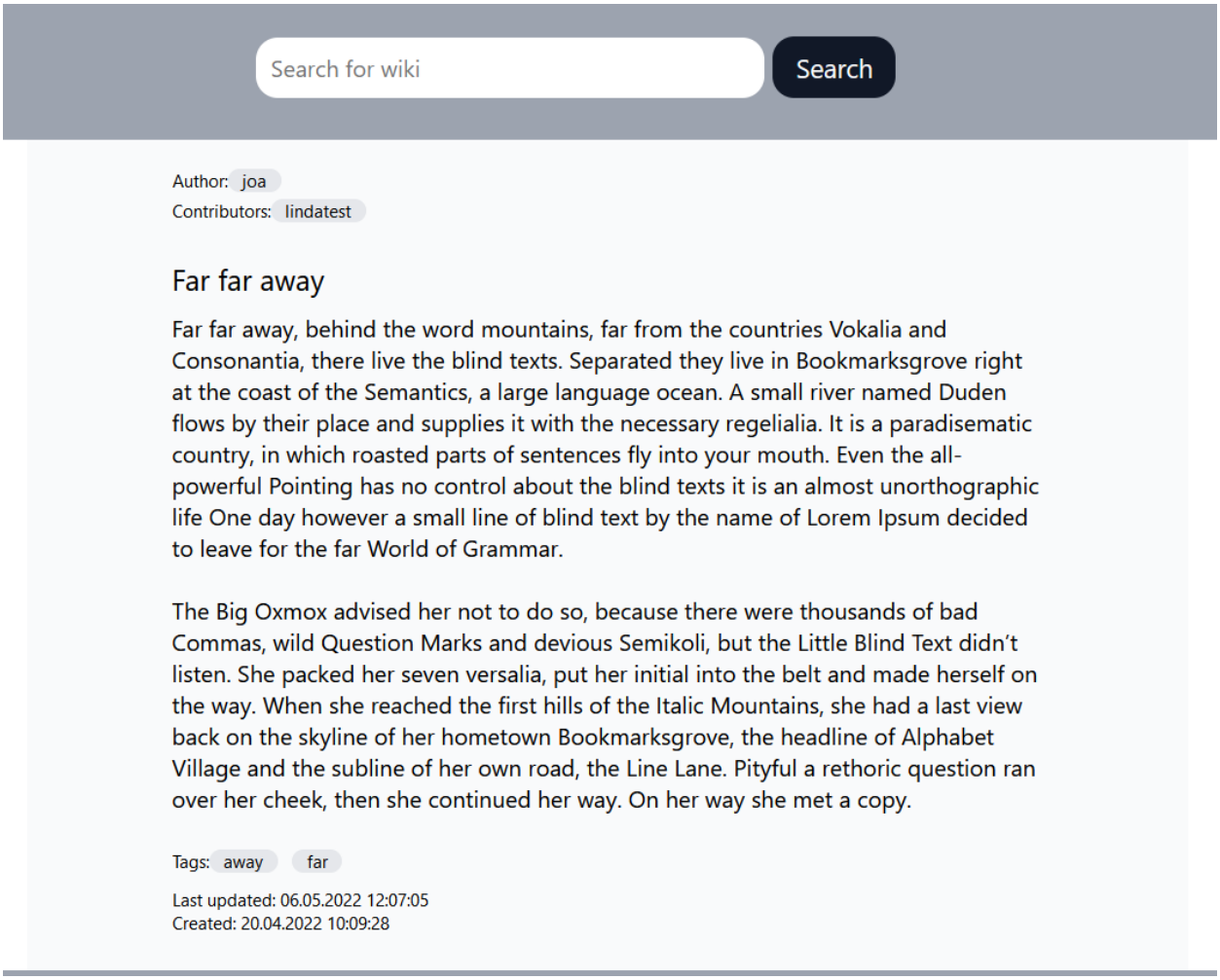
3.3 Innlogging



Figur 6: Siden for innlogging av bruker.

Det er vist i figur 6 hvordan innloggingssiden ser ut. Når en bruker prøver å logge inn blir autentisering av epost og passord håndtert av Firebase. Dersom backenden får beskjed fra Firebase om at enten epost eller passord er feil, får bruker under inputfeltene fram en beskjed som sier «Incorrect input, please try again». Hvis brukerinput er riktig og bruker blir autentisert får backenden et Firebase-brukeren og et autentiseringstoken. Deretter blir brukernavnet til brukeren hentet fra ArangoDB-samlingen «User» ved å querye brukeren som har den spesifiserte identifikasjons-stringen. Et objekt bestående av ID, epost, brukernavn og token blir sendt til frontenden der tilstanden til innlogget bruker og token blir oppbevart. I brukergrensesnittet vil man nå se brukernavnet sitt oppe i høyre hjørne sammen med en knapp for å logge ut.

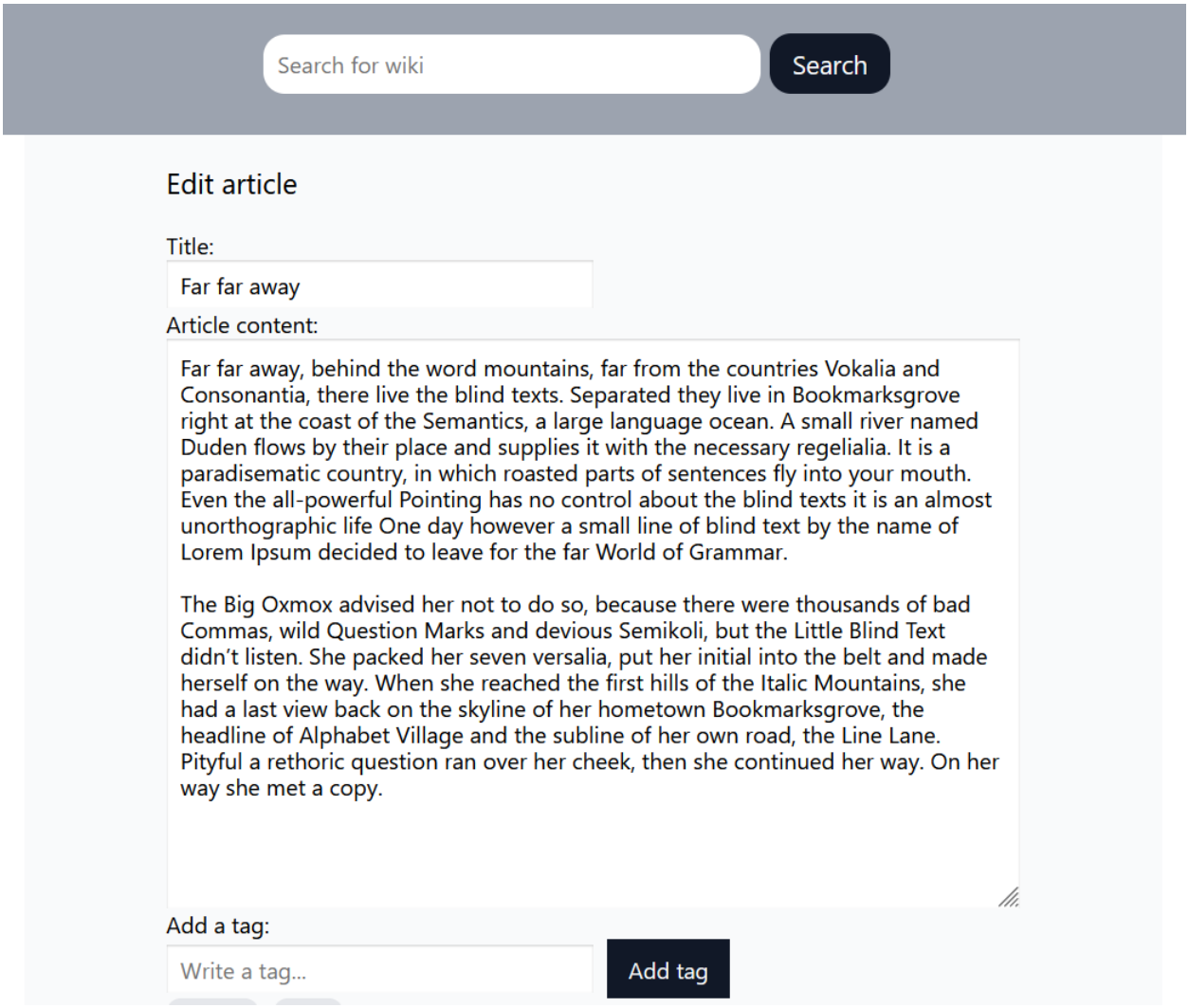
3.4 Visning av en artikkel



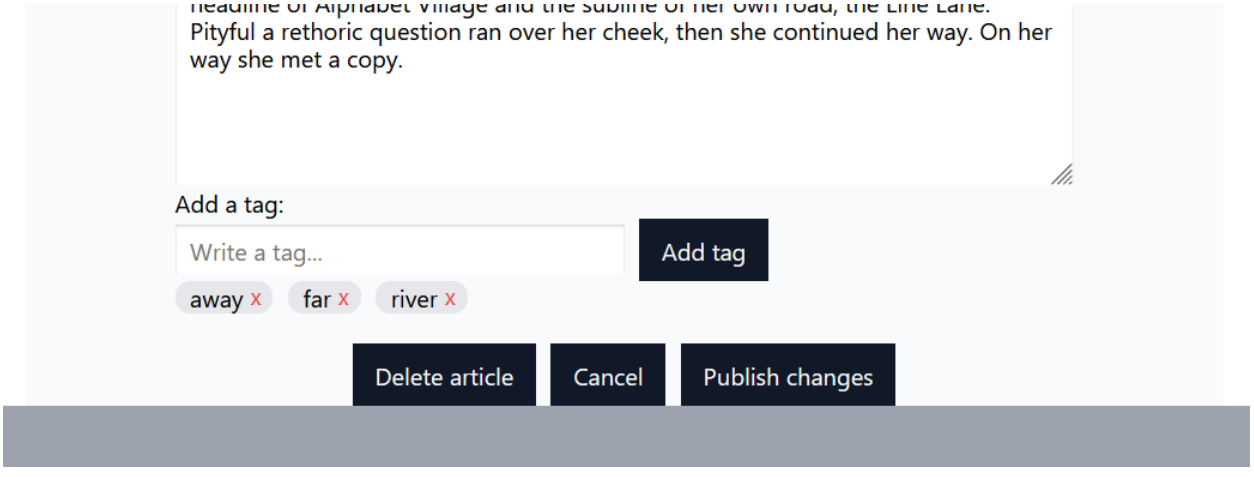
Figur 7: Visning av en artikkel.

Når man på forsiden trykker på en artikkel, blir artikkelobjektet hentet fra vektoren lagret i frontend og vist på en egen side. Figur 7 viser hvordan en artikkel ser ut. Dersom bruker er logget inn får de fram to knapper som lar de redigere eller slette artikkelen. Når man leser en artikkel får man se hvem som er forfatter, hvem som eventuelt er bidragsytere, tittel av artikkelen, artikkelens innhold, tags som er lagt til på artikkelen, siste tidspunkt den ble oppdatert, og tidspunkt den ble opprettet.

3.5 Redigere artikkel



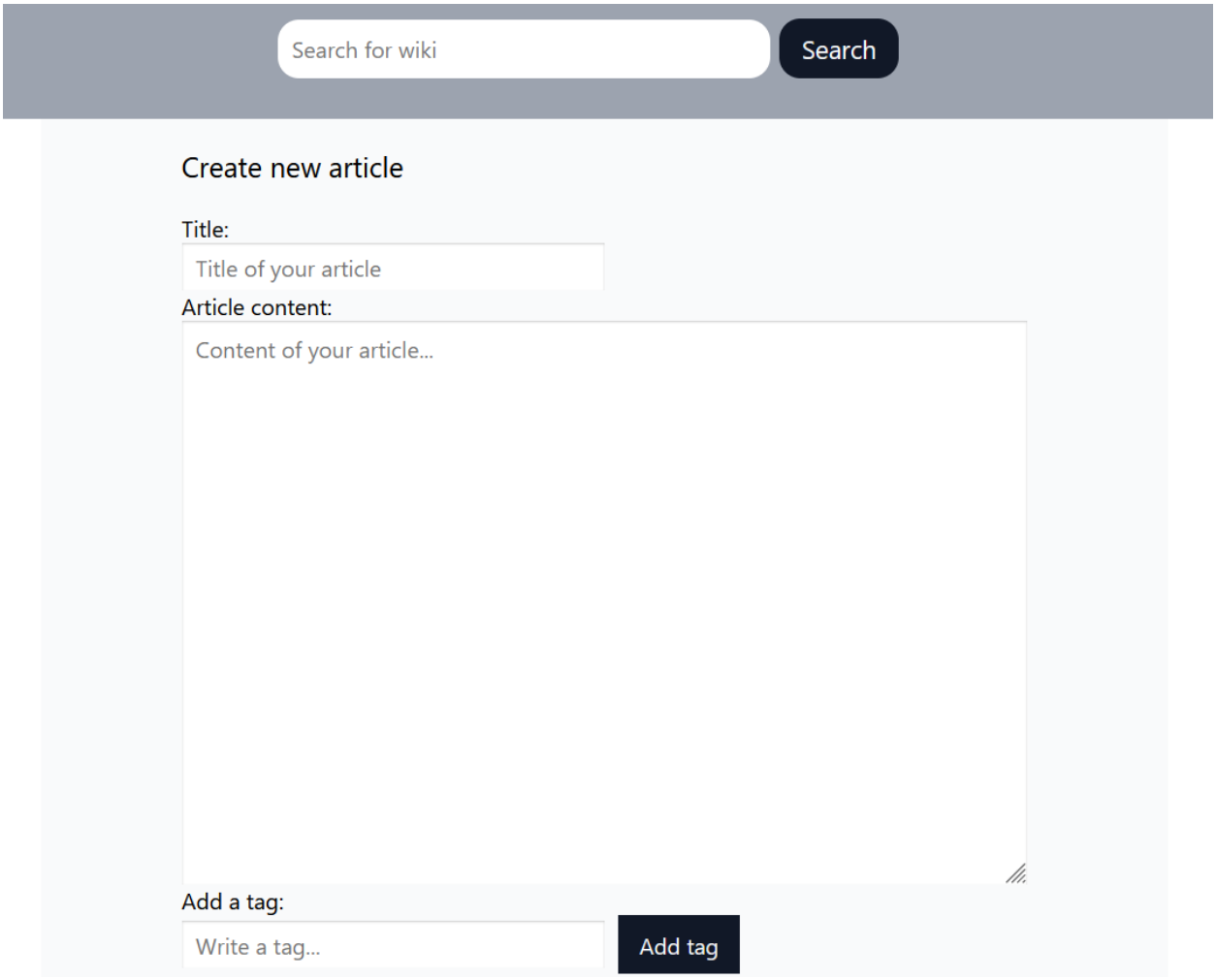
Figur 8: Redigering av en artikkel.



Figur 9: Bunnen av redigeringssiden for en artikkel.

Hvis man trykker på «Edit» inne på visningssiden for en artikkel, vil det komme fram en side som lar bruker redigere artikkelen. Denne siden vises i figur 8 og 9. Skriften i tittel og innhold kan bli forandret, og man kan slette og legge til tags. På bunnen av artiklene har bruker mulighet til å slette eller publisere artikkelen, eller å kansellere redigeringen. Hvis bruker er forfatter av artikkelen vil et trykk på knappen «Delete article» resultere i en dialog som spør om de er sikre på at de vil slette artikkelen. Er brukeren derimot ikke forfatter vil de få opp en dialog som sier at kun forfatter har rett til å slette en artikkel. Hvis brukeren trykker på «Cancel» kommer det frem en dialog som sier at forandringene de har gjort ikke vil bli lagret, og de får velge om de ønsker å kansellere eller ikke. Når en bruker trykker «Publish» blir artikkelen i databasen oppdatert med endringene, og bruker blir sendt til forsiden.

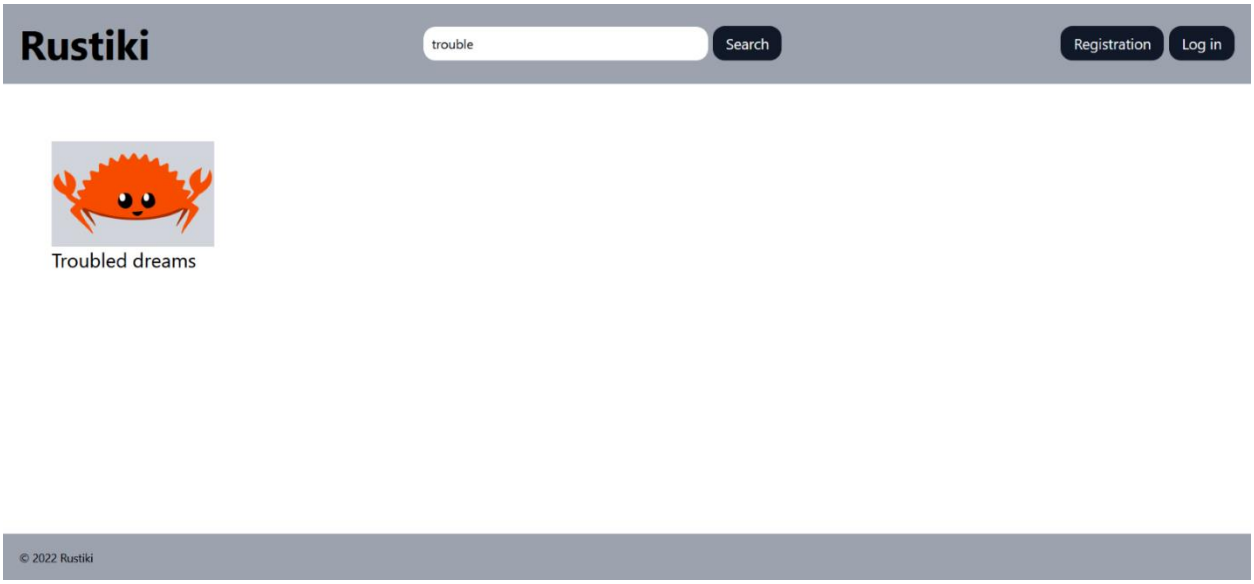
3.6 Lage ny artikkel



Figur 10: Side for å lage en ny artikkel.

Hvis man trykker på knappen «Create new article» fra forsiden, får man frem siden for skriving av ny artikkel. Den består av et standard inputfelt for tittelen av artikkelen, en textarea-komponent som gruppen har laget selv som bidrag til rammeverket MoonZoon, og til slutt et inputfelt for å skrive tags. Denne siden vises i figur 10. Feltet hvor du skriver innholdet i artikkelen lagrer teksten som en String, og støtter at man legger inn ekstra luft mellom linjer for å indikere paragrafer. Når man skriver inn et ord i inputfeltet for tags kan man legge til taggen enten ved å trykke Enter eller å trykke på knappen ved siden av inputfeltet. Dersom taggen allerede eksisterer på artikkelen vil bruker få opp en dialog om dette, og taggen vil ikke bli lagt til. Man kan også slette tagger.

3.7 Søk



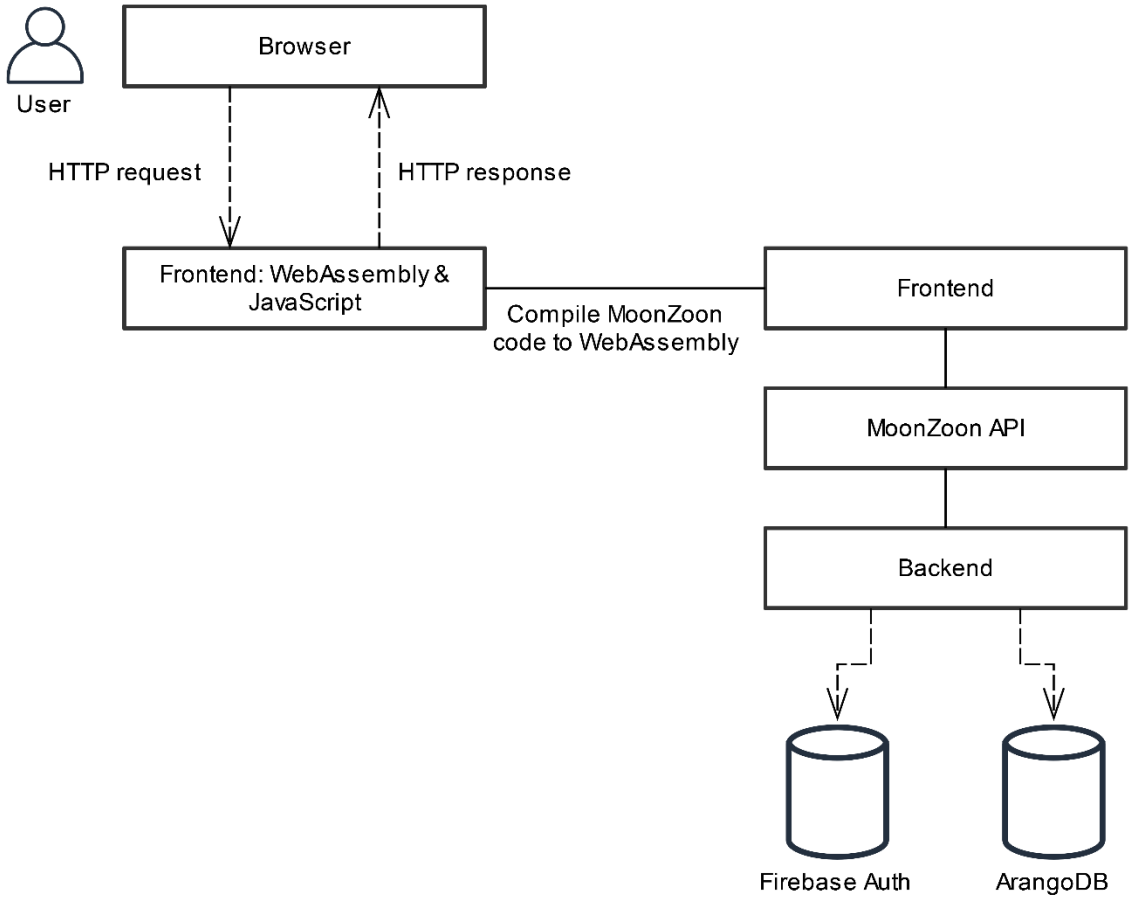
Figur 11: Resultater etter å ha søkt etter «trouble» i søkefeltet.

Søk foregår direkte på forsiden av wikien. Man skriver i søkefeltet i headeren, og for søke trykker man enten Enter-knappen på tastaturet eller «Search»-knappen til høyre for søkefeltet. Figur 11 viser hvordan det ser ut når man har gjort et søk der det er ett treff. For å komme ut av søk kan man enten trykke på logoen oppe til venstre eller fjerne søkeordet i søkefeltet. Det går an å søke på tags, tittel, forfatter og bidragsytere. I koden foregår søking i vektoren av artikler som ligger i frontend.

Når artikler blir hentet fra databasen ved innlasting av forsiden blir disse artiklene oppbevart i to forskjellige vektorer, en som representerer den originale listen av artikler i databasen og den andre vektoren er den som til alle tider blir vist frem i brukergrensesnittet. Når man søker blir visningsvektoren satt til å matche søkefrasen bruker skrev inn, og når de enten visker ut søketeksten eller trykker på logoen oppe til venstre blir visningsvektoren satt til å være den originale listen av artikler igjen.

4. Arkitektur

Dette kapittelet inneholder skisse av produktets arkitektur.

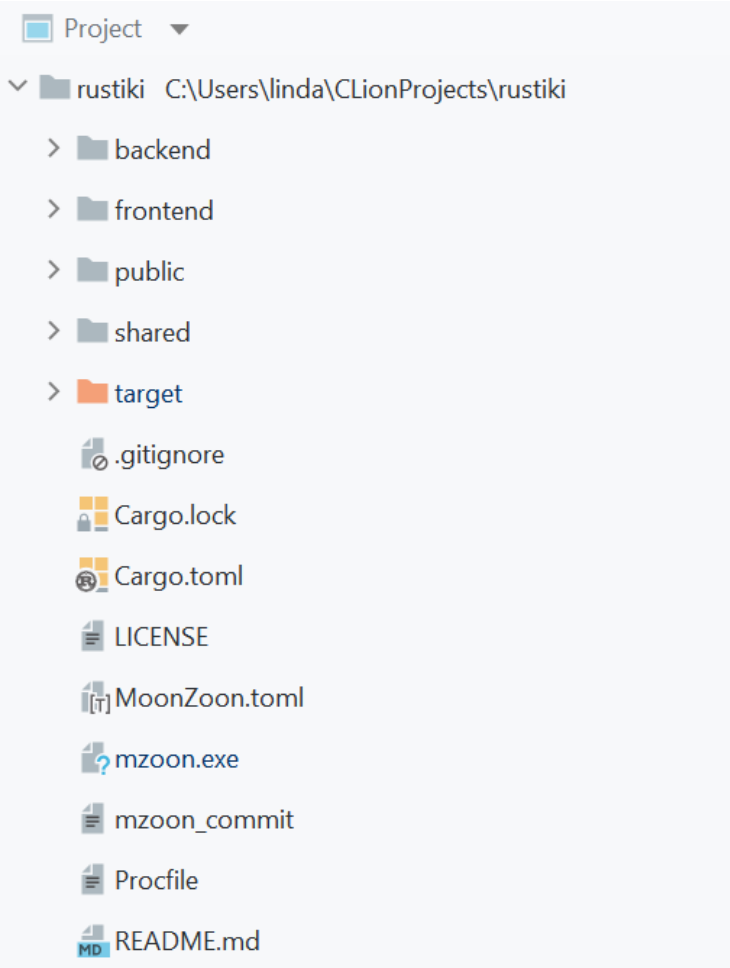


Figur 12: Arkitekturskisse.

Figur nummer 12 viser en skisse av produktets arkitektur.

5. Prosjektstruktur

Dette kapittelet viser og forklarer strukturen av prosjektet.



Figur 13: Oversikt over prosjektets struktur.

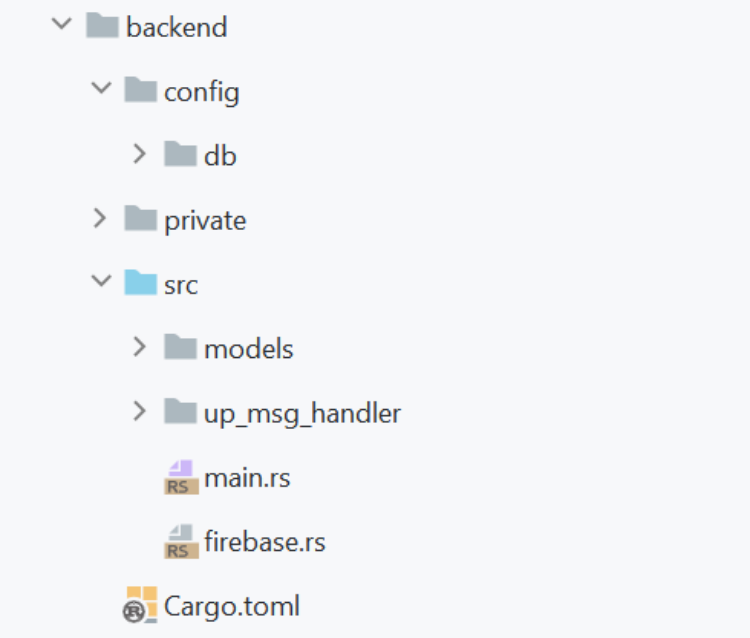
I Rust skiller man mellom en binær applikasjon og et bibliotek. Begge disse kan refereres til som en crate, som vil si roten av koden. Rusts prosjektstruktur gjøres opp av moduler. En modul er privat med mindre man bruker nøkkelordet «pub» når man deklarerer modulen. Moduler må opprettes ved å deklarere de i en eksisterende modul.

Når man lager et prosjekt i rammeverket MoonZoon vil dette inneholde fire mapper:

- «backend» – binær applikasjon.
- «frontend» – bibliotek.
- «public» – mappe for ressurser.
- «shared» – bibliotek.

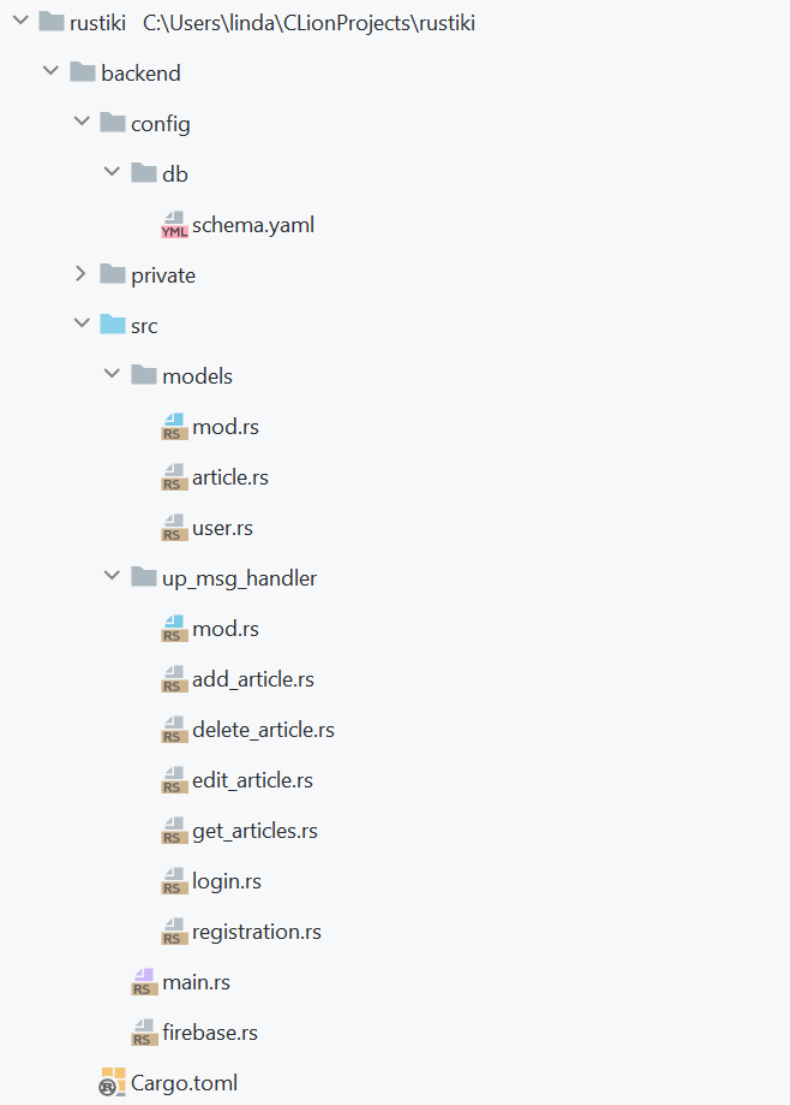
Et MoonZoon-prosjekt inneholder en «Cargo.toml» som står for å definere mappene «backend», «frontend» og «shared» som del av et og samme Rust-prosjekt. Filen «mzoon.exe» må ligge i prosjektets rot for å kunne kjøre prosjektet. Figur 13 viser den overordnede strukturen av gruppens prosjekt.

5.1 Backend



Figur 14: Strukturen av prosjektets crate «backend».

I gruppens prosjekt består backend av mappene «config», «private» og «src», som vist i figur 14. Da denne crate-en er en binær applikasjon, har den «main» som rot.



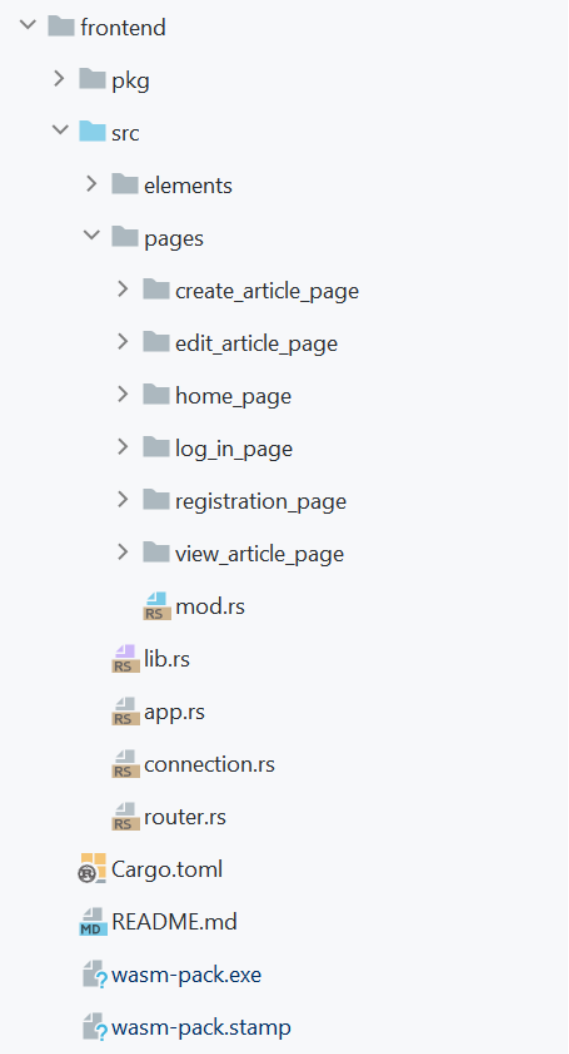
Figur 15: Alt innhold i prosjektets crate «backend».

Gruppen har opprettet mappen «config» for å ha en plass å legge en autogenerert schema-fil fra objekthåndteringsverktøyet Aragog, som kan sees i figur 15. Mappen «private» inneholder konfigurasjon for MoonZoon, og har ikke blitt modifisert av gruppen. Under «src» finner man «main»-modulen, «firebase»-modulen som henter en tilkobling til firebase, «models»-modulen som inneholder objekter som blir brukt, og up_msg_handler som inneholder handlers som har spesifikke oppgaver.

Funksjonalitet for selve modulen «models» blir definert i den underliggende filen «mod.rs». På samme vis blir funksjonaliteten for modulen «up_msg_handler» definert i underliggende fil «mod.rs». Man kan si at «mod» er roten av mappen. Dette er standard oppsett for Rust-moduler hvis de legges i mapper. Man er ikke nødt å legge moduler i mapper, men gruppen ønsket å gjøre det for økt orden.

Som vist i figur 15 inneholder «backend» en fil kalt «Cargo.toml». Denne filen brukes til å definere dependencies for denne crate-en.

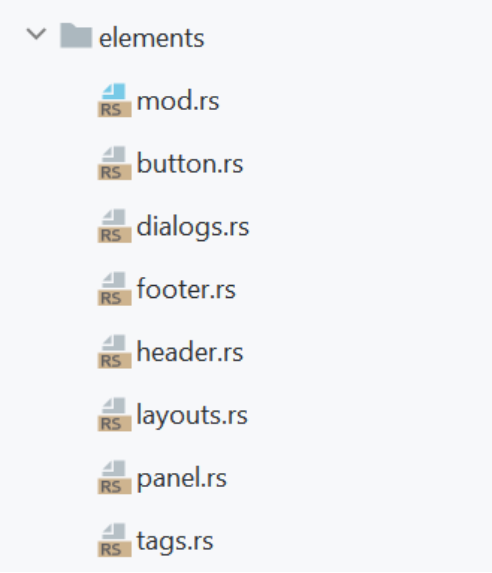
5.2 Frontend



Figur 16: Strukturen av prosjekts crate «frontend».

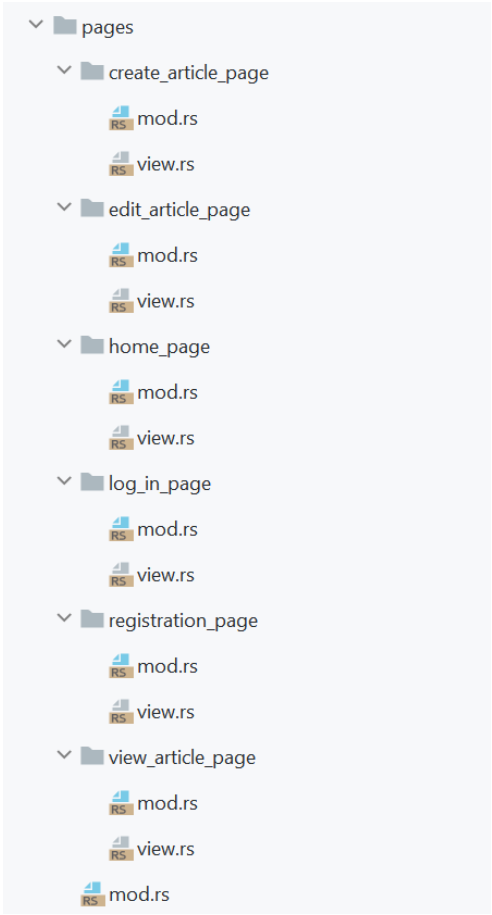
Prosjektets crate «frontend» består av mappene «pkg» og «src», som vises i figur 16. Mappen «pkg» inneholder MoonZoon-konfigurasjoner, og har ikke blitt modifisert av gruppen. Filen «Cargo.toml» definerer dependencies for crate-en «frontend». Roten av crate-en «frontend» er filen «lib». Grunnen til at denne filen ikke heter «main» - som den heter i crate-en «backend» - er at «frontend» er et bibliotek, mens «backend» er en binær applikasjon.

For å gjøre prosjektet mer oversiktlig har gruppen lagt to mapper i «src», som vist i figur 16; «elements» inneholder kode for visuelle elementer, og «pages» inneholder en mappe for hver av sidene til sin MoonZoon-applikasjon.



Figur 17: Innhold i mappen «elements».

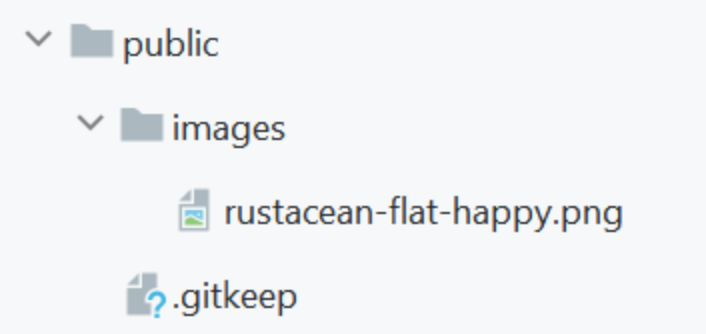
Mappen «elements» som ligger under frontend/src har filer som inneholder kode for visuelle elementer i gruppens applikasjon som ikke er spesifikke for én side. Modulen «mod» er rotmodulen til «elements». I figur 17 ser man de modulene gruppen har laget for visuelle elementer, som blant annet «button» og «tags».



Figur 18: Innholdet i «pages» sine underliggende mapper.

Mappen «pages» inneholder en mappe for hver av de fem forskjellige sidene til gruppens applikasjon, som vist i figur 18. Under hver av sidene ligger det to filer; «mod» og «view». Roten av mappen, «mod», inneholder logikken til siden, mens modulen «view» inneholder alt det visuelle.

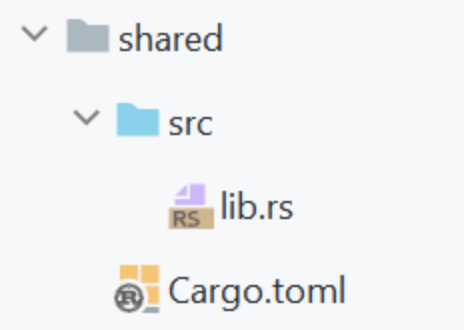
5.3 Public



Figur 19: Innholder i prosjektets mappe «public».

Mappen «public» i MoonZoon-prosjekter blir gjerne brukt til bilder eller stilark. Som man kan i på figur 19 har gruppen et bilde i den mappen.

5.4 Shared



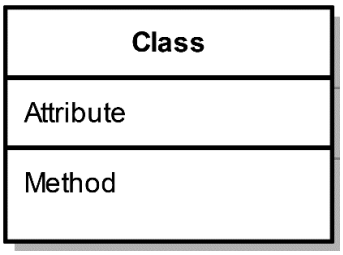
Figur 20: Innholdet i prosjektets mappe «shared».

Crate-en «shared» er et bibliotek, og inneholder derfor en fil kalt «lib» som sin rot. Man kan lage flere moduler eller mapper, men som vist i figur 20 har gruppen valgt å ha kun den ene filen, fordi de fleste MoonZoon eksempelprosjekter gjør det slik, og fordi filen inneholder lite kode.

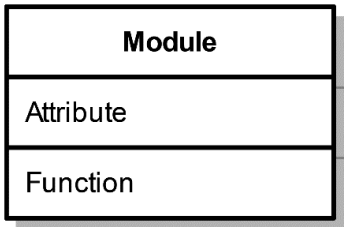
6. Strukturdiagram

Dette kapittelet inneholder et diagram som viser strukturen av produktets backend.

Programmeringsspråket Rust bruker ikke klasser, men moduler. Der finnes ingen standard måte å representere Rust-moduler som UML, så derfor har gruppen laget et strukturdiagram som ligner på klassediagrammer, men hvor klasse er erstattet av modul.

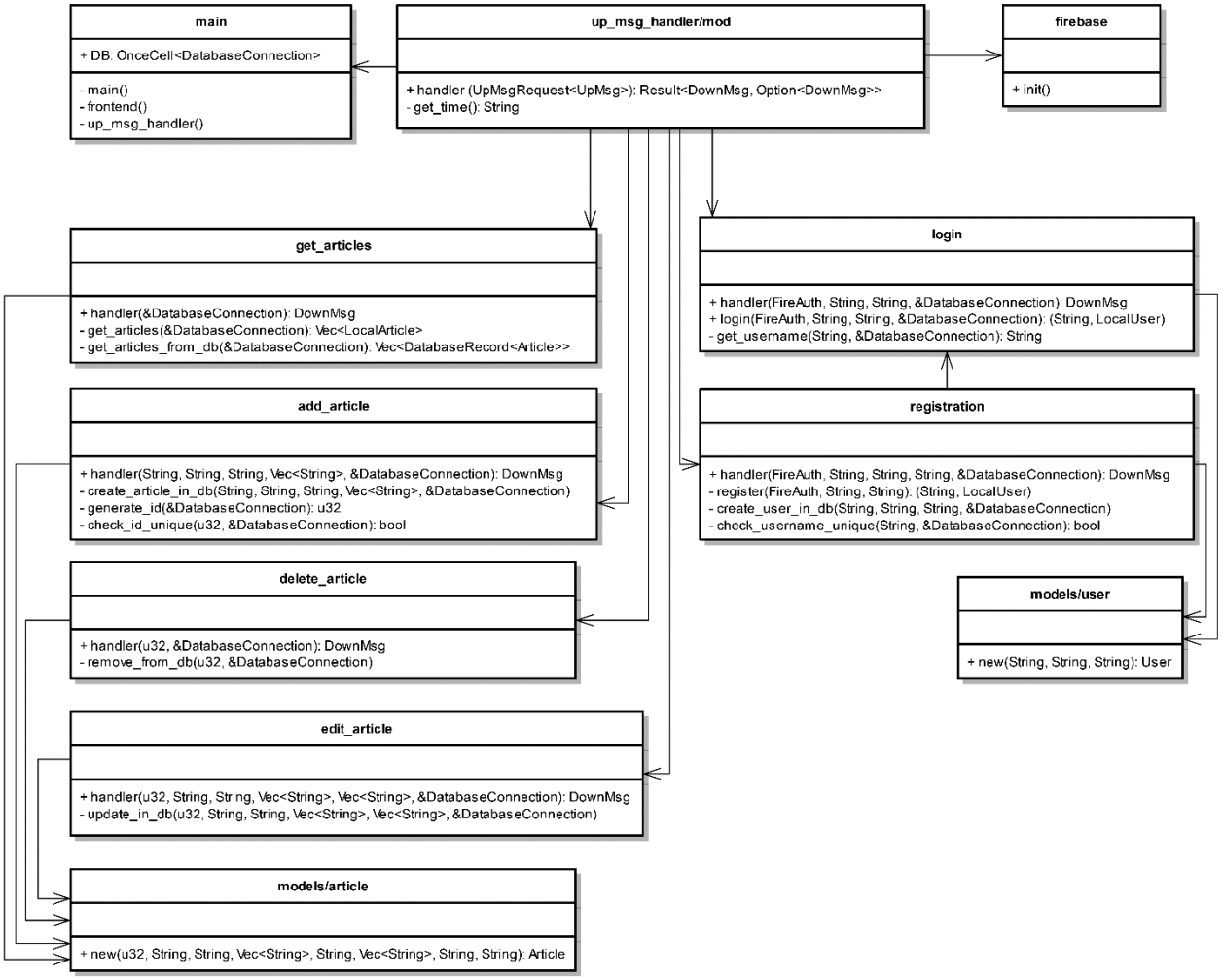


Figur 21: Strukturen av en klasse i et klassediagram.



Figur 22: En struktur bestående av først modulnavn, deretter attributter, og til slutt funksjoner.

Et klassediagram består vanligvis av klassens navn, eventuelle attributter, og til slutt funksjoner, som vist i figur 21. Gruppen velger å fremstille backend-delen av sitt Rust-prosjekt med strukturen vist i figur 22, bestående av modulnavn, attributter/felt, og til slutt funksjoner.



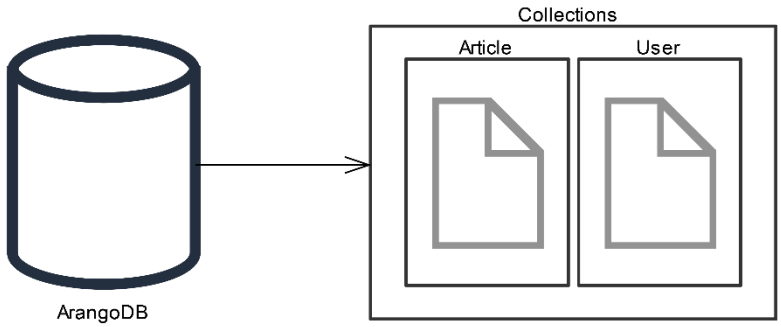
Figur 23: Strukturen av produktets backend.

Figur 23 viser strukturen av produktets backend. For å se figur 23 som vektorgrafikk, se filen kalt «structure_diagram.svg» i mappen «Diagrammer» i innlevert zip-fil.

7. Databasemodell

Dette kapittelet inneholder databasemodell for databasen gruppen brukte.

ArangoDB er en NoSQL-database. I ArangoDB må alle objekter innenfor en samling være av samme type, men hvilken type dette er må ikke defineres i databasen. Denne typen bestemmes ved å lage et struct i koden (struct definerer et objekt), og deretter begynne å lagre objekter av denne typen i samlingen.



Figur 24: Strukturen av gruppens database i ArangoDB.

```rust
#[derive(Debug, Serialize, Deserialize, Clone, Record)]
#[serde(crate = "serde")]
pub struct Article {
    pub id: u32,
    pub title: String,
    pub content: String,
    pub contributors: Vec<String>,
    pub author: String,
    pub tags: Vec<String>,
    pub created_time: String,
    pub updated_time: String,
}
```

Figur 25: Et struct som definerer hvordan objekter av typen Article er bygget opp.

```rust
#[derive(Debug, Serialize, Deserialize, Clone, Record)]
#[serde(crate = "serde")]
pub struct User {
    pub id: String,
    pub email: String,
    pub username: String,
}
```

Figur 26: Et struct som definerer hvordan objekter av typen User er bygget opp.

Gruppens prosjekt har to forskjellige samlinger i ArangoDB, som vist i figur 24. Dette er "Article" og "User". Disse objektene blir definert av structs i backend/src/models i gruppens prosjekt. Et «Article»-objekt, som vist i figur 25, inneholder et felt som heter "author" og inneholder brukernavnet til forfatter av artikkelen. Gruppen bruker altså brukernavn til å koble sammen en artikkel med en bruker. Et «User»-objekt – som vist i figur 26 – består av brukernavn, e-post og ID. Denne ID-en er den som blir generert av Firebase når en ny bruker registrerer seg. Vi velger å lagre denne ID-en som del av brukerobjektet i ArangoDB for å ha en måte å koble sammen autentisering med brukerinfo.

8. Server-tjenester

Dette kapittelet forklarer server-tjenester brukt i gruppens wikirammeverk.

```
pub fn connection() -> &'static Connection<UpMsg, DownMsg> {
    Connection::new(|down_msg, _cor_id| {
        match down_msg {
            // ------ Auth ------
            DownMsg::LoggedIn(user) => app::set_logged_in_user_and_token(user),
            DownMsg::LoginError(string) => log_in_page::set_login_error(string),
            DownMsg::RegistrationError(string) => registration_page::set_error_msg(string),

            // ----- Article -------
            DownMsg::Articles(vec) => home_page::set_articles(vec),
            DownMsg::ArticleAdded(_) => {
                home_page::get_articles();
                router().go(Route::Home);
            }
            DownMsg::ArticleUpdated => {
                home_page::get_articles();
                router().go(Route::Home);
            }
            DownMsg::ArticleRemoved => {
                home_page::get_articles();
                router().go(Route::Home);
            }
        }
    })
    .auth_token_getter(app::auth_token)
}
```

Figur 27: Kode i frontend/src/connection blir kjørt basert på hvilken DownMsg som mottas fra backend.

Rammeverket MoonZoon bruker en abstraksjon av REST-tjenester mellom backend og frontend. Denne abstraksjonen gjør at frontend kan sende en UpMsg til backend, mens backend kan svare med en DownMsg. UpMsg tilsvarer altså en HTTP request, og DownMsg tilsvarer responsen. I figur 27 ser man hvordan man i frontend kan håndtere DownMsg som mottas fra backend, ved å legge inn en eller flere funksjoner som skal kjøres for hver variant av DownMsg som mottas.

```rust
pub enum DownMsg {
    // ------ Auth ------
    LoggedIn(LocalUser),
    LoginError(String),
    RegistrationError(String),
    // ------Article-----
    Articles(Vec<LocalArticle>),
    ArticleAdded(String),
    ArticleUpdated,
    ArticleRemoved,
}
```

Figur 28: Enum-et DownMsg med konstantene som gruppen har definert.

UpMsg og DownMsg er enums, og ligger under "shared"-mappen i prosjektet, som er et bibliotek som blir importert av både frontend og backend. Under enum-ene UpMsg og DownMsg har gruppen definert forskjellige konstanter, for å kunne sende nødvendig data mellom frontend og backend. Figur 28 viser konstantene definert for DownMsg, og man kan se hvilken type data disse inneholder.

9. Sikkerhet

Dette kapittelet handler om produktets nivå av sikkerhet. Gruppens wikirammeverk er open source, og kommer til å bli videre utviklet før noen deployer en wiki bygget på rammeverket.

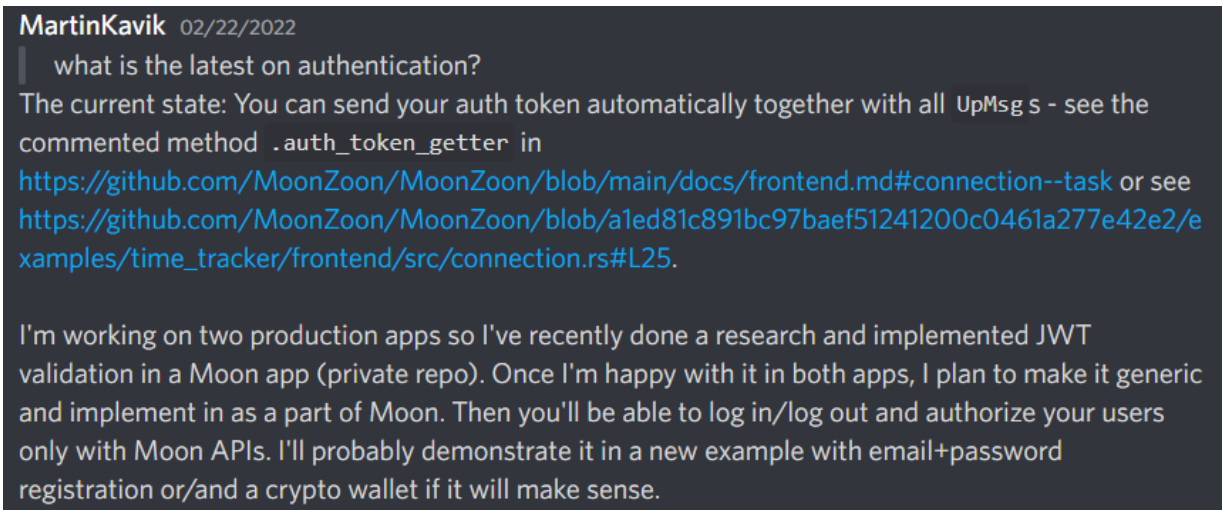
9.1 HTTPS

Når noen skal deploye en wiki som er bygget på gruppens wikirammeverk, er det viktig at de bruker HTTPS for å sikre at ikke innhold i requests blir sendt i klartekst. Dette kan ha konsekvenser dersom uvedkommende får fatt i innholdet, som for eksempel personers passord og e-post.

9.2 Passord

Wikirammeverket bruker Firebase Auth til registrering og innlogging av brukere. Firebase Auth lagrer e-post som klartekst, og lagrer en hashet versjon av brukerens passord, som ikke er synlig for administrator (Firebase Open Source, u.d.)

9.3 Access token



Figur 29: Hovedutvikleren bak MoonZoon forteller på MoonZoons Discord hvordan rammeverket ligger an med autentisering.

Wikirammeverket mottar access token fra Firebase som deretter blir lagret i frontend. Dette burde heller bli lagret i en cookie, da dette er sett på som best practice. Dette er lagt inn som en issue på prosjektets GitHub. Tokenet blir ikke validert. Som vist i figur 29 jobber hovedutvikleren bak MoonZoon med en utvidelse av den funksjonaliteten som rammeverket per nå har for tokens, og planlegger en abstraksjon for validering av JWT tokens. Det kan implementeres en egen løsning for tokens i gruppens wikirammeverk, men siden det er forventet videre utvikling før produktet brukes i deployet form kan det være fordelaktig å vente på funksjonaliteten i MoonZoon.

9.4 Parameter-injection

Ifølge ArangoDB (ArangoDB, u.d.) er ikke deres queryspråk AQL sårbart for injeksjon av parametere, men presiserer at en query fra klient som består av sammensatte strings kan være sårbar for injeksjon. På dette grunnlaget anbefaler ArangoDB at man bruker et verktøy for queries som separerer teksten i en query fra verdiene som er satt inn (ArangoDB, u.d.). Gruppens wikirammeverk benytter objekthåndteringsverktøyet Aragog, som altså hjelper å beskytte mot injeksjon.

9.5 Cross-site scripting

En ting som kan hjelpe mot cross-site scripting (XSS) er å filtrere brukerinput så strengt som mulig (PortSwigger, u.d.). Filtreringen av brukerinput er et punkt som kan forbedres i gruppens produkt. Dette er lagt til som en issue på prosjektets GitHub. Det å bruke riktig header kan også hjelpe å forhindre XSS, ved at man definerer hvilke typer innhold som er i en HTTP-respons (PortSwigger, u.d.). I tillegg er det mulig å forhindre alvorlighetsgraden av eventuelle XSS-angrep ved å ha en streng Content Security Policy (Weichselbaum, 2021). Det går ikke an å sette innholdstype eller definere en streng CSP for en applikasjon laget i MoonZoon uten at man utvider funksjonaliteten til rammeverket. Siden MoonZoon fortsatt er i en ganske tidlig utviklingsfase, er det enda mye som ikke er implementert. MoonZoons hovedutvikler forklarer i sin dokumentasjon (Kavík, 2022) at rammeverket ikke enda er ment å brukes til standalone applikasjoner. Hans prioritet er å gjøre ferdig MoonZoon Cloud, som vil tillate brukere å hoste en MoonZoon-applikasjon. Når denne løsningen er ferdigutviklet er det også planlagt å forbedre både CORS og andre sikkerhetstiltak på serversiden for å optimalisere MoonZoon (Kavík, 2022).

10. Installasjon og kjøring

Gruppen har laget for forskjellige instruksjoner for installasjon og kjøring – en kun for sensor, og en annen for alle brukere som ønsker å ta i bruk rammeverket. Gruppen gir sensor sine miljøvariabler, slik sensor får kjøre prosjektet med samme instanser av ArangoDB og Firebase som gruppen. Se vedlagt fil «Readme sensor» i mappen «Dokumenter».

Installasjonsinstrukser som er for alle andre brukere ligger tilgjengelig på prosjektets GitHub, og er levert sammen med kildekoden. Se filen «README.md» i roten av gruppens kildekode.

11. Dokumentasjon av kildekode

I dette kapittelet forklares det hvordan man får tilgang til dokumentasjon av kildekode.

```rust
//! Defines different reusable buttons.
use crate::router::router;
use crate::router::Route;
use zoon::named_color::*;
use zoon::*;

/// Returns a Button element
///
/// # Arguments
/// * `id` - A string slice that decides the (unique) HTML id of the element.
/// * `label_text` - A string slice that holds the text to be shown on the button label.
/// * `function` - A function to be called on key down. Function takes no arguments.
pub fn button(id: &str, label_text: &str, function: fn()) -> impl Element {
    let (hovered, hovered_signal) = Mutable::new_and_signal(false);
    Button::new()
        .id(id)
        .s(Font::new().size(16).color(GRAY_0))
        .s(Background::new().color_signal(hovered_signal.map_bool(|| GRAY_5, || GRAY_9)))
        .s(Padding::new().y(10).x(15))
        .on_hovered_change(move |is_hovered| hovered.set(is_hovered))
        .label(label_text)
        .on_click(function)
        .focus(true)
        .on_key_down_event(move |event| event.if_key(Key::Enter, function))
}
```

Figur 30: Utsnitt fra dokumentasjonen i prosjektets modul «button», som ligger under frontend/src/elements/button.

Programmeringsspråket Rust har sin egen standard for hvordan dokumentasjon skal skrives. Gruppen har fulgt denne standarden, som er forklart i Rusts offisielle dokumentasjon (Rust, u.d.). På øverste linjen i figur 30 ser man hvordan gruppen har dokumentert selve modulen «button». Litt lenger nede i figur 30 kan man se et eksempel på en dokumentert funksjon som tar inn tre parametere og returnerer et Button-element. Funksjoner som gjør mer kompliserte oppgaver, inneholder gjerne en lengre forklarende tekst enn funksjonen i figur 30.

Gruppen vil anbefale å se på dokumentasjonen inne i selve prosjektet. Det er mulig å generere et html-dokument med dokumentasjonen for koden som kan åpnes i nettleser, men å lese dokumentasjon på denne måten medfører to ulemper; man får kun se dokumentasjon for moduler og funksjoner som er deklarert med nøkkelord «pub» for

public, og man får med dokumentasjon for alle brukte biblioteker – inkludert hele MoonZoon-rammeverket. For å generere og åpne filen med dokumentasjon kan man i prosjektets terminal kjøre kommandoen «cargo doc --open».

12. Tester

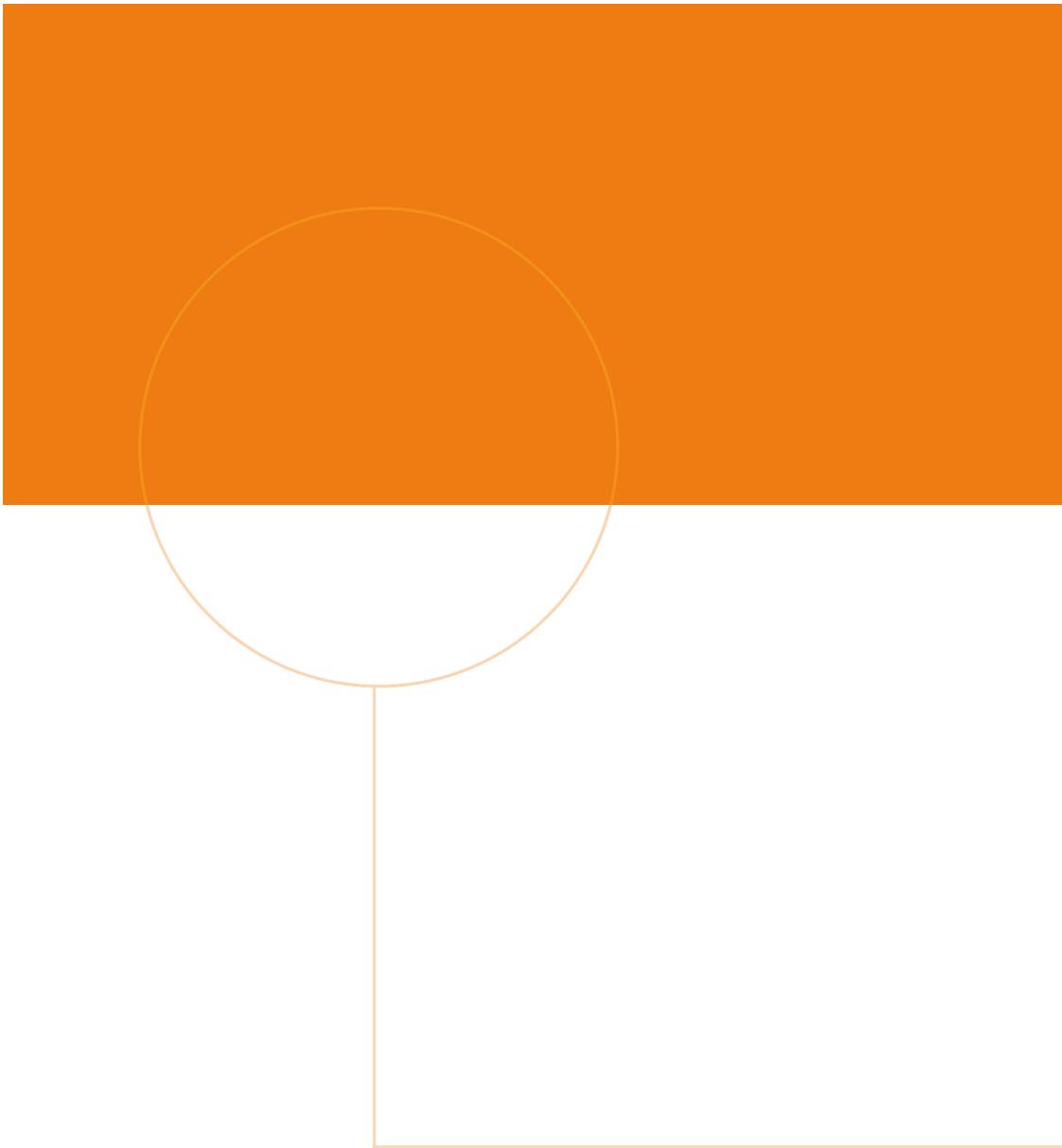
```rust
/// Tests for the test_username_unique and register functions.
/// For someone using the Rustiki framework, the values in these
/// tests must be customized to your own existing users in Firebase and ArangoDB.
#[cfg(test)]
mod registration_test {
    use super::*;
    use crate::firebase;
    use aragog::DatabaseConnection;
    use fireauth::FireAuth;

    /// Allows for async tests.
    /// Instead of using .await on a function, wrap in in aw!() instead.
    macro_rules! aw {
        ($e:expr) => {
            tokio_test::block_on($e)
        };
    }

    #[test]
    fn test_username_unique() {
        let conn = aw!(async {
            DatabaseConnection::builder()
                .with_schema_path("config/db/schema.yaml")
                .build()
                .await
                .unwrap()
        });
        let res = aw!(check_username_unique("test".to_string(), &conn));
        assert_eq!(res.clone(), false);
        println!("Expected 'false', got {}", res);
    }

    #[test]
    fn test_register() {...}
}
```

Figur 31: Testing av asynkrone funksjoner i modulen backend/src/up_msg_handler/register.rs.

Gruppen har lagt til tester på to av modulene i backend, for å få erfaring med testing i Rust og illustrere at det er mulig. En av disse testene vises i figur 31. Siden alle modulene i backend er knyttet opp mot queries til enten ArangoDB eller Firebase, vil ikke disse testene være overførbare til personer tar i bruk rammeverket og settes opp egne instanser av ArangoDB og Firebase Auth – med mindre de har identiske brukere og objekter som testene. Dette er dokumentert i koden. For å kjøre testene bruker man kommandoen «cargo test -- --nocapture». Delen «-- --nocapture» er ikke nødvendig, men gjør at man kan se eventuelle print-statements fra testen i terminalen.

13. Referanser

ArangoDB, u.d. *Common Errors.* [Internett]
Available at: https://www.arangodb.com/docs/stable/aql/common-errors.html
[Funnet 14 05 2022].
Firebase Open Source, u.d. *Firebase Authentication Password Hashing.* [Internett]
Available at: https://firebaseopensource.com/projects/firebase/scrypt/
[Funnet 15 05 2022].
Kavík, M., 2022. *Frontend - Zoon.* [Internett]
Available at: https://github.com/MoonZoon/MoonZoon/blob/main/docs/frontend.md
[Funnet 15 05 2022].
PortSwigger, u.d. *Cross-site scripting.* [Internett]
Available at: https://portswigger.net/web-security/cross-site-scripting
[Funnet 14 05 2022].
Rust, u.d. *Documentation.* [Internett]
Available at: https://doc.rust-lang.org/rust-by-example/meta/doc.html
[Funnet 15 05 2022].
Weichselbaum, L., 2021. *Mitigate cross-site scripting (XSS) with a strict Content Security Policy (CSP).* [Internett]
Available at: https://web.dev/strict-csp/
[Funnet 14 05 2022].



NTNU

Kunnskap for en bedre verden