Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vårsemesteret 2021 Åpen
Forfattere: Bjørn Kristian Teisrud, Oskar Skjærvø Gjølga	
Fagansvarlig: Hein Meling Veileder: Hein Meling	
Tittel på bacheloroppgaven: Quickfeed: Brukergrensesnitt med SwiftUI Engelsk tittel: Quickfeed: Desktop User Interface with SwiftUI	
Studiepoeng: 20	
Emneord: Brukergrensesnitt, QuickFeed, macOS, Swift, gRPC, Protobuf	Sidetall: 89 + vedlegg/kildekode: autograde/quickfeed-swiftui Stavanger 15. mai 2021

Universitetet
i Stavanger

QuickFeed: Brukergrensesnitt med SwiftUI

Bjørn Kristian Teisrud
Oskar Skjærvø Gjølga

Oppgave presentert for graden
Bachelor i ingeniørfag

15. Mai 2021

Institutt for data- og elektroteknologi
Det teknisk-naturvitenskapelige fakultet

Sammendrag

Denne oppgaven hadde som formål å lage en brukergrensesnitt-applikasjon for operativsystemet macOS. Målet var et alternativt brukergrensesnitt til rettesystemet QuickFeeds eksisterende web-applikasjon.

Oppgaven ble presentert av Institutt for data- og elektroteknologi ved Universitet i Stavanger. Oppgaven skulle løses med et utvalg teknologier for håndtering av selve brukergrensesnittet og kommunikasjonen med serveren. Utover det sto gruppen fritt til å velge hvordan oppgaven skulle gjennomføres, og hvordan brukergrensesnittet skulle se ut.

Programvaren rapporten omhandler er tilgjengelig på GitHub under autograde/quickfeed-swiftui.

Anerkjennelser

Vi vil gjerne takke vår veilder, Professor Hein Meling, for nyttige diskusjoner, hjelpsomme forklaringer, og gode råd angående applikasjonens design gjennom hele utviklingen av vår oppgave.

Innhold

Figurer

Kodeutdrag

Kapittel 1

Introduksjon

QuickFeed[7] (tidligere Autograder) er et programvaresystem som er utviklet på Universitetet i Stavanger, for automatisk retting av programmeringsoppgaver i datatekniske emner. Når en student leverer en innlevering kjøres et sett med automatiserte tester på oppgavene, og en tilbakemelding blir i løpet av kort tid tilgjengelig for studenten i brukergrensesnittet. Tilbakemeldingen gir en oversikt over hvilke deler av koden som tilfredsstiller kravene, og hvilke deloppgaver det eventuelt må jobbes ytterlige med. En innlevering kan enten konfigureres til automatisk godkjenning når studentene oppnår en fastsatt poenggrense, eller til godkjenning av faglærere og studentassistenter ved menneskelig verifisering. Faglærere og studentassistenter har oversikt over studentenes resultater, og har tilgang til funksjoner for å administrere studentene i emnet og emnet i seg selv. Den eksisterende klient-applikasjonen er skrevet i TypeScript, og rammeverket ReactJS håndterer brukergrensesnittet.

1.1 Oppgavebeskrivelse

Oppgaven går ut på å utvikle en klient-applikasjon for QuickFeed. Applikasjonen skal implementere funksjonaliteten som er tilgjengelig i den nåværende web-applikasjonen. Ytterlige funksjoner som forbedrer brukeropplevelsen kan legges til. Applikasjonen skal være et supplerende alternativ til web-applikasjonen for studenter og ansatte som benytter macOS til studie- eller arbeidsverktøy.

1.2 Mål

Målsetningen er at applikasjonen skal kunne kjøres på macOS. Muligheten for å utvide målgruppen med iOS-støtte skal også utforskes som en sekundær målsetning. Brukergrensesnittet skal være selvforklarende, og brukeren skal bruke minimalt med tid på å navigere mellom applikasjonens funksjonaliteter.

1.3 Motivasjon

QuickFeed har som resultat av en bacheloroppgaven *Experience Report - Replacing REST with gRPC in Autograder* [6] fra 2019, erstattet JSON/REST-APIer med gRPC for tjener-klient-kommunikasjon. Siden programmeringspråket Swift nylig har fått støtte for gRPC, er det interessant å utforske hvordan en Swift-applikasjon kan ta i bruk den eksisterende gRPC-tjenesten. En skrivebordsapplikasjon gir både muligheter og begrensninger kontra en web-applikasjon. Blant annet kan en skrivebordsapplikasjon i enkelte tilfeller gi en bedre arbeidsflyt, for eksempel ved at applikasjonen sender ut en notifikasjon når en test er fullført.

MacOS har lenge vært et populært operativsystem blant utviklere, og ifølge JetBrains undersøkelse *The State of Developer Ecosystem* [3] fra 2020 benytter 44% av respondentene operativsystemet. I 2020 annonserte Apple overgangen fra Intel-arkitekturen til ARM64-arkitekturen på selskapets maskiner. En applikasjon basert på Swift og SwiftUI kan kompileres til å kjøre

native på begge maskinvarearkitekturene, uten arkitekturspesifike tilpasninger.

Kapittel 2

Bakgrunn

Gjennom hele rapporten refereres det med jevne mellomrom til ulike programvareteknologier. Formålet med dette kapittelet er å gi leseren bakgrunnskunnskapene som behøves for å enklere forstå innholdet i de etterfølgende kapittelene.

2.1 Swift

Programmeringsspråket Swift ble lansert av Apple i 2014. Motivasjonen bak var å erstatte Objective-C med et mer moderne språk for Apples egne plattformer. Fra starten av var Swift Apple-proprietært, men i versjon 2.2 ble kildekoden åpnet. Siden den tid har Swift blitt videreutviklet av et samfunn av bidragsytere, med Apple som prosjektleder. Swift kan kjøres på macOS, Windows, og enkelte av de mest populære Linux-distribusjonene. På relativt kort tid har Swift tatt over som det dominerende programmeringsspråket for Apple-plattformene. Og blant respondentene i *The State of Developer Ecosystem*[3] som hadde enten Swift eller Objective-C som et av sine hovedspråk brukte 70% Swift, 17% Objective-C, og 13% både Swift og Objective-C.

2.1.1 Xcode

Xcode er Apples integrerte utviklingsmiljø (IDE) som kun er tilgjengelig på macOS, og inkluderer blant annet et bredt spekter av verktøy for feilsøking, tekstbehandling, og programvarebygging. Xcode er skreddersydd til utvikling av programvare med Swift og Objective-C for Apple-plattformene iOS, iPadOS, macOS, watchOS og tvOS. Swift-programvare kan også skrives med tredjeparts-programvare som AppCode og VSCode, men for å bygge programvaren til en kjørende applikasjon for Apple-plattformene kreves verktøy fra Xcode.

2.1.2 AppKit og UIKit

AppKit og UIKit er Apples eksisterende rammeverk for å utvikle grafiske brukergrensesnitt for macOS og iOS. Med rammeverkene kan brukergrensesnitt utvikles gjennom ren koding, eller ved hjelp av grafiske løsninger med Storyborards som er inkludert i Xcode. Rammeverkene bruker en imperativ syntaks, som vil si at elementer opprettes og manipuleres ved å programmatisk modifisere visningenes egenskaper. Kodeutdrag 2.1 viser hvordan en visning defineres imperativt i UIKit.

```
override func loadView() {
    view = UIView()
    view.backgroundColor = .white
    scoreLabel = UILabel()
    scoreLabel.text = score
    if approved{
        scoreLabel.color = .green
    } else{
        scoreLabel.color = .blue
    }
    view.addSubview(scoreLabel)
}
```

Kode 2.1: Eksempel på en imperativt definert visning i UIKit

2.1.3 SwiftUI

SwiftUI er Apples nye rammeverk for å utvikle grafiske brukergrensesnitt for alle Apples operativsystemer. SwiftUI bruker en deklarativ syntaks fremfor den imperative syntaksen som brukes i AppKit og UIKit. Med deklarativ syntaks vil man erklære hva en visning skal gjøre, gitt tilstanden man har ved opprettelse og ny innlasting. På grunn av dette er det enkelt å produsere gjenbrukbare SwiftUI-visninger, som generelt fører til en mer lesbar kode. Figur 2.2 viser hvordan en visning defineres deklarativt i SwiftUI. For å tilegne visninger stil og oppførsel brukes *modifikatorer* (Modifiers), som med *foregroundColor*-modifikatoren i Kodelistingen under.

```
struct SubmissionScore: View {
    var score: String
    @State private var approved: Bool
    var body: some View {
        Text(score)
          .foregroundColor(approved ?? .green : .blue)
    }
}
```

Kode 2.2: Eksempel på en deklarativt definert visning i SwiftUI

Xcode har innebygd støtte for forhåndsvisning av SwiftUI-visninger, hvor adskilte deler av brukergrensesnittet presenteres i en egen seksjon av utviklingsmiljøet. I forhåndsvisningen kan man også kan gjøre enkelte endringer visuelt med grafiske verktøy. Ved grafiske endringer vil koden endres

i samsvar med de grafiske endringene, og koden representerer derfor alltid visningens sannhetskilde.

2.1.4 Combine

Combine er et Swift-rammeverk utviklet av Apple for prosessering av hendelser over tid. Rammeverket inkluderer funksjonalitet for publisering av, og abonnering på verdier. Combine-APIen er kompatibel med SwiftUI-APIen, og i kombinasjon kan rammeverkene i stor grad forenkle utvikling av reaktive brukergrensesnitt.

2.2 Nettverkskommunikasjon

2.2.1 Protobuf

Protokollbuffere, eller protobuf er en mekanisme for serialisering av strukturert data som er utviklet av Google. Protobuf består av et *Interface Description Language* (IDL) og *protoc*-kompilatoren. Datastrukturer og tjenester defineres i en *proto*-fil, og med bruk av denne filen genererer *protoc*-kompilatoren strukturer og tjenester til flere programmeringsspråk.

QuickFeed bruker protobuf til å generere datastrukturer som lagres i en SQLite-database. Siden grensesnittet mellom server-applikasjonen og databasen består av en ORM (*Object-Relational Mapping*), er det proto-filen som definerer databasens design. Protobuf brukes også til serialisering av kommunikasjonen mellom serveren og web-klienten, siden protobuf-formatet er standarden i gRPC-kommunikasjon.

2.2.2 gRPC

Et *eksternt prosedyrekall* (RPC) er en kommunikasjonsteknikk som benyttes for å kalle en prosedyre i et annet adresserom. Et prosedyrekall utføres ved at en klient sender en forespørsel til en tjeneste på en server om å kjøre

en prosedyre, hvor den forespurte prosedyren gjennomføres, og et svar blir sendt tilbake til klienten.

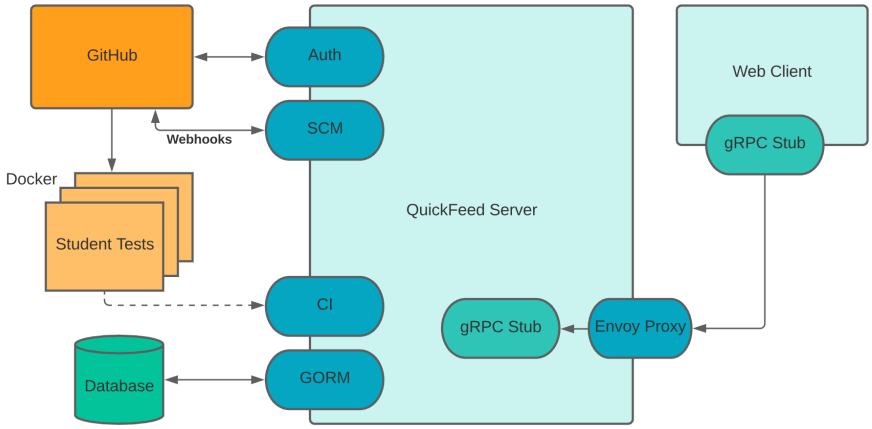
gRPC er et åpent rammeverk for eksterne prosedyrekall, som opprinnelig ble utviklet av Google. gRPC bruker protokollbuffere for serialisering av data, og dataen sendes over HTTP/2. En gRPC-tjeneste defineres i en proto-fil. Tjenestene består av gRPC-metoder som klienten bruker til kommunikasjon med serveren, og datastrukturer som brukes som input-argumenter og output-verdier i metodene. Proto-filen kompileres deretter til språkspesifik kode for bruk i støttede språk. I tillegg til typiske ensartede prosedyrekall har gRPC støtte for strømming av data, både fra server til klient og fra klient til server. Swift-implementasjonen av gRPC er skrevet med SwiftNIO i bunn.

2.2.3 SwiftNIO

SwiftNIO er et hendelsesdrevet rammeverk for nettverkskommunikasjon i Swift-programmer. NIO er ikke ment som et fullstendig web-rammeverk, men som et underliggende lavnivå-rammeverk for bruk i komplette løsninger [4]. Rammeverket inkluderer ulike datatyper for å abstraktere typiske kommunikasjonsmekanismer, som blant annet benyttes i gRPC-Swift.

2.3 QuickFeed

Den nåværende versjonen av QuickFeed er skrevet i to forskjellige programmeringsspråk. Server-siden er skrevet i språket Go og klient-siden er skrevet i TypeScript. Kommunikasjonen mellom server og klient foregår i protobuf-format (seksjon 2.2.1) gjennom generete gRPC-kall (seksjon 2.2.2). Serveren lagrer informasjonen i en SQLite-database. Figur 2.1 viser en oversikt over QuickFeeds arkitektur.



Figur 2.1: QuickFeeds nåværende arkitektur

QuickFeed bruker GitHub for håndtering av studentenes kildekode, publisering av innleveringsoppgaver, og autentisering. GitHub er en internettbasert utviklingsplattform som tilbyr distribuert versjonskontroll og kildekodeadministrasjon gjennom Git. I tillegg til versjonskontroll tilbyr også GitHub autentisering over *OAuth 2.0* og ulike samarbeidsfunksjoner. Samarbeidsfunksjonene gir studentene muligheten til å kunne jobbe sammen på innleveringer som utføres i grupper. GitHub har over 65 millioner registrerte brukere, og regnes som verdens mest brukte og avanserte programvareutviklingsplattformer. [5].

Brukere av QuickFeed autentiseres via GitHub. Autentiseringen mellom QuickFeed og GitHub foregår over *OAuth 2.0*-protokollen. OAuth 2.0 fungerer ved at applikasjonen kan autentisere brukere, men uten at OAuth 2.0 applikasjonen (GitHub) trenger å gi fra seg passordet til brukeren som skal

autentiseres. Med bruk av denne formen for autentisering trenger QuickFeed ingen form for passordhåndtering, og unngår sikkerhetsrisiko knyttet til lagring av kombinasjoner av brukernavn og passord. Når en bruker har blitt registrert, henter QuickFeed ut relevant informasjon som brukernavn, avatar-lenke, brukerid for GitHub, og tilgangsbevis for GitHub APIen.

En bruker kan melde seg opp til aktuelle emner i brukergrensesnittet. Når brukeren blir godtatt til emnet, blir det opprettet et datalager for innlevering av kode på GitHub. Serveren oppretter datalageret via GitHub APIen, i kombinasjon med brukerens lagrede tilgangbevis. Brukeren får også tilgang til et kodelager med emneinformasjon, og et kodelager hvor faglæreren publiserer oppgavebeskrivelser og startkode for hver innlevering.

Når en bruker laster opp kode til sitt eget kodelager på GitHub, trigges en handling på serversiden av en webhook. Denne handlingen laster ned koden over GitHub APIen, og tester koden opp mot tester definert av fagansvarlig. Testene kjøres i en Docker-Container, som betyr at testingen utføres i et isolert miljø. Docker-Containeren inneholder kun det som er nødvendig av ressurser for å kunne utføre testingen. Med bruk av Docker tilføres også et ekstra lag med sikkerhet rundt koden som kjøres, ettersom koden ikke kjøres direkte i serverens operativsystem. Etter koden er testet lagres relevant informasjon som peongsum, testut, og hvor mange tester som godkjennes og leveringsdato i en SQLite database på server siden.

Når en bruker åpner opp et klientvindu av QuickFeed-applikasjonen åpnes en gRPC-kobling mellom klienten og serveren. Med bruk av protobuf og gRPC kan QuickFeeds datatyper benyttes i applikasjoner som er bygd opp av ulike programmeringsspråk. Etter gRPC kanalen er opprettet mellom server og klient, kan klienten utføre gRPC-kall til server-siden. Alt av informasjon som vises på klient siden er hentet ut fra SQLite-databasen gjennom gRPC kall fra klienten til serveren.

Kapittel 3

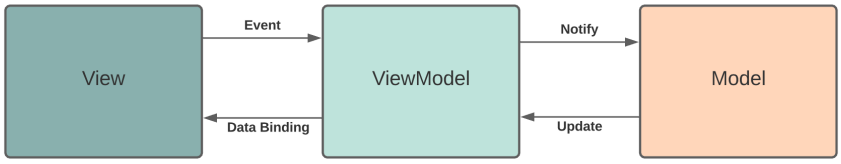
Design og Arkitektur

Valg av design og arkitekturmønster er viktig for å få en vedlikeholdbar kildekode. Med valg av et passende arkitekturmønster blir det lettere å gjøre endringer i koden, og feilsøking blir mindre arbeidsomt.

Vi vil i dette kapittelet forklare ulike aspekter som påvirker beslutninger relatert til programvarens design og arkitektur.

3.1 Model-View-ViewModel

Model-View-ViewModel (MVVM) er et lagdelt arkitekturmønster som tar for seg separeringen av det grafiske brukergrensesnittet (View), fra logikken på server-siden (Model) [2]. Visningslagets oppgave er begrenset til å presentere informasjon. En visningsmodell (ViewModel) fungerer som et bindeledd mellom visningen og modellen. Figur 3.1 viser hvordan informasjon og hendelser kommuniseres i en Model-View-ViewModel-arkitektur.



Figur 3.1: Dataflyt i Model-View-ViewModel

- **Model** - Består av datastrukturer og sammenhenger mellom strukturene.
- **View** - Har ansvar for utgjør strukturen, oppførselen, og utseendet for applikasjonens brukergrensesnitt. I tillegg trigges metoder i visningsmodellen ved brukerinteraksjoner som inndata og knappetrykk. Visningslaget avhenger av visningsmodellen.
- **ViewModel** - Har ansvaret for tilstanden av informasjonen som presenteres i visningslaget. Responderer til hendelser i visningen, kaller funksjoner som henter data fra modellaget, og publiserer data for bruk i visningslaget. Visningsmodellen avhenger kun av modellen.

For QuickFeed eksisterer modellaget allerede på serversiden, og derfor vil denne applikasjonen i hovedsak bestå av visningsmodeller og visninger. Beslutninger må likevel tas i forhold til hvordan modellaget skal tilgjengeliggjøres for visningsmodellen, og om det må gjøres optimaliseringer og tilpasninger i modellaget.

I en tradisjonell JSON-implementasjon av MVVM vil visningsmodellen dekode data i JSON-format fra modellaget til språkspesifike datastrukturer for visningslaget. I en gRPC-implementasjon vil ikke dekoding av data være nødvendig, siden protobuf håndterer serialisering og deserialisering av objektformatet.

3.2 Roller

Brukstilfellene av QuickFeed er nokså ulike for brukere med forskjellige bruker- og emneroller. Når en feil skal utbedres, eller en funksjon legges til, angår det ofte kun en spesifikk rolle. For å ta høyde for utfordringer i utviklingsprosessen og fremtidige endringer, er separasjon av funksjonalitet knyttet til de ulike rollene et viktig utgangspunkt i programvarens design.

3.2.1 Brukerroller

QuickFeed opererer med ulike roller som definerer brukerens adganger til informasjon og tjenester. For systemet i sin helhet kan brukeren enten ha administratorprivilegier eller ikke:

- **Bruker** - Kan registrere seg i tilgjengelige emner, og får emnespesifike tilganger i emner brukeren blir akseptert i.
- **Administratorbruker** - Kan i tillegg opprette og administrere emner, gi andre brukere administratorprivilegier, og administrere registrerte brukere. QuickFeeds vedlikeholdere og fagansvarlige har typisk denne rollen.

Uavhengig av hvilke emneroller en administrator har, skal administratorfunksjonene være tilgjengelige i brukergrensesnittet.

3.2.2 Emneroller

En bruker kan registreres og aksepteres i emner, og for hvert enkelt emne vil brukeren ha en emnerolle:

- **Teacher** - Kan administrere brukere på emnenivå, og har tilgang til studententes resultater i emnet. I et emne har som regel fagansvarlige og studentassistenter denne rollen.
- **Student** - Har kun tilgang til sine egne resultater i emner der brukeren er registrert som student. Kan opprette en gruppe der brukeren selv er medlem.
- **Pending** - Når brukeren har registrert seg for et kurs, men ikke enda har blitt akseptert av en bruker med Teacher-rollen.

I likhet med web-applikasjonen egner en sidebar seg for å navigere mellom applikasjonens funksjoner. En bruker kan ha Teacher-rollen i et emne, og samtidig ha Student-rollen i et annet emne. Derfor er det naturlig å separere grensesnittet ut fra hvilken rolle brukeren har i emnet som er valgt. En bruker skal alltid ha en emnevelger tilgjengelig, og med en gang valget endres skal resten av brukergrensesnittet lastes inn for det aktuelle emnet.

3.2.3 Rollebasert tilgang

På serversiden er enkelte gRPC-metoder kun tilgjengelig for brukere med admin-rollen og/eller teacher-rollen. Applikasjonen kan med utgangspunkt i de aktuelle metodene definere skillet mellom rollene i visningsmodellen. En visning som brukes eksklusivt for en av rollene, avhenger kun av visningsmodellen for den aktuelle rollen. Visningsmodellene for de ulike rollene implementerer kun metoder som samsvarer med de tilgjengelige gRPC-metodene for rollen. For eksempel skal ikke en liste med resultater for hele emnet eksistere i visningsmodellen som en studentvisning avhenger av.

3.2.4 Emnespesifikke anvendelser

Av emnene som benytter QuickFeed for å håndtere retting av innleveringer er det enkelte aspekter som skiller anvendelsen. Ulike emner har ulike behov, for eksempel er ikke alle typer programmeringsoppgaver egnet for automatisk retting og godkjenning. Manuell retting benyttes ikke i de fleste emnene på nåværende tidspunkt, og for å holde brukergrensesnittet oversiktlig bør menyer som angår rettelistene kun være tilgjengelige dersom emnet trenger det.

3.3 QuickFeeds databasemodell

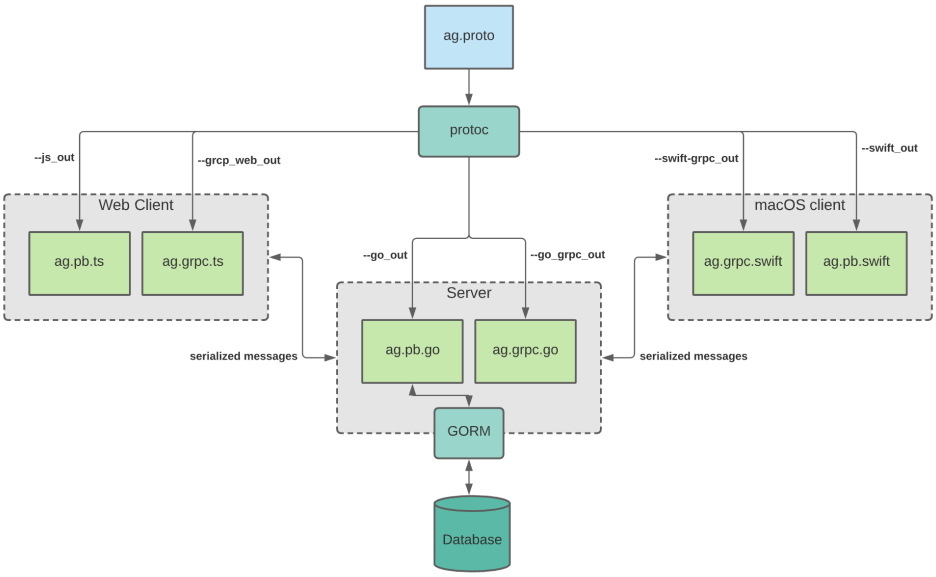
I QuickFeed brukes den samme filen til å generere datastrukturer i Go for server-siden, Typescript/Javascript for web-klienten, og Swift for denne applikasjonen. Overordnet definerer filen strukturen og relasjonene i databasen, men inneholder i tillegg enkelte strukturer som kun er ment for presentasjon i brukergrensesnittet. Grensesnittet mellom server-applikasjonen og databasen består av en object-relational mapping, der tabeller og relasjoner blir definert ut ifra de genererte Go-strukturene i en SQLite-database. Figur 3.2 viser en oversikt over anvendelsen av protobuf i QuickFeed.

Siden kommunikasjonen i hovedsak skal gå gjennom gRPC med dataen i protobuf-format, er det naturlig at applikasjonen bygges rundt de genererte modellene.

```
1  message User {
2      uint64 ID = 1;
3      bool isAdmin = 2;
4      string name = 3;
5      string studentID = 4;
6      string email = 5;
7      string avatarURL = 6;
8      string login = 7;
9      repeated RemoteIdentity remoteIdentities = 8;
10     repeated Enrollment enrollments = 9;
11 }
```

Kode 3.1: Proto-definisjon for User (ag.proto)

3.3 QuickFeeds databasemodell



Figur 3.2: Oversikt over anvendelsen av protobuf i QuickFeed

Når protoc kjøres med *proto-gen-swift* og ag.proto som input, genereres Swift-koden vist i kodeutdrag 3.2. Data sendes til og fra serveren i det genererte formatet, og man unngår repetitiv kode som typisk dekoder data i JSON-format til Swift-objekter og motsatt.

```
struct User {
  var id: UInt64 = 0
  var isAdmin: Bool = false
  var name: String = String()
  var studentID: String = String()
  var email: String = String()
  var avatarURL: String = String()
  var login: String = String()
  var remoteIdentities: [RemoteIdentity] = []
  var enrollments: [Enrollment] = []
  var unknownFields = SwiftProtobuf.UnknownStorage()
  init() {}
}
```

Kode 3.2: Swift-kode generert av protoc (ag.pb.swift)

3.4 Dataflyt i Applikasjonen

En protokoll er Swifts variant av et grensesnitt (Interface). En klasse eller struktur i Swift sies å *samsvare med* (conforms to) en protokoll, dersom den implementerer metodene og instansvariablene som er definert i protokollen.

For å håndtere egenskaper i en datatype som representerer typens tilstand, er det i Swift vanlig å ta i bruk *egenskapswrappere* for å erklære logikk som trigges når egenskapen endres.

For å håndtere publisering av data fra visningsmodellen til visningen, benyttes protokoller og wrappere fra *Combine*-rammeverket:

- **ObservableObject** - En klasse som samsvarer med ObservableObject-protokollen kan observeres av en annen struktur eller klasse. En SwiftUI-visning som observerer et slikt objekt vil lastes inn på nytt når enkelte verdier endres i objektet.
- **@Published** - En observatør av et ObservableObject vil kun ta i mot oppdateringer reaktivt når en instansvariabel med @Published-wrapperen i det observerte objektet endres.

For å håndtere tilstanden av data i visningsstrukturenes livssyklus, inkluderer SwiftUI-APIen wrappere som definerer hvordan strukturens egenskaper responderer på ny innlasting:

- **@ObservedObject** - Brukes i en visning for å definere et ObservableObject som skal observeres.
- **@State** - En variabel der tilstanden er eid av visningsstrukturen den er definert i, og som dermed beholder sin verdi når visningen lastes inn på nytt.
- **@StateObject** - En kombinasjon av @State og @ObservedObject. En visning med en visningsmodell markert med @StateObject vil ha eierskap til visningsmodellen.

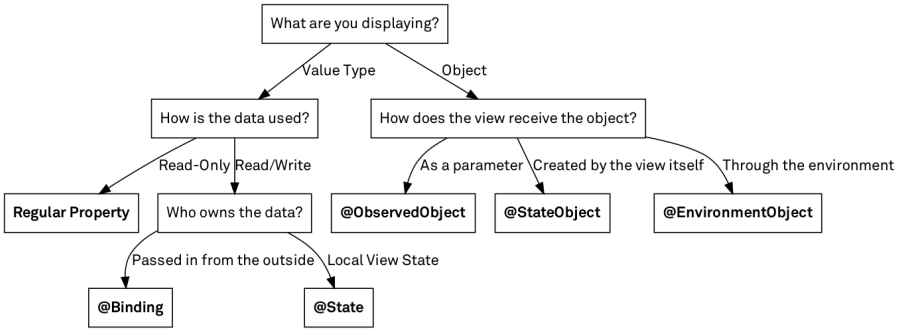
- **@Binding** - En variabel som ikke har eierskap til variabelens verdi i visningen den er definert i, men refererer til en @State-variabel i en forelder.
- **@EnvironmentObject** - Fungerer på samme måte som @ObservedObject ved at endringer vil laste inn visningen på nytt. Et @EnvironmentObject settes i miljøet ved med bruk av modifikatoren *.environmentObject()*. Et @EnvironmentObject kan dermed tilgjengeliggjøres for alle visninger i en foreldrevisning, fremfor å sendes inn som parameter for hver enkelt visning.

Kodeutdrag 3.3 viser en overordnet forklaring på hvordan en visning observerer en visningsmodell. Når visningsmodellens liste over innleveringer oppdateres, vil visningen bli oppdatert siden listen publiseres til visningen. Ut i fra navngivingen kan det virke som en implementasjon av *Observatør-mønsteret*, men implementasjonen ligner mer på designmønsteret *Publish-Subscribe* (pubsub). Visningsmodellen publiserer data for bruk i visninger som avhenger av den. I Publish-Subscribe har ikke subjektet (Objektet som publiserer) kjennskap til mottakerene, som subjektet vil ha i en implementasjon av *Observatør-mønsteret*.

```
1  import Combine
2
3  class ViewModel: ObservableObject{
4      @Published var assignments: [Assignment] = []
5      func loadAssignments(){
6          ....
7      }
8  }
9
10 import SwiftUI
11
12 struct AssignmentList: View {
13     @ObservedObject var viewModel: ViewModel
14     var body: some View {
15         List{
16             ForEach(viewModel.assignments, id: \.self){
       assignment in
17                 ...
18             }
19         }
20     }
21 }
```

Kode 3.3: Publisering av data fra visningsmodell til visning

Figur 3.3 forklarer brukstilfellene av egenskapswrappere i SwiftUI.



Figur 3.3: Beslutningstre som forklarer anvendelsen av egenskapswrappere i SwiftUI

[1]

3.5 Manuell retting

I QuickFeed kan innleveringer rettes manuelt ved bruk av en liste med kriterier. Når en student har levert en innlevering, rettes innleveringen av en eller flere brukere med teacher-rollen. Hvert enkelt kriterium i listen godkjennes eller underkjennes av retteren, og kommentarer kan legges til for hvert kriterium. Basert på andelen godkjente kriterier blir innleveringens poengsum kalkulert. Rettingen markeres som klar til utlevering etter at alle kriteriene er rettet.

En separat del av brukergrensesnittet håndterer utlevering og karaktersetting av ferdigrettede innleveringer. Utleveringen gjøres typisk av faglæreren, som velger om innleveringen godkjennes, underkjennes, eller må leveres på nytt ut fra den ferdige rettelisten.

Kodeutdrag 3.4 viser relevante strukturer som benyttes for manuell retting.

```
1
2 message GradingBenchmark {
3     uint64 ID = 1;
4     uint64 assignmentID = 2;
5     string heading = 3;
```

```
6       string comment = 4;
7       repeated GradingCriterion criteria = 5;
8   }
9
10  message Benchmarks {
11      repeated GradingBenchmark benchmarks = 1;
12  }
13
14  message GradingCriterion {
15      enum Grade {
16          NONE = 0;
17          FAILED = 1;
18          PASSED = 2;
19      }
20      uint64 ID = 1;
21      uint64 points = 2;
22      uint64 benchmarkID = 3;
23      string description = 4;
24      Grade grade = 5;
25      string comment = 6;
26  }
27
28  message Review {
29      uint64 ID = 1;
30      uint64 submissionID = 2;
31      uint64 reviewerID = 3;
32      string review = 4; // JSON encoded grading criteria
33      string feedback = 5;
34      bool ready = 6;
35      uint64 score = 7;
36      repeated GradingBenchmark benchmarks = 8 [(gogoproto.
        moretags) = "sql:\"-\""];
37      string edited = 9;
38  }
39
40  message Reviewers {
41      repeated User reviewers = 1;
42  }
43
44  message ReviewRequest {
45      uint64 courseID = 1;
46      Review review = 2;
47  }
```

Kode 3.4: Strukturer tilknyttet manuell retting

Kriteriene lastes inn i systemet i form av en JSON-fil. QuickFeed leser inn filen og lagrer den i databasen som en liste av *GradingBenchmark*-strukturen.

Ved opprettelse av en retting tar man utgangspunkt i denne strukturen og initialiserer en *Review*-struktur på klient-siden. RPCene som benyttes i tilknytning til manuell retting er vist i figur 3.5.

```
1 rpc CreateReview(ReviewRequest) returns (Review) {}
2 rpc UpdateReview(ReviewRequest) returns (Void) {}
3 rpc GetReviewers(SubmissionReviewersRequest) returns (Reviewers)
      {}
4 rpc LoadCriteria(LoadCriteriaRequest) returns (Benchmarks) {}
```

Kode 3.5: RPCer for manuell retting

Som resultat av RPCen *CreateReview* lagres strukturen i databasen. Ved endringer av rettingen på klientsiden oppdateres databasen med *UpdateReview*.

Majoriteten av logikken knyttet til manuell retting er implementert på klient-siden. Serveren sin rolle er i stor grad begrenset til lagring av strukturene fra klient-siden i databasen.

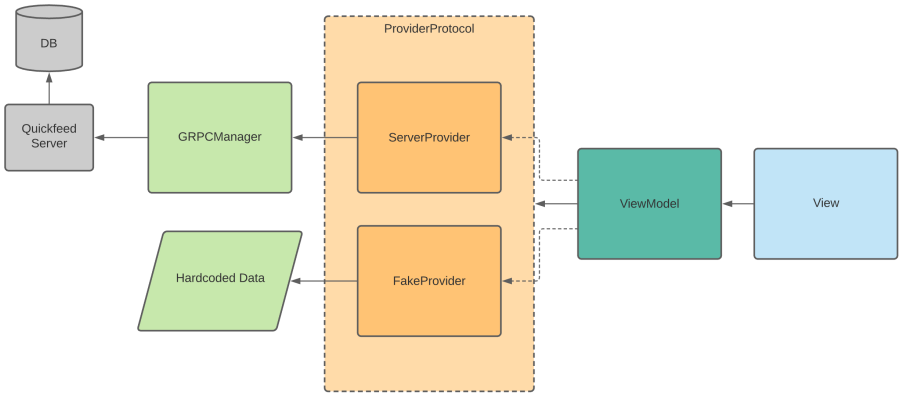
Kapittel 4

Implementasjon

Dette kapittelet tar for seg utviklingen av applikasjonen, problemer som oppstod i utviklingsprosessen, og valg som ble gjort angående programvarearkitekturen.

4.1 Programvarestruktur

Vår opprinnelige plan var å bruke *avhengighetsinjeksjon* (Dependency Injection) for visningsmodellenes datakilder. I en slik implementasjon kan brukergrensesnittet enten kjøres med data fra serveren, eller med hardkodet data. Figur 4.1 viser avhengighetene fra visningslaget til modellaget, med avhengighetsinjeksjon mellom visningsgslaget og modellaget.



Figur 4.1: Avhengighetene fra visningslaget til modellaget

ProviderProtocol definerer alle metodene en visningsmodell skal ha tilgjengelig. En visningsmodell avhenger av denne protokollen for å hente data til visningslaget, og enhver klasse som samsvarer med protokollen kan brukes.

Motivasjonen bak *FakeProvider* var å ha muligheten til å kjøre klientapplikasjonen uten kommunikasjon med en server. Planen var å hardkode objekter av alle de ulike strukturene fra *ag.pb.swift* for testing av brukergrensesnittet i oppstartsfasen. Denne løsningen fungerte bra til å komme i gang med rammene rundt brukergrensesnittet, men ble tidlig komplisert og dermed tidkrevende å vedlikeholde. Som et alternativ ble det satt opp en lokal versjon av serveren med en gammel databasefil for bruk i utviklingsprosessen.

Den alternative datakilden *ServerProvider* samsvarer med *ProviderProtocol*, men istedet for hardkodede data holder den en instans av typen *GRPCManager* som brukes til å kommunisere med serversiden.

4.2 Kommunikasjon over gRPC

For å kommunisere med serveren må Swift-filene for protokollbuffere og gRPC-tjenesten genereres. Med bruk av protoc-kompilatoren, protoc-gen-

swift, og den samme proto-filen som på serversiden som input, ble filene *ag.pb.swift* og *ag.grpc.swift* generert. For å ta i bruk filene i applikasjonen må grpc-swift og SwiftNIO legges til som pakkeavhengigheter i Xcode-prosjektet.

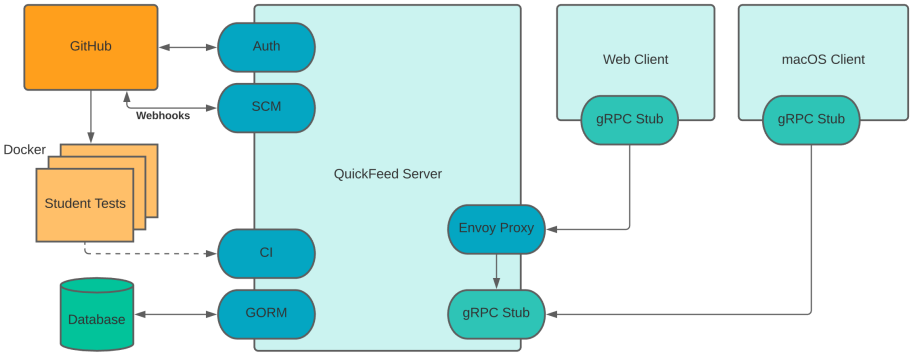
4.2.1 Konfigurasjon

Alt relatert til kommunikasjon mot serversiden gjennom gRPC er implementert i klassen *GRPCManager* (Kodeutdrag 4.1). En kobling til localhost på portnummer 9090 brukes i opprettelsen av en klient for kommunikasjon med serveren kjørende lokalt. For å håndtere innkommende kommunikasjonshendelser fra serversiden asynkront bruker gRPC-Swift en *EventLoopGroup*, som er en datatype fra SwiftNIO-rammeverket. I den genererte filen *ag.grpc.swift* defineres klassen AutograderServiceClient, med de tilhørende gRPC-metodene som brukes for å kommunisere med serveren. HPACKHeaders inkluderes i hvert metodekall for å identifisere brukeren, og inneholder en brukerid som serveren bruker til å avgjøre om brukeren har tilgang til den aktuelle metoden basert på brukerrolle og emnerolle.

```
1  import NIO
2  import GRPC
3  import NIOHPACK
4
5  class GRPCManager {
6      let eventLoopGroup: EventLoopGroup
7      let channel: ClientConnection
8      let quickfeedClient: AutograderServiceClient
9      let headers: HPACKHeaders
10
11     private init(){
12         let hostname = "localhost"
13         let port = 9090
14         self.eventLoopGroup = MultiThreadedEventLoopGroup(
       numberOfThreads: 1)
15         self.channel = ClientConnection.insecure(group: self.
       eventLoopGroup).connect(host: hostname, port: port)
16         self.quickfeedClient = AutograderServiceClient(channel:
       channel)
17     }
18 }
```

Kode 4.1: Konfigurasjon av gRPC for kommunikasjon med lokal server

I web-implementasjonen av gRPC kreves *envoy-proxy* til videresending av gRPC-spørringer fra brukerens nettleser til serveren. For gRPC-trafikk mellom serveren og Swift-klienten er ikke envoy nødvendig, og med macOS-klienten inkludert ender man opp med en overordnet arkitektur som vist i figur 4.2.



Figur 4.2: QuickFeeds overordnede arkitektur med macOS-klienten

4.2.2 Asynkron henting av data

For å forhindre at brukergrensesnittet fryser imens applikasjonen venter på data fra serveren, er det for enkelte prosedyrekall viktig at data oppdateres asynkront. Et prosedyrekall returnerer et svar i form av SwiftNIO-typen *EventLoopFuture*.

En *EventLoopFuture* er en fremtidig lovnad om et svar fra serversiden, som returneres i det øyeblikket gRPC-prosedyren kalles. Ved å pakke ut lovnaden asynkront i visningsmodellen, vil brukeren kunne fortsette å bruke applikasjonen frem til resultatet kommer, og brukergrensesnittet oppdateres reaktivt.

```
1 func loadEnrollmentLinks(){
2     let response = self.provider.getSubmissionsByCourse(courseId
      : self.currentCourse.id, type: SubmissionsForCourseRequest.
      TypeEnum.all)
3     _ = response.always {(response: Result<CourseSubmissions,
      Error>) in
4         switch response {
```

```
5           case .success(let response):
6               DispatchQueue.main.async {
7                   self.enrollmentLinks = response.links
8               }
9           case .failure(let err):
10              print("[Error] Connection error or enrollments not
    found: \(err)")
11              self.enrollmentLinks = []
12          }
13      }
14  }
```

Kode 4.2: Asynkron oppdatering av visningsmodellens tilstand

4.3 AppKit-visninger i SwiftUI

SwiftUI er enda på et tidlig stadie, og visningene som rammeverket tilbyr viste seg og ikke dekke alle behovene for applikasjonen. For en bruker med teacher-rollen er det viktig å ha søkefunksjonalitet for navigasjon i lister med mange studenter. Å filtrere en liste kan gjøres med en tekstfeltvisning, men det er foreløpig ikke mulig å modifisere tekstfeltet til å se ut som et søkefelt.

I SwiftUI kan visninger fra det mer modne rammeverket AppKit inkluderes for å løse nevnte utfordringer. For å bruke en Appkit-visning i SwiftUI lager man en type som samsvarer med *NSViewControllerRepresentable*-protokollen. Tilsvarende kan en AppKit-applikasjon ta i bruk SwiftUI-visninger som samsvarer med *NSHostingController*-protokollen. Kodeutdrag 4.3 inneholder en AppKit-visning som kan inkluderes i en SwiftUI-visning.

```
1  import SwiftUI
2  import AppKit
3
4  struct SearchField: NSViewControllerRepresentable {
5      @Binding var query: String
6
7      func makeNSViewController(
8          context: NSViewControllerRepresentableContext<
    SearchField>
9      ) -> SearchFieldController {
10         return SearchFieldController(query: $query)
```

```
11      }
12
13      func updateNSViewController(
14          _ nsViewController: SearchFieldController,
15          context: NSViewControllerRepresentableContext<
    SearchField>
16      ) {
17      }
18  }
```

Kode 4.3: AppKit-visning som kan brukes i SwiftUI

Visningen kan dermed anvendes i en SwiftUI-visning, på samme måte som andre SwiftUI-visninger (Kodeutrag 4.4).

```
1  import SwiftUI
2
3  struct ContentView: View {
4      @State var searchQuery: String = ""
5      var body: some View {
6          SearchField(query: $searchQuery)
7              .frame(minWidth: 200, maxWidth: 350)
8      }
9  }
```

Kode 4.4: Inkludering av AppKit-visningen i SwiftUI

4.4 Optimalisering av resultatinnlasting

En faglærer/studentassistent i et emne kan se en oversikt over resultatene alle studentene har oppnådd for alle innleveringene i emnet. Resultatet for en innlevering er presentert med en poengsum i prosent, og en statusfarge som indikerer om innleveringen er godkjent, underkjent, eller må leveres på nytt.

For å laste denne siden brukes RPCen *GetSubmissionsByCourse* som henter en struktur med alle detaljene om alle studentenes innleveringer (Kodeutdrag 4.5). I enkelte emner med mange fag kan denne oversikten ta lang tid å laste. Dette skyldes at *Submission*-strukturen inkluderer hele rapporten (*buildInfo*) som blir produsert når serveren tester innleveringen. Denne rap-

porten kan bli svært lang, og i et emne med over hundre studenter blir mye data sendt over nettet.

```
message Submission {
    enum Status {
        NONE = 0;
        APPROVED = 1;
        REJECTED = 2;
        REVISION = 3;
    }
    uint64 ID = 1;
    uint64 assignmentID = 2;
    uint64 userID = 3;
    uint64 groupID = 4;
    uint32 score = 5;
    string scoreObjects = 6;
    string buildInfo = 7;
    string commitHash = 8;
    bool released = 9; // defines whether the feedback is
    visible for the student or group members
    Status status = 10;
    string approvedDate = 11;
    repeated Review reviews = 12;
}

rpc GetSubmissionsByCourse(SubmissionsForCourseRequest) returns
    (CourseSubmissions) {}
```

Kode 4.5: Strukturen Submission og RPCen GetSubmissionsByCourse (ag.proto)

En løsning på problemet er å kun hente nødvendige detaljer for presentasjon av oversikten. Hele stukturen inkludert buildInfo lastes først når man velger en innlevering for nærmere inspeksjon. Løsningen ble implementert ved å legge til feltet *skipBuildInfo* i *SubmissionsForCourseRequest* (Kodeutdrag 4.6), som er input-argumentet i *GetSubmissionsByCourse*. Videre logikk er implementert på server-applikasjonen, slik at når skipBuildInfo er sann vil verdien i *buildInfo*-feltet for hver submission være en tom streng.

```
message SubmissionsForCourseRequest {
    enum Type {
        ALL = 0;
        INDIVIDUAL = 1;
        GROUP = 2;
    }
    uint64 courseID = 1;
    Type type = 2;
    bool skipBuildInfo = 3;
```

```
10 }
```

Kode 4.6: Strukturen SubmissionsForCourseRequest (ag.proto)

Tabell 4.1 viser en oversikt med lastetider målt med *DispatchTime*-datatyper i Swift. I en slik måling vil det alltid være variable faktorer som ressursbruk på egen maskin og lignende som spiller inn, men spesielt i emnet DAT320 kan man se en markant tidsreduksjon. Sammenlignet med DAT310 hadde DAT320 flere studenter, og mer utstrakt bruk av automatiserte tester. Reduksjonen i lastetid for DAT310 ble dermed ikke like merkbar.

Emne	Lastetid med buildInfo	Lastetid uten buildInfo	Studenter
DAT310	0.8s	0.3s	120
DAT320	5.8s	0.4s	180

Tabell 4.1: Sammenligning av lastetid med og uten buildInfo

Kapittel 5

Resultater og diskusjon

To av våre hovedmål (seksjon 1.2) var rettet mot designet av brukergrensesnittet. Applikasjonen skulle være selvforklarende, og brukeren skulle benytte minimalt med tid for å navigere seg rundt i applikasjonen. Disse målene har vært vårt hovedfokus gjennom hele utviklingen av brukergrensesnittet.

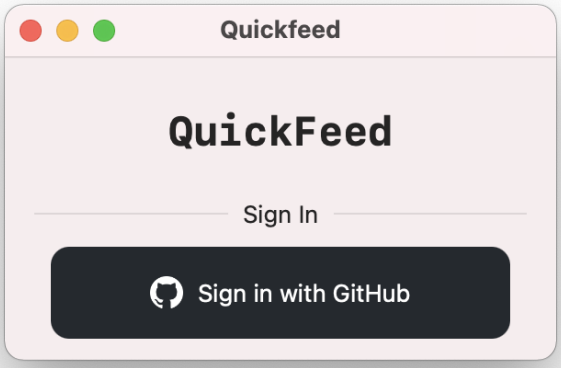
Vi vil i dette kapittelet ta for oss hvert hovedbrukergrensesnitt av QuickFeed. Under hvert brukergrensesnitt vil vi kort forklare designets oppbygging i form av tekst og bilde. I tillegg vil vi også forklare brukergrensesnittets funksjonalitet i form av tekst, bilde og kodeutklipp. Alle bildene, av applikasjonen, viser applikasjonen i lyst modus. QuickFeed-applikasjonen har også støtte for mørkt modus. QuickFeed bruker maskinens systeminnstillinger for å finne frem til hvilken modus som skal benyttes.

5.1 LogIn

Brukergrensesnittet for å logge inn på applikasjonen er det første som møter brukeren. For brukeren har denne siden en funksjon. Det er å logge brukeren inn på applikasjonen.

Design

Logg inn siden for QuickFeed har et minimalistisk design. Som vist i figur 5.1 består siden av tre elementer. Første element er navnet på applikasjonen. Neste element er tekstelementet "Sign In". Dette elementet viser hvor brukeren skal logge seg inn i applikasjonen. Siste elementet er "Sign in with GitHub"'- knappen. Denne knappen skal logge brukeren inn til applikasjonen gjennom GitHub og OAuth 2.0.



Figur 5.1: Brukergrensesnitt for å logge inn på applikasjonen

Første gang en bruker logger inn vil det åpnes et vindu i hens standard nettleser. Her blir brukere bedt om å godta at QuickFeed kan få tilgang til brukerens tilgangsbevis (Access token). Om en bruker har logget seg inn på

QuickFeed tidligere vil hen komme rett inn på applikasjonen.

Diskusjon

Dagens implementasjon av QuickFeed-server er ikke kompatibel med autentisering for andre applikasjoner enn web-applikasjonen. Etter en bruker er autentisert på serveren videresendes hen til web-applikasjonen, uavhengig av hvilken applikasjon brukeren benyttet fra start.

Selv om serveren ikke er kompatibel med autentisering i swift-applikasjonen, har vi valgt å utvikle et forslag til hvordan klient-siden kan håndtere autentisering med en fremtidig implementasjon av serveren. Kodelistingen 5.1 viser at vi har benyttet oss av Apples egen *ASWebAuthenticationSession*. ASWebAuthenticationSession er en metode, utviklet av Apple, laget for å autentisere applikasjoner opp mot tredjeparts internett-tjenester. Metoden tar inn URL-adressen til autentiseringstjenesten og *callbackURLScheme*. ASWebAuthenticationSession åpnes i et forgrunnsvindu, og vil inneholde brukerens standard nettleser. Autentiseringen fullføres med at brukeren videresendes til *quickfeed://* etterfulgt av autentiseringsvariablene. Etter brukeren er videre sendt til denne adressen vil applikasjonen prosessere autentiseringsvariablene, forgrunnsvinduet med nettleseren blir avsluttet automatisk og brukeren vil da bli logget inn på QuickFeed.

```
1  guard let authURL = URL(string: "https://QuickFeed.no/auth/
       github/github") else { return }
2  let session = ASWebAuthenticationSession(url: authURL,
       callbackURLScheme: "quickfeed", completionHandler: { (
       callbackURL, error) in
3      guard error == nil, let callbackURL = callbackURL else {
       return }
4
5      let queryItems = URLComponents(string: callbackURL.
       absoluteString)?
6          .queryItems
7
8      let code = queryItems!.first(where: { $0.name == "code" })?.
       value
9
10     DispatchQueue.main.async {
11         self.viewModel.setUser(userID: code)
12     }
13 })
```

```
14
15 session.prefersEphemeralWebBrowserSession = true
16 session.presentationContextProvider = self
17 session.start()
```

Kode 5.1: Forslag til ASWebAuthenticationSession hentet fra GitHubManager.swift

Ulempen med at serveren ikke er kompatibel med autentisering for applikasjon vår gjør at flere elementer på QuickFeed-applikasjonen ikke vil fungere optimalt. Dette er elementer som tar i bruk GitHub tilgangsbeviset til brukeren. Uten autentisering henter ikke serveren oppdatert tilgangsbevis fra GitHub. Elementer som blir rammet av dette er:

- En lærer eller studentassistent kan ikke godkjenne brukere til et emne.
- En lærer eller studentassistent kan ikke endre medlemmers status i et emne.
- En lærer eller studentassistent kan ikke godkjenne nye grupper.
- En administrator kan verken opprette nye emner eller endre eksisterende emner.
- Det vil ikke være mulig å hente oppdatert informasjon fra GitHub
- Det vil ikke være mulig å registrere nye bruker.

Alle elementene i listen over er implementert på swift-applikasjonen, men vil ikke fungere før det oppstår autentisering mellom server og klient.

5.2 Bruker

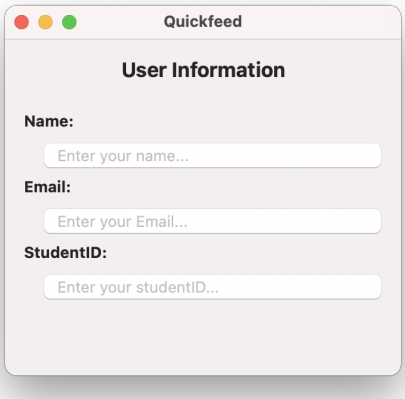
I QuickFeed applikasjonen finnes det to typer brukergrensesnitt som er tilgjengelig for alle brukere. En bruker som besøker applikasjonen for første gang vil få opp et brukergrensesnitt hvor brukeren vil måtte fylle ut litt informasjon om seg selv. Det andre grensesnittet som er tillgjengelig for alle brukere er brukerens profil. Her kan brukeren endre sin brukerinformasjon, samt melde seg opp til nye kurs.

5.2.1 Førstegangsregistrering

Når en bruker logger inn for første gang vil den få opp et brukergrensesnitt for førstegangsregistrering. Her må brukeren oppgi informasjon som vil være nyttig for QuickFeeds administratorer og lærere. Lærere vil bruke denne informasjonen til å finne ut hvilke studenter som har fått godkjent hvilke oppgaver, samt rette og karaktersette oppgaver for studentene.

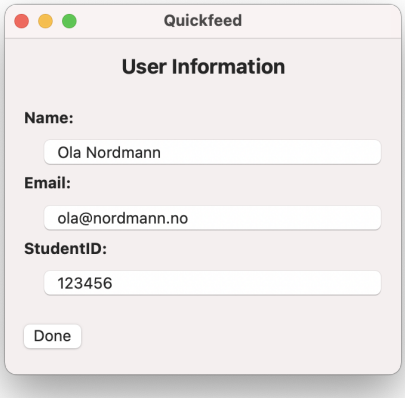
Design

For at registreringen skal gå enklest mulig, er førstegangsregistrering designet med effektivitet og enkelhet i tankene. Som vist i figur 5.2, er grensesnittet for førstegangsregistrering bygd opp av tre enkle tekstfelt med tilhørende overskrifter. Når tekstfeltene står tomme vil det som standard stå en liten beskrivende tekst i hvert felt. Denne teksten beskriver hva feltet skal inneholde.



Figur 5.2: Førstegangsregistrering på QuickFeed

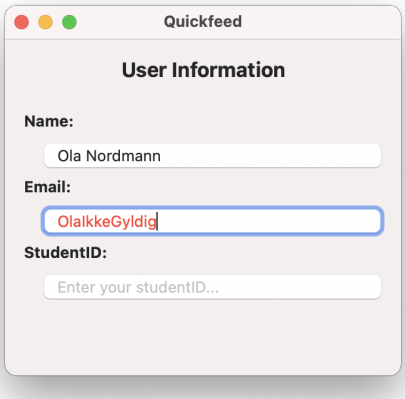
Etter en bruker er ferdig med å fylle ut informasjonsskjemaet vil en knapp bli synlig i bunnen av skjemaet. Denne knappen gir brukeren mulighet til å kunne fullføre registreringen.



Figur 5.3: Ferdig utfylt førstegangsregistrering

Funksjonalitet

For at informasjonen brukerne gir fra seg skal være så pålitelig som mulig, sjekkes alle informasjonsfeltene for gyldig innhold i sanntid. I tilfeller hvor en bruker har oppgitt ugyldig innhold i et felt vil teksten, i det feltet som inneholder feil, farges rødt helt frem til det ugyldige er rettet opp. Det vil ikke være mulig for en bruker å registrere seg med ugyldig informasjon da knappen for å fullføre brukerinformasjonen ikke vil være tilgjengelig før alle feltene inneholder gyldig informasjon.



Figur 5.4: Ugyldig input i førstegangsregistrering

I kodeutdraget 5.2 ser vi at de tre tekstfeltene i figuren over er definert som "@State private" variabler. Med bruk av denne type variabler kan koden sjekkes i sanntid.

```
1 @State private var userName: String = ""
2 @State private var userEmail: String = ""
3 @State private var userStudentID: String = ""
```

Kode 5.2: @State variabler hentet fra NewUser.swift

Når brukeren fyller inn sin informasjon i @State variablene sjekkes feltene konitnuerlig opp i funksjonene vist i kodeutdrag 5.3. Funksjonen "isValid-Name()" sjekker om navnet inneholder andre tegn enn bokstaver. Videre

sjekker "isValidEmail()" om email variabelen oppfyller formen på en gyldig email adresse. Siste funksjon "isValidStudentID()" sjekker om student IDen til brukeren inneholder noe annet enn tall.

```
1  func isValidName() -> Bool {
2      return self.userName.replacingOccurrences(of: " ", with: "")
       .allSatisfy{ $0.isLetter }
3  }
4
5  func isValidEmail() -> Bool {
6      let emailRegEx = "[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z
       ]{2,64}"
7
8      let emailPred = NSPredicate(format:"SELF MATCHES %@",
       emailRegEx)
9      return emailPred.evaluate(with: self.userEmail)
10 }
11
12 func isValidStudentID() -> Bool {
13     return self.userStudentID.allSatisfy{ $0.isNumber } && !self
       .userStudentID.isEmpty
14 }
```

Kode 5.3: Funksjoner brukt til å validere brukerinformasjon hentet fra NewUser.swift

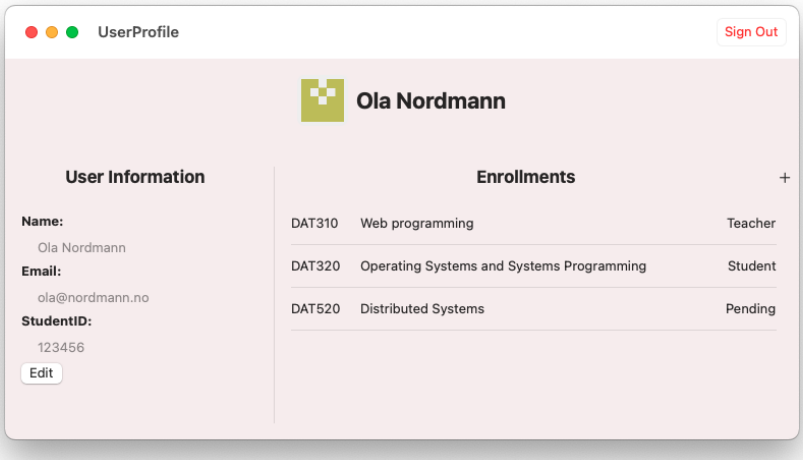
5.2.2 Brukerprofil

Fra brukerprofilsiden vil en bruker kunne administrere sin egen informasjon, administrere registrerte emner, samt logge seg ut av applikasjonen.

Design

Øverst på profilen til brukerne er det plassert en verktøylinje. Til venstre i denne verktøylisten er overskriften på brukergrensesnittet plassert. Til høyre i denne verktøylinjen finner brukeren "Sign Out" knappen. Denne knappen kan brukeren benytte seg av når hen ønsker å logge ut av applikasjonen. Selve grensesnittet til brukerprofilen er bygd opp av en vertikal stabel som starter med GitHub avataren og navnet til brukeren. Under dette kommer det en horisontal stabel som inneholder et område for brukerinformasjon,

og et område for brukerens tilhørende emner. I området med brukerinformasjon kan brukeren både se og endre sin egen informasjon. Området med brukerens tilhørende emner er bygd opp av en liste som inneholder emnekode, emnenavn og emnerollen til brukeren i det emnet. Emnelisten blir sortert ut fra emnekodene til de forskjellige emnene. Ved å trykke på "+"-ikonet til høyre for "Enrollments"-overskriften får brukeren mulighet til å kunne melde seg opp til nye emner på QuickFeed.

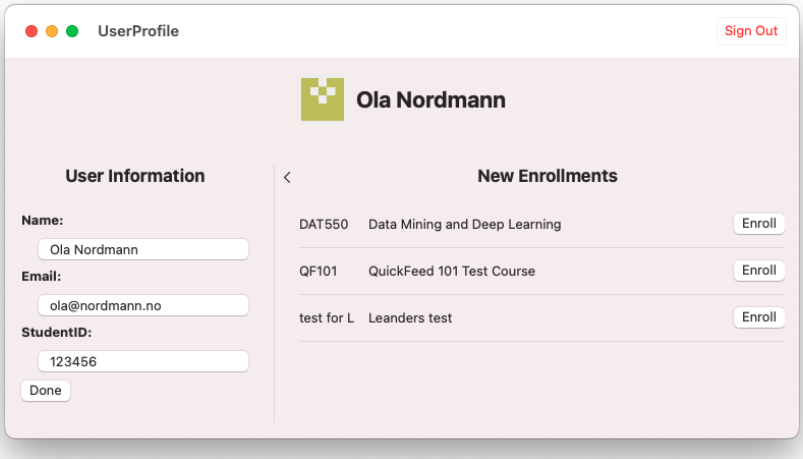


Figur 5.5: Brukerens profil

Funksjonalitet

Om en bruker ønsker å endre brukerinformasjonen som er lagret, blir brukergrensesnittet for brukerinformasjon endret til et brukergrensesnitt identisk med førstegangsregistrering (figur 5.3). Her vil funksjonen til grensesnittet være helt lik funksjonaliteten til førstegangsregistrering (seksjon 5.2.1). Når en bruker velger å melde seg opp til nytt emne vil overskriften på "Enrollments"-brukergrensesnittet endres til "New Enrollments" og listen med emner vil endres til en liste med tilgjengelige emner. Den nye listen vil være sortert på emnekode og vil inneholde emner som brukeren ikke har noe emnerolle i. For å melde seg opp i et emne trykker brukeren på "Enroll"-knappen til det valgte emnet. Brukeren vil da bli oppmeldt og få

emnerollen Pending i dette emnet. Etter en bruker har blitt tatt opp til et emne, enten som student eller som lærer, vil det være mulig å trykke på dette emnet i emnelisten. Det emnet som har blitt trykket på vil da åpnes i navigasjonsfeltet til applikasjonen (seksjon 5.3).



Figur 5.6: Brukerens profil med redigering av brukerinformasjon og kurs oppmelding

For å sortere emnene etter emnekode benytter vi oss av funksjonen vist i kodelisten 5.4. Her henter funksjonen en listen med emner fra visningsmodellen, før den sorterer listen etter emnekode. Til slutt returneres den sorterte listen.

```
1 func sortCourseByCode() -> [Course] {
2     var courses = viewModel.getAllCourses()!
3     courses.sort { $0.code < $1.code }
4     return courses
5 }
```

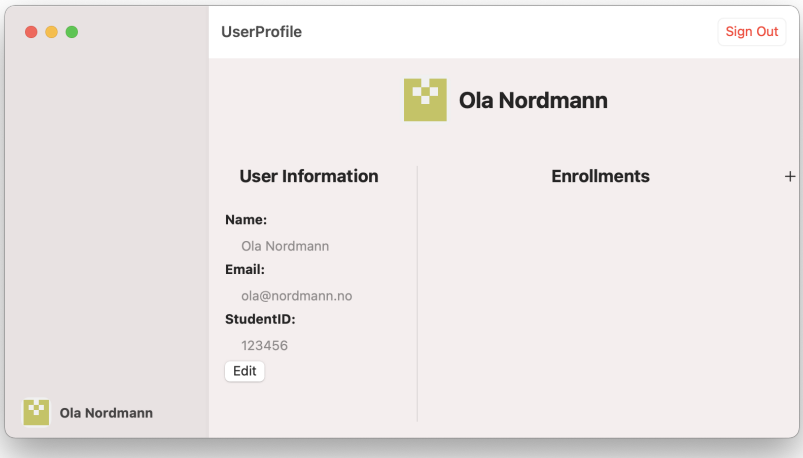
Kode 5.4: Metode for å sortere etter emnekode hentet fra UserEnrollments.swift

5.3 Navigasjonsfelt

Et av våre hovedmål var at det skulle ta minst mulig tid å navigere seg gjennom applikasjonen. Dette har vi løst ved å utvikle et navigasjonsfelt som til en hver tid vil være tilgjengelig for brukeren. Dette navigasjonsfeltet vil variere i innhold ut i fra hvilken brukerrolle (seksjon 3.2.1) brukeren har i applikasjonen og hvilken emnerolle (seksjon 3.2.2) brukeren har i det valgte emnet. Uavhengig av brukerrolle og emnerolle vil navigasjonsfeltet alltid være tilgjengelig til venstre i applikasjonen.

5.3.1 Ordinær bruker

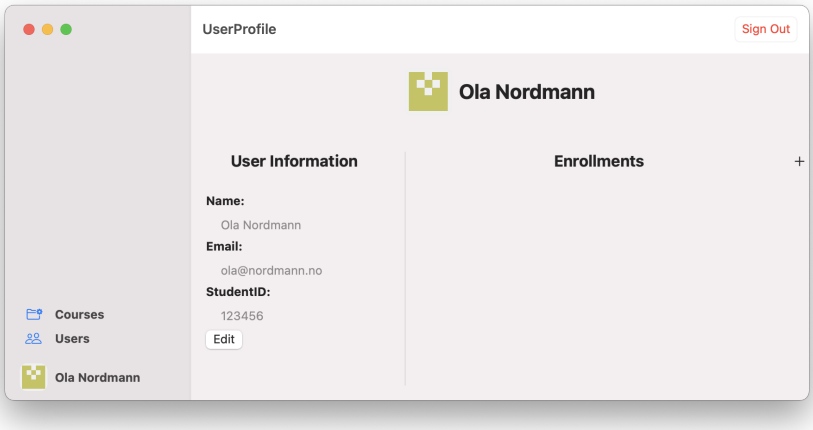
Som en ordinær bruker av QuickFeed, uten emnerolle i noen av QuickFeeds emner, vil navigasjonsfeltet være relativt tomt. Som vist i figur 5.7 vil navigasjonsfeltet helt i bunnen inneholde en knapp som gir brukeren tilgang til å navigere til sin egen brukerprofil. Denne knappen inneholder brukerens GitHub avatar, etterfulgt av brukerens navn. Navigasjonsknappen for brukerprofil vil til enhver tid være tilgjengelig for brukeren uavhengig av hvor i applikasjonen brukeren befinner seg.



Figur 5.7: Felles navigasjonsfelt for brukere

5.3.2 Administrator

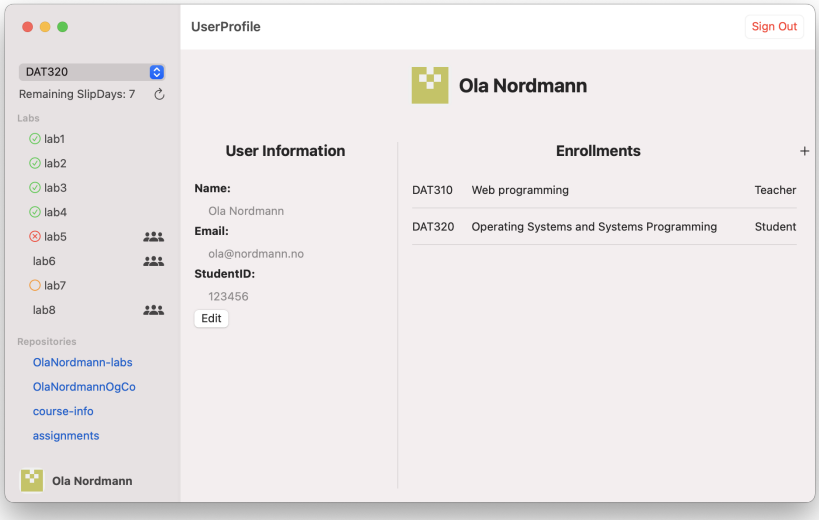
En bruker som har oppnådd brukerrollen (seksjon 3.2.1) Administrator vil få mulighet til å navigere seg frem til ulike administrator sider på applikasjonen. Dette løste vi, som vist i figur 5.8, med å legge til to ekstra navigasjonsknapper over brukerprofilknappen i navigasjonsfeltet. En av navigasjonsknappene er merket "Course", for å kunne administrere ulike emner på QuickFeed, og en er merket "Users", for å kunne administrere ulike brukere på QuickFeed. Disse navigasjonsknappene vil navigere brukeren, avhengig av hvilken navigasjonsknapp som blir benyttet, til en applikasjonsside for å administrere brukere eller administrere emnene på QuickFeed. På lik linje med brukerprofilknappen, som forklart i seksjon 5.3.1 og figur 5.7, vil administratorknappene også være tilgjengelig for brukeren til enhver tid uavhengig av hvor i applikasjonen brukeren befinner seg.



Figur 5.8: Navigasjonsfelt for administratorer

5.3.3 Student

Etter en bruker har blitt tatt opp i et emne med emnerollen (seksjon 3.2.2) Student vil navigasjonsfeltet endres til en navigasjonsfelt som er optimaliserte for studenter i det valgte emnet. Som vist i figur 5.9, vil det øverst i navigasjonsfeltet være en flervalgsmeny hvor brukeren kan velge hvilket emne hen vil se. Under denne emnevelgeren vil brukeren kunne se hvor mange slipDays hen har igjen i det valgte emnet. Her er det også plassert en oppdateringsknapp som brukeren kan bruke for å oppdatere innleveringsresultatene. Etter slipDays får brukeren opp en liste med alle innleveringene i emnet. Hver innlevering er en egen navigasjonsknapp som åpner innleveringssiden til den valgte innleveringen på høyre side av applikasjonen. Etter listen med innleveringer får brukeren opp en ny liste med nyttige HTTP lenker. Disse fører brukeren til nyttige GitHub-sider som er relatert til det valgte emnet.



Figur 5.9: Navigasjonsfelt for Student

Som sett i figuren over, har noen av de forskjellige innleveringene et lite symbol plassert foran navnet. Dette symbolet beskriver hvilken status innleveringen har. Med denne statusen plassert foran innleveringen, trenger

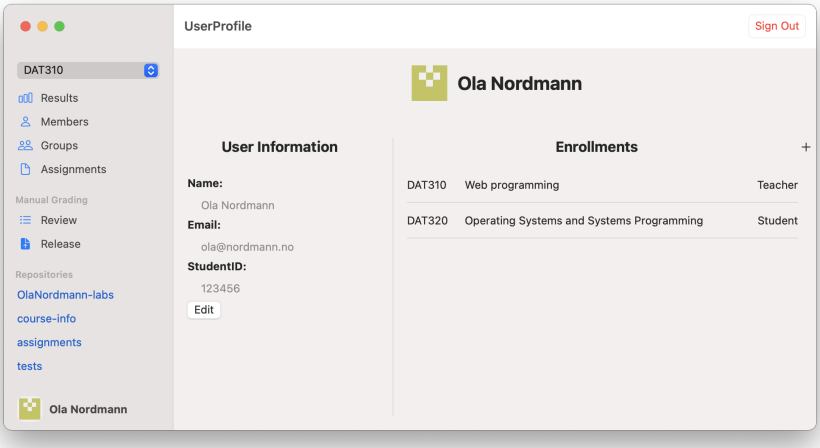
ikke brukeren trykke seg inn på innleveringen for å se statusen. De forskjellige statusene betyr:

- **Grønn checkbox** - Godkjent innlevering
- **Oransje ring** - Prøvd å få godkjent, et nytt forsøk
- **Rødt kryss** - Ikke godkjent innlevering
- **Blå ring** - Påbegynt innlevering
- **Ingen tegn** - Ikke startet arbeid med innleveringen

Innleveringene i noen emner kan være gruppeinnleveringer. Dette har vi valgt å indikere med et tegn som viser flere personer. Dette tegnet er plassert etter navnet på innleveringene i innleveringslisten (se figur 5.9 lab5, lab6 og lab8). Om et emne inneholder gruppeinnlevering, og brukeren ikke har noen gruppe, vil brukeren få opp en ekstra navigasjonsknapp mellom innleveringseksjonen og GitHub lenkene. Denne navigasjonsknappen er merket "New Group", og vil gi brukeren mulighet til å lage seg en gruppe, dette vil bli forklart nærmere i seksjon 5.4.3.

5.3.4 Teacher

Etter en bruker har blitt tatt opp i et emne med emnerollen 3.2.2 Teacher vil navigasjonsfeltet endres til en navigasjonsfelt som er optimaliserte for studentassistenter og lærere i det valgte emnet. Som vist i figur 5.10 vil navigasjonsfeltet endres til en liste med navigasjonsknapper til nyttige sider for studentassistenter og lærere for det gitte kurset. Funksjonen til de ulike sidene vil bli presentert i seksjon 5.5. Under listen med navigasjonknapper får brukeren opp en ny liste med nyttige HTTP lenker. Disse fører brukeren til nyttige GitHub sider som er relatert til det valgte emnet.



Figur 5.10: Navigasjonsfelt for brukere med emnerolle Teacher

Som vist i figuren over, kan innleveringer rettes manuelt. Om et emne ikke tilbyr manuelt rettet innleveringer, vil ikke denne seksjonen være med i navigasjonsfeltet til dette emnet.

Funksjonalitet for navigasjonsfeltene

Når en bruker bytter emne i emnevelgeren, eller via brukerprofilen, vil navigasjonsfeltet automatisk endres til riktig navigasjonsfelt for det valgte emnet. For at det skulle bli mulig å endre fra to forskjellige plasser er @State variabelen i navigasjonsfeltet. Denne variabelen sendes som en @Binding variabel ut til brukerprofilen og til emnevelgeren.

Fordelen med et slikt dynamisk navigasjonsfelt vil være å kunne tilpasse hvilke navigasjonsknapper som kan benyttes for hvert emne. Det vil aldri være mulig å navigere til en unødvendig side for det valgte emnet.

Som forklart i hvert delkapittel, i seksjon 5.3, er alle knappene i navigasjonsfeltet navigasjonsknapper, bortsett fra GitHub lenkene. Alle navigasjonsknappene vil åpne den valgte applikasjonssiden på høyre side av applikasjonen. Dette gjør at navigasjonsfeltet alltid vil være tilgjengelig i venstre side av applikasjonen, med mindre brukeren selv har valgt å fjerne navigasjonsfeltet.

5.4 Student

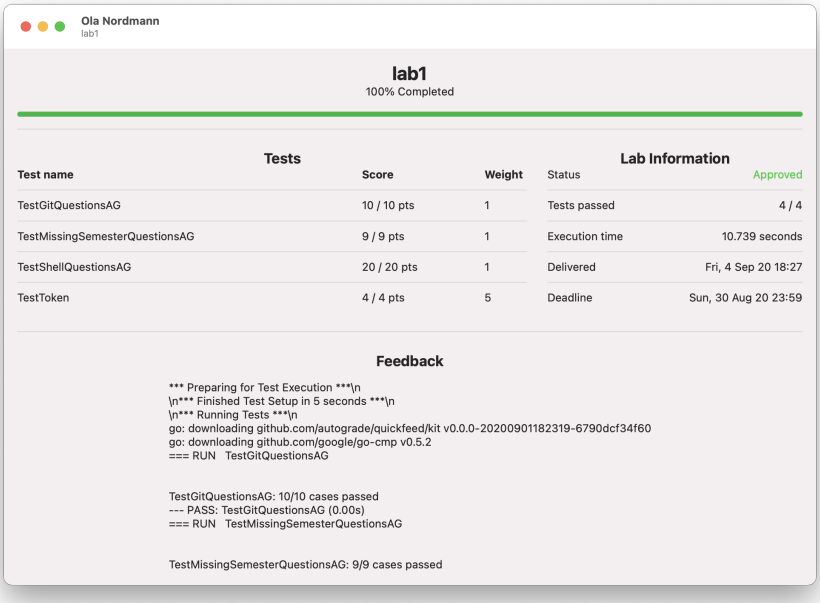
Som beskrevet i seksjon 5.3 får en bruker med emnerolle (seksjon 3.2.2) Student mulighet til trykke seg inn på de forskjellige innleveringene i det valgte faget. Etter en bruker har valgt innlevering vil hen få opp en av to tilbakemeldingsider. Hvilken side brukeren får opp avhenger av hvilken metode innleveringen rettes etter, om den er manuelt eller automatisk rettet.

5.4.1 Automatisk rettet oppgave

Ved automatisk rettet innleveringer blir innleveringene testet i det brukeren laster opp sin programvarekode på GitHub (forklart i seksjon 2.3). På siden for automatisk rettet innleveringer kan brukeren få informasjon om hvor mange prosent riktig en innlevering er, hvilke tester som godkjennes og hvilke som feiler. I tilegg blir det gitt kort og relevant informasjon om innleveringen og en kodegenerert tilbakemelding.

Design

Som vist i figur 5.11, starter siden for automatisk rettet innlevering med innleveringens navn, brukerens oppnådde poengsum i prosent og en fremdriftslinje som indikerer innleveringens status samt hvor mye som gjenstår av innleveringen for å oppnå 100%. Under denne informasjonen følger en horisontal stabel som starter med å vise en liste over hvilke tester som godkjennes og hvilke tester som mislykkes. Videre i den horisontale stabelen vises relevant informasjon om innleveringen som status på innleveringen, hvor mange tester som godkjennes, hvor lang tid det tok å kjøre koden, tid innleveringen ble levert og innleveringsfrist. Etter den horisontale stabelen følger den kodegenererte tilbakemeldingen. Her kan brukeren lese gjennom tilbakemeldingen for å finne ut hva som er rett og galt med innleveringen.



Figur 5.11: Automatisk rettet innlevering for student

Funksjonalitet

Som beskrevet over, vil fremdriftslinjen være med på å beskrive status på innleveringen og vise hvor mange prosent riktig innleveringen er. Både fremdriftslinjen og status i labinformasjonslisten endrer farge og innhold ut i fra hvilken status innleveringen har. De forskjellige fargene og status kodene betyr:

- **Grønn, Approved** - Godkjent innlevering
- **Oransje, Revision** - Prøvd å få godkjent, et nytt forsøk
- **Rødt, Rejected** - Ikke godkjent innlevering
- **Blå, None** - Påbegynt innlevering

Når en innlevering har status som "None" eller "Revision", vil det bli plassert et merke på fremdriftslinjen. Dette merket indikerer hvor mye av innleveringen som må godkjennes for å få godkjent innleveringen.

Fra serveren blir all informasjon om de automatisk rettede innleveringene sendt i JSON-format over gRPC. Før denne informasjonen kunne bli tatt i bruk måtte vi oversette fra JSON til Swift strukturer. Swift strukturene, som vist i kodeutdrag 5.5, holder samme form som JSON objektet.

```
struct SessionBuildInfo: Codable {
    var builddate: String
    var buildid: Int
    var buildlog: String
    var execTime: Int
}

struct ScoreObj: Codable, Hashable {
    var TestName: String
    var Score: Int
    var MaxScore: Int
    var Weight: Int
}
```

Kode 5.5: Strukturer for å oversette JSON objekt hentet fra JSON.swift

For å enkelt kunne få tak i elementene vi trengte for automatisk rettet innleveringer valgte vi å utvide proto definisjonen (seksjon 3.3) av Submission. Som vist i kodeutdrag 5.6 blir Submission utvidet med variablene "buildInfoJSON" og "scoreObj". Innholdet i JSON objektet blir oversatt til Swift strukturer før de returneres til de utvidede variablene.

```
extension Submission{
    var buildInfoJSON: SessionBuildInfo {
        let jsonData = self.buildInfo.data(using: .utf8)!
        return try! JSONDecoder().decode(SessionBuildInfo.self,
    from: jsonData)
    }

    var scoreObj: [ScoreObj]? {
        let jsonData = self.scoreObjects.data(using: .utf8)!
        let scoreObject: [ScoreObj]? = try? JSONDecoder().decode
    ([ScoreObj].self, from: jsonData)

        return scoreObject
    }
}
```

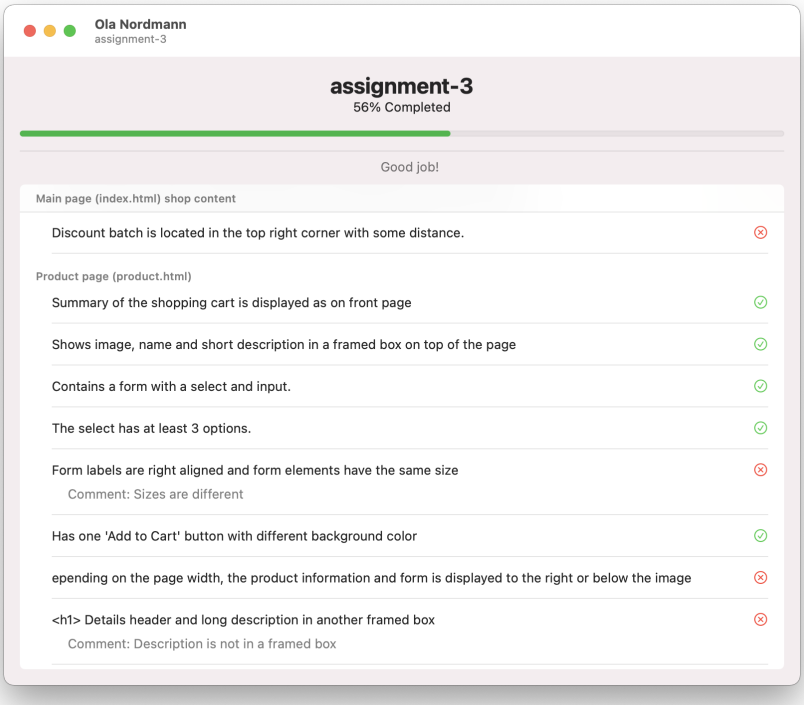
Kode 5.6: Oversette fra JSON til Swift struktur hentet fra ProtoExtensions.swift

5.4.2 Manuelt rettet oppgave

Når en innlevering er manuelt rettet, blir den rettet av en med emnerollen (seksjon 3.2.2) Teacher. Det vil si at en student ikke vil få opp tilbakemelding på innleveringen før en lærer eller studentassistent har rettet innleveringen. Tilbakemeldingsiden vil på manuelt rettet innleveringer se noe anderledes ut enn på den automatisk rettet innleveringen. På de manuelt rettet innleveringene får brukeren informasjon om hvilke status de ulike kriteriene på innleveringen har fått. Brukeren får også opp en fremdriftslinje lik den i automatisk rettet innleveringer.

Design

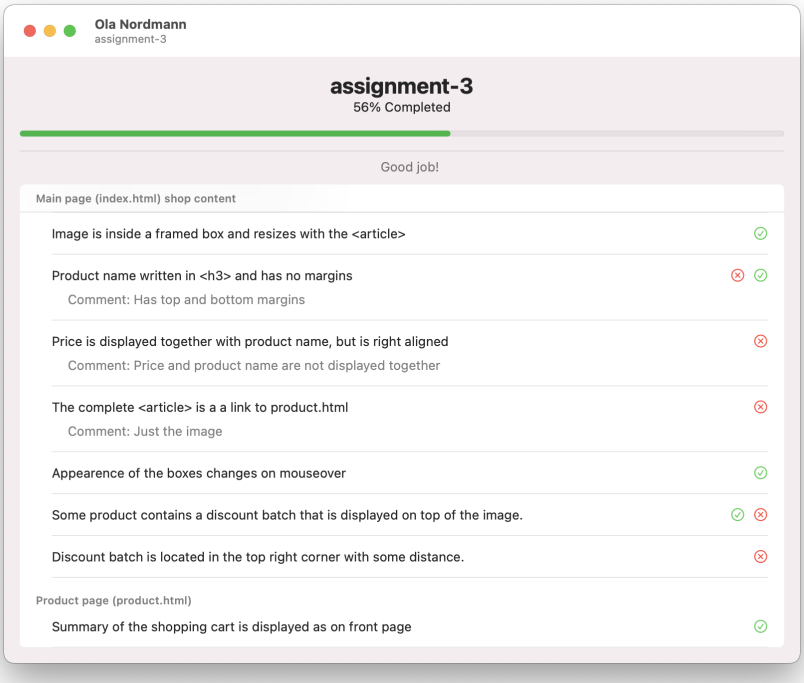
Designet på manuelt rettet innleveringer starter på lik måte som automatisk rettet innleveringer. Som visst i figur 5.12, vil brukeren øverst på siden kunne se navnet på innleveringen sammen med en poengsum vist i prosent og en fremdriftslinje som indikerer innleveringens status, og hvor mye som gjenstår av innleveringen for å oppnå 100%. Fremdriftslinjen på manuelt rettet innlevering har samme funksjoner med farge og merke for innleveringene som forklart i automatisk rettet innleveringer (seksjon 5.4.1). Under dette vil brukeren kunne se tilbakemelding for hele innleveringen. Etter tilbakemeldingen kommer en liste med de ulike kriteriene for innleveringen. I denne listen vil studenten kunne se hva kriteriene er og om brukeren som har rettet innleveringen har lagt til en kommentar til kriteriet. Hvert kriterium har også en godkjent/ikke godkjent poengsum, denne er plassert til høyre for kriteriet.



Figur 5.12: Manuelt rettet innlevering for student

Funksjonalitet

I enkelte tilfeller kan det hende at flere brukere har rettet samme innlevering. Om dette er tilfellet vil applikasjonen først sjekke om de ulike brukerne har gitt forskjellig poengsum til samme kriterium. Om brukerne har gitt samme poengsum, vil det bare vises et poengsumtegn på høyre side av kriteriet, men om brukerne har gitt forskjellig poengsum på samme kriterium, vil begge poengsummene vises (kriterium nr. 2 og 6, figur: 5.13). Om flere av brukerne som rettet innleveringen har lagt til kommentar på samme kriterie, vil alle kommentarene vises for studenten.



Figur 5.13: Manuelt rettet innlevering med flere rettelser for student

Som vist i kodeutdrag 5.7 og kodeutdrag 5.8 er det to funksjoner som brukes for å finne riktige status og riktige kommentarer for et kriterium. For å finne frem til riktige kommentarer brukes funksjonen "getCriteriaComments". Denne funksjonen tar inn listen med alle rettelsene sammen benchmark ID og kriterium ID som argumenter. Ut i fra benchmark ID og kriterium ID finner funksjonen alle kommentarene for samme kriterium og returnere en liste med kommentarer.

```
1 func getCriteriaComments(reviews: [Review], benchmarkIndex: Int,
      criteriaIndex: Int) -> [String]{
2     var criteriaComments: [String] = []
3     for element in reviews{
4         for (index, benchmark) in element.benchmarks.enumerated
      (){
5             if index == benchmarkIndex && benchmark.criteria[
      criteriaIndex].comment != ""{
6                 criteriaComments.append(benchmark.criteria[
```

```
    criteriaIndex].comment)
7               }
8           }
9       }
10      return criteriaComments
11  }
```

Kode 5.7: Finner hvilke kommentarer som skal vises hentet fra HelpFunctions.swift

For å finne frem til riktige status brukes funksjonen "getCriteriaGrade". Denne funksjonen tar inn listen med alle rettelsene sammen benchmark ID og kriterium ID som argumenter. Ut i fra benchmark ID og kriterium ID finner funksjonen alle statusene for samme kriterium og returnere en liste med statuser.

```
1 func getCriteriaGrade(reviews: [Review], benchmarkIndex: Int,
    criteriaIndex: Int) -> [GradingCriterion.Grade]{
2     var grades: [GradingCriterion.Grade] = []
3     for element in reviews{
4         for (index, benchmark) in element.benchmarks.enumerated
    (){
5             if index == benchmarkIndex && !grades.contains(
    benchmark.criteria[criteriaIndex].grade){
6                 grades.append(benchmark.criteria[criteriaIndex].
    grade)
7             }
8         }
9     }
10    return grades
11 }
```

Kode 5.8: Finner hvilke status som skal vises hentet fra HelpFunctions.swift

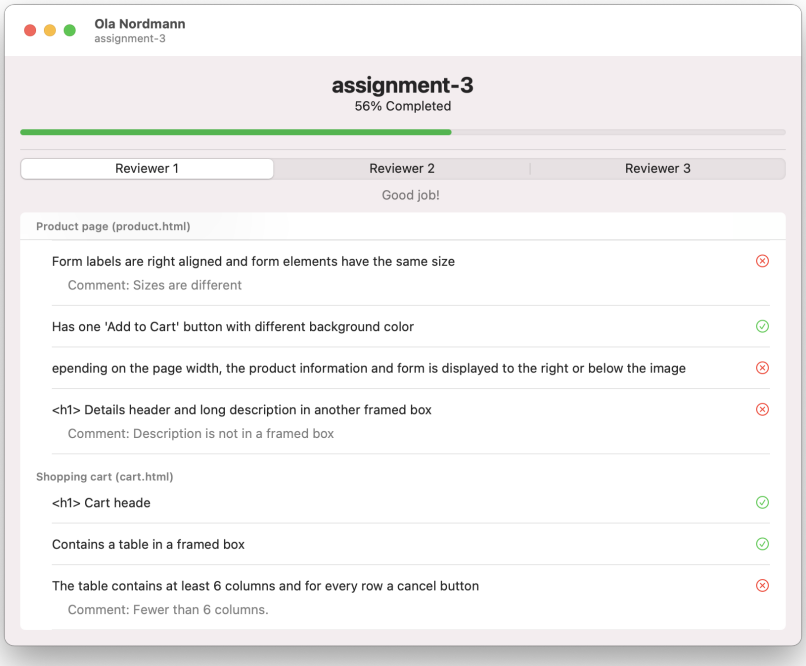
Diskusjon

Under utviklingen av siden for manuelt rettet innleveringer for studenter møtte vi på en utfordring. Vi hadde en klar tanke om hvordan denne siden måtte se ut, men utfordringen viste seg når flere med emnerollen (seksjon 3.2.2) Teacher hadde rettet samme innlevering. Dette kunne løses på tre forskjellige måter:

1. Skulle siden bare vise en av rettingene?
2. Skulle studenten få opp alle rettingene i hver sin liste?
3. Skulle vi designe en kombinasjon av disse to? En som viste en innlevering i liste, men med flere poengsum hvor de forskjellige rettingene hadde ulike poengsum.

Første alternativ hadde vi allerede utviklet med visning når en innlevering bare hadde en retting. Designet på denne løsningen så helt identisk ut med designet forklart i delkapittelet "design" og figur 5.12. Med denne løsningen kunne det skje at den mest relevante rettingen ikke ville blitt presentert for studenten.

Videre utviklet vi alternativ to. Som vist i figur 5.14, startet denne siden med å vise innleveringens tittel sammen med poengsum og fremdriftslinje. Under dette fikk brukeren valget mellom hvilken retting hen ville se. Listen under viste den valgte rettingen.



Figur 5.14: Alternativ til manuelt rettet innlevering med flere rettelser

Vi fant heller ikke denne løsningen som optimal da de fleste rettelsene stort sett ville være like. Og brukeren trenger i de fleste tilfeller ikke flere tilbakemeldinger.

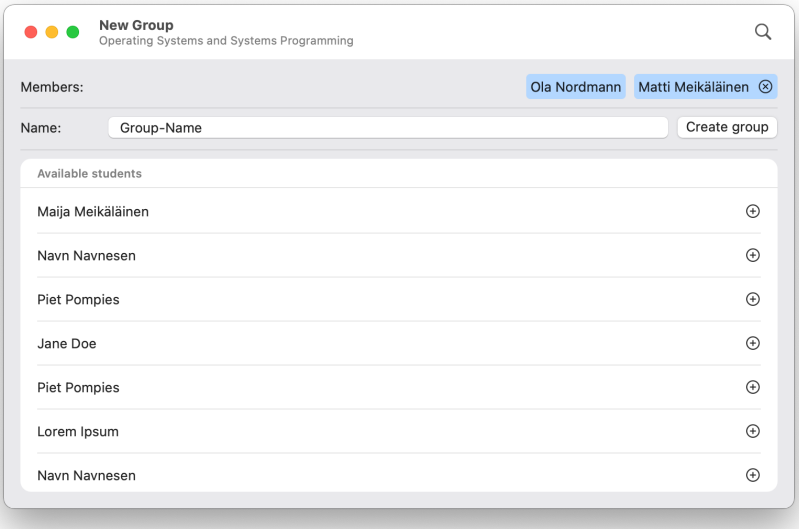
Til slutt utviklet vi alternativ tre. Dette ble den endelige løsningen. Hvordan denne ble er forklart nærmere i delkapittel "Funksjonalitet" og kan sees i figur 5.13. Denne løsningen åpner for at det vil se ut som det er en rettelse på hver innlevering helt frem til de forskjellige rettelsene er uenige, da ville begge rettelsene vises for studenten. Her vil studenten få all den informasjon som er relevant, uten å måtte se gjennom flere relativt like rettelser av samme innlevering.

5.4.3 Grupper

På siden "New Group" kan brukere med emnerollen Student opprette gruppe med andre studenter i emnet. Brukeren kan velge å opprette gruppe med studenter som ikke er medlem av en gruppe i det emnet.

Design

Som vist i figur 5.15 er det plassert en verktøylinje øverst på gruppesiden. Denne verktøylinjen inneholder sidens overskrift "New Group" etterfulgt av underoverskriften som inneholder emnets navn. Til høyre i verktøylinjen befinner det seg et søkefelt. Dette søkefeltet kan studenten benytte seg av for å søke etter andre studenter som ikke har gruppe. Hovedsiden for å lage ny gruppe inneholder tre elementer. Øverste element er et felt som beskriver gruppens medlemmer. Her vil gruppemedlemmene vises samtidig som de blir valgt inn i gruppen. Når en student lager ny gruppe vil hen være med i gruppen. Derfor vil hens navn være i denne listen fra start. Det vil også være mulig å fjerne medlemmer fra gruppen her, men brukeren kan ikke fjerne seg selv. Under gruppens medlemmer kommer det et tekstfelt. Dette tekstfeltet skal inneholde gruppens navn. Bak tekstfeltet er det en "Create Group" knapp. Denne knappen vil være deaktivert frem til gruppen har et navn. Under gruppens navn finner brukeren en liste med emnets medlemmer. Dette er studenter som ikke har gruppe fra før. Når brukeren velger å legge til en bruker i gruppen forsvinner brukerene fra listen og vises da i feltet med gruppens medlemmer.



Figur 5.15: Ny gruppe for student

Etter en bruker har opprettet gruppen får gruppestatus som Pending. Gruppen venter da på at faglærer eller studentassistenter skal godkjenne opprettelsen av gruppen.

5.5 Lærer

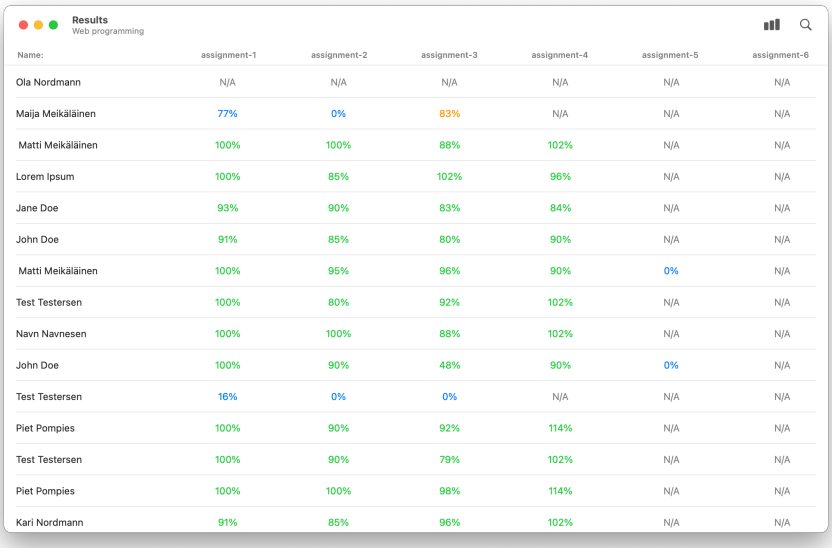
Som vist i figur 5.10 får en bruker med emnerollen Teacher tilgang til flere nye sider av applikasjonen. Lærere og studentassistenter i de forskjellige emnene trenger å ha en oversikt over hvilke brukere som har meldt seg opp til emnet. Brukere med Teacher rolle vil også ha muligheten til å dele emnets medlemmer opp i grupper for gruppeinnleveringer. De trenger også muligheten til å kunne se resultater de ulike studentene har oppnådd i emnet. Videre trenger de en oversikt over emnets forskjellige innleveringer. I noen tilfeller må innleveringene rettes manuelt. Etter en innlevering har blitt manuelt rettet må denne sjekkes og godkjennes før den leveres ut til studenten.

5.5.1 Resultater

På resultatsiden kan brukeren skaffe seg en oversikt over hvordan de forskjellige studentene mestrer faget. Her vil brukeren få opp samtlige studenters resultater i emnet. På denne siden befinner det seg også en statistikk som viser hvor mange personer som har fått godkjent de ulike innleveringene.

Design

Øverst på resultatsiden er det plassert en verktøylinje. Som vist i figur 5.16 inneholder denne verktøylinjen sidens overskrift "Results" sammen med en underskrift som inneholder emnets navn. Til venstre i verktøylinjen finnes det et statistikk-ikon. Dette ikonet fører brukeren til en statistikk side. Denne siden inneholder statistikk over emnets innleveringer. Etter statistikk-ikonet finner vi et forstørrelsesglass. Her kan brukeren søke etter brukere på resultatsiden. Selve hovedsiden for resultater er bygd opp av en liste, hvor hver rad representerer en ny bruker. Antall kolonner i listen vil variere. Første kolonne inneholder studentenes navn. Det vil bli laget en ny kolonne for hver innlevering det er i emnet. Hver kolonne beskrive hvilken poengsum i prosent studenten har oppnådd på innleveringen. Om en student ikke har startet med arbeidet på en innlevering vil kolonnen vise "N/A".



Name:	assignment-1	assignment-2	assignment-3	assignment-4	assignment-5	assignment-6
Ola Nordmann	N/A	N/A	N/A	N/A	N/A	N/A
Maija Meikäläinen	77%	0%	83%	N/A	N/A	N/A
Matti Meikäläinen	100%	100%	88%	102%	N/A	N/A
Lorem Ipsum	100%	85%	102%	96%	N/A	N/A
Jane Doe	93%	90%	83%	84%	N/A	N/A
John Doe	91%	85%	80%	90%	N/A	N/A
Matti Meikäläinen	100%	95%	96%	90%	0%	N/A
Test Testersen	100%	80%	92%	102%	N/A	N/A
Navn Navnesen	100%	100%	88%	102%	N/A	N/A
John Doe	100%	90%	48%	90%	0%	N/A
Test Testersen	16%	0%	0%	N/A	N/A	N/A
Piet Pompies	100%	90%	92%	114%	N/A	N/A
Test Testersen	100%	90%	79%	102%	N/A	N/A
Piet Pompies	100%	100%	98%	114%	N/A	N/A
Kari Nordmann	91%	85%	96%	102%	N/A	N/A

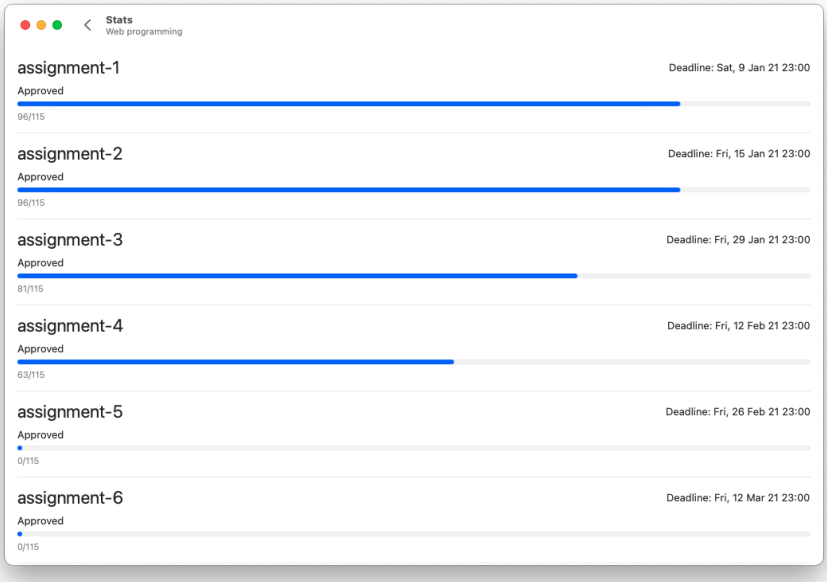
Figur 5.16: Resultatliste for et emne

Som sett i figuren over har poengsummene blitt farget i forskjellige farger. De forskjellige fargene beskriver hvilken status den gitte innleveringen har. Fargene betyr følgende:

- **Grønn** - Godkjent innlevering
- **Oransje** - Prøvd å få godkjent, et nytt forsøk
- **Rødt** - Ikke godkjent innlevering
- **Blå** - Påbegynt innlevering

Ved å trykke på statistikk-ikonet får brukeren opp en statistikk side. Som vist i figur 5.17 endres overskriften i verktøylinjen til “Stats“. Foran overskriften blir det plassert en pil som gir brukeren mulighet til å returnere til resultatlisten. Hovedpoenget med denne siden er at brukeren fort skal få en oversikt over hvor mange av medlemmene som har fått godkjent de ulike innleveringene. Hovedsiden inneholder en liste med alle innleveringene

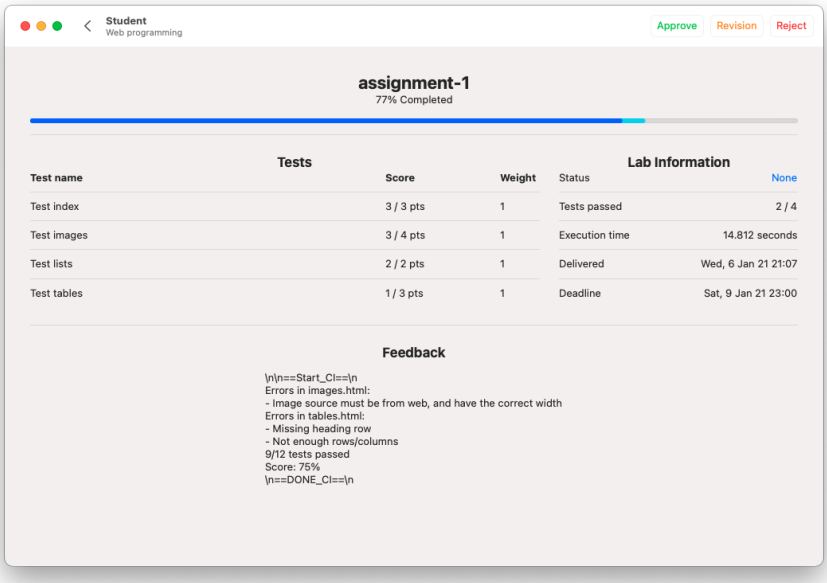
for emnet. Hvert liste element inneholder innleveringens navn sammen med innleveringens frist. Videre i liste elementet finner brukeren en fremdriftslinje som indikere hvor mange medlemmer av totalt alle medlemmene som har fått godkjent den gitte innleveringen. Under fremdriftslinjen kan hen finne hvor mange medlemmer i antall som har fått godkjent innleveringen.



Figur 5.17: Statistikk for et emne

Funksjonalitet

Tidligere i figur 5.16 så vi at hver student hadde flere kolonner som inneholdt poengsum på de forskjellige innleveringene. En bruker med emnerolle Teacher kan trykke på disse poengsummene, og da få opp tilbakemeldingsiden til den valgte innleveringen. Som vist i figur 5.18 vil hovedsiden være helt identisk med en students tilbakemeldingside (figur 5.11 og figur 5.12). Eneste som fraviker fra studentens tilbakemeldingside er verktøylinjen. I verktøylinjen til en lærer og studentassistent vil det være mulig å manuelt endre status på innleveringen.



Figur 5.18: Teacher visning av en students innlevering

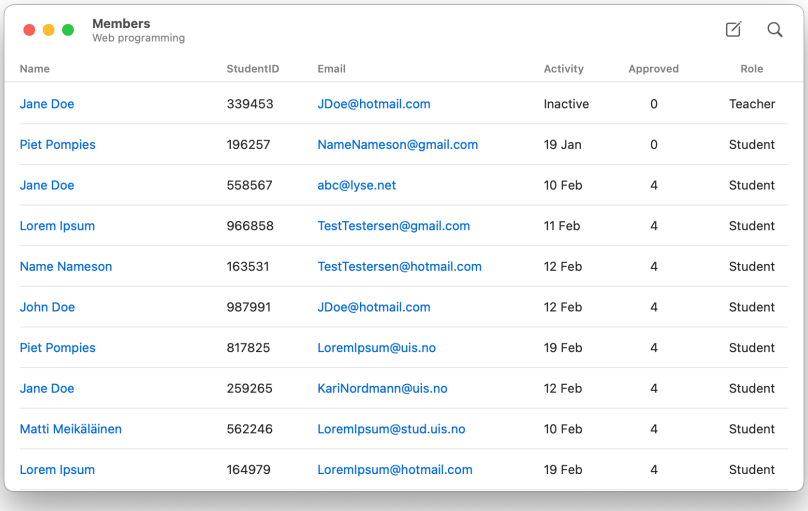
5.5.2 Medlemmer

For en lærer og studentassistent er det viktig med god oversikt over emnets brukere. Fra medlemsiden vil brukerene med emnerolle Teacher få mulighet til å se hvem som er medlemmer i emnet, samt tidspunktet disse medlemmene sist var aktive på applikasjonen. Brukerene får videre tilgang til å kunne godta andre brukere til å ta del i emnet.

Design

Som vist i figur 5.19 er medlemsiden bygd opp med en verktøylinje øverst. Denne verktøylinjen inneholder sidens overskrift "Members" etterfulgt av underoverskriften som inneholder emnets navn. Til høyre i verktøylinjen er det plassert to ikoner. Det ene ikonet gir brukeren mulighet til å endre emnerolle til andre brukere. Det andre ikonet gir brukeren tilgang til et

søkefelt hvor hen kan søke etter brukere. Selve medlemsiden er bygd opp av en liste av alle medlemene. Hver rad i listen representerer ett medlem. Hver rad har fem kolonner. Første kolonne inneholder brukerens navn, andre kolonne er medlemets student ID, tredje er medlemmets mail-adresse, fjerde beskriver tid medlemmet sist var aktiv på QuickFeed og femte kolonne viser medlemmets emnerolle.



Name	StudentID	Email	Activity	Approved	Role
Jane Doe	339453	JDoe@hotmail.com	Inactive	0	Teacher
Piet Pompies	196257	NameNameson@gmail.com	19 Jan	0	Student
Jane Doe	558567	abc@lyse.net	10 Feb	4	Student
Lorem Ipsum	966858	TestTestersen@gmail.com	11 Feb	4	Student
Name Nameson	163531	TestTestersen@hotmail.com	12 Feb	4	Student
John Doe	987991	JDoe@hotmail.com	12 Feb	4	Student
Piet Pompies	817825	LoremIpsum@uis.no	19 Feb	4	Student
Jane Doe	259265	KariNordmann@uis.no	12 Feb	4	Student
Matti Meikäläinen	562246	LoremIpsum@stud.uis.no	10 Feb	4	Student
Lorem Ipsum	164979	LoremIpsum@hotmail.com	19 Feb	4	Student

Figur 5.19: Medlemsliste for emner

Funksjonalitet

I listen over medlemmer er både navnet til medlemmene og mail-adressen markert i blått. Begge disse er bygd opp som lenker. Navnet sender brukeren til medlemmets GitHub profil, mens mail-adressen sender brukeren til dens standard mail applikasjon. I mail applikasjonen vil det automatisk åpnes en tom mail, hvor brukerens mail vil være avsender og brukeren hen trykket på vil være mottaker.

Ved å trykke på endre ikonet i verktøylinjen får brukeren tilgang til å endre medlemmers emnerolle. Når en bruker går inn for å bytte enmerolle på emnets medlemmer får hen to valg. Hen kan velge å endre emnerolle fra

Student til Teacher og omvendt. Hen kan også velge å melde noen av emnet.

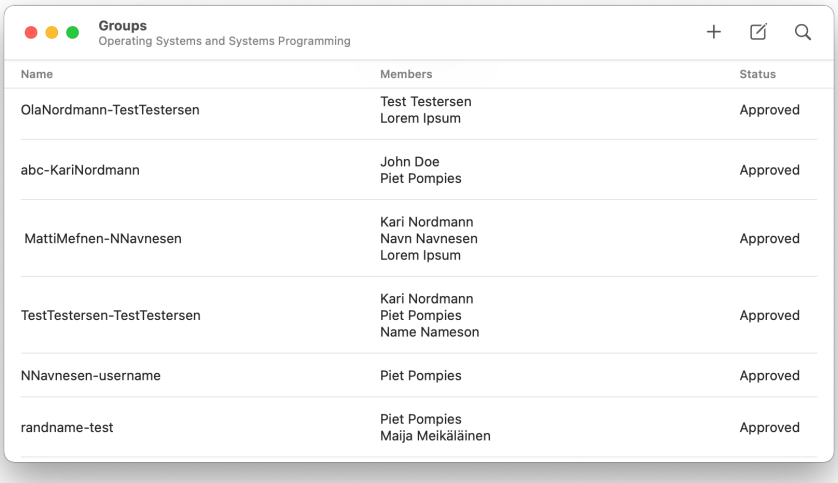
Når brukere på QuickFeed melder seg opp til et emne vil det være i denne listen de vil vises. Etter brukeren har meldt seg opp får den emnerolle som Pending. En lærer eller studentassistent kan da velge å godta eller avvise brukerens oppmelding til emnet.

5.5.3 Grupper

På siden "Groups" kan brukere med emnerollen Teacher skaffe seg en oversikt over hvilke grupper som finnes i det valgte emnet. Her kan brukeren se navnet til gruppene, gruppenes medlemmer og hvilken status gruppene har. Fra denne siden vil det også være mulig for lærere og studentassistenter å opprette nye grupper for studentene.

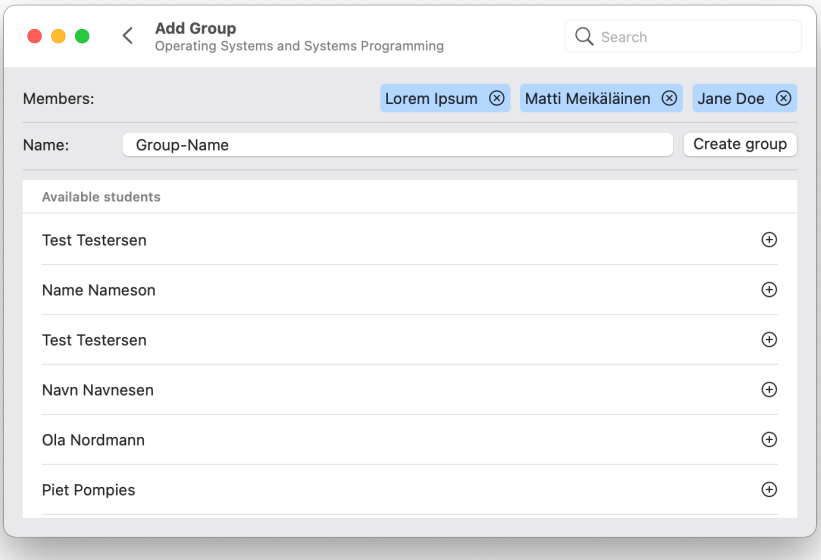
Design

Som vist i figur 5.20 er det plassert en verktøylinje øverst på gruppesiden. Denne verktøylinjen inneholder sidens overskrift "Groups" etterfulgt av underoverskriften som inneholder emnets navn. Til høyre i verktøylinjen befinner det seg tre knapper. En av knappene er for å opprette nye grupper, en er for å endre status på grupper og siste er for å søke etter grupper. Hovedsiden for grupper er bygd opp av en liste bestående av emnets grupper. Hver rad i listen er en ny gruppe. Hver rad inneholder tre kolonner. Første kolonne viser gruppens navn. Andre kolonne beskriver hvilke brukere som er med i gruppen. Siste kolonne viser gruppens status.



Figur 5.20: Gruppeliste for emner

Når en bruker velger å opprette en ny gruppe for et emne får hen opp et nytt grensesnitt (figur 5.21). Øverst på dette grensesnittet finner vi sidens verktøylinje. Denne verktøylinjen starter med en pil mot venstre. Denne pilen vil føre brukeren tilbake til listen over grupper. Videre kommer sidens overskrift "Add Group", med tilhørende underoverskrift som vil bestå av emnets navn. Til venstre i verktøylinjen finner vi et søkefelt. Dette søkefeltet brukes til å søke etter medlemmer som ikke har gruppe. Hovedsiden for å lage nye grupper inneholder tre elementer. Øverste element er et felt som beskriver gruppens medlemmer. Her vil gruppemedlemmene vises samtidig som de blir valgt inn i gruppen. Det vil også være mulig å fjerne medlemmer fra gruppen her. Under gruppens medlemmer kommer det et tekstfelt. Dette tekstfeltet skal inneholde gruppens navn. Bak tekstfeltet er det en "Create Group" knapp. Denne knappen vil være deaktivert frem til gruppen har noen medlemmer, samt at gruppen har et navn. Under gruppens navn finner brukeren en liste med emnets medlemmer. Dette er medlemmer som ikke har gruppe fra før. Når brukeren velger å legge til en bruker i gruppen forsvinner brukerene fra listen og kommer frem i feltet over gruppens medlemmer.



Figur 5.21: Oppretting av ny gruppe

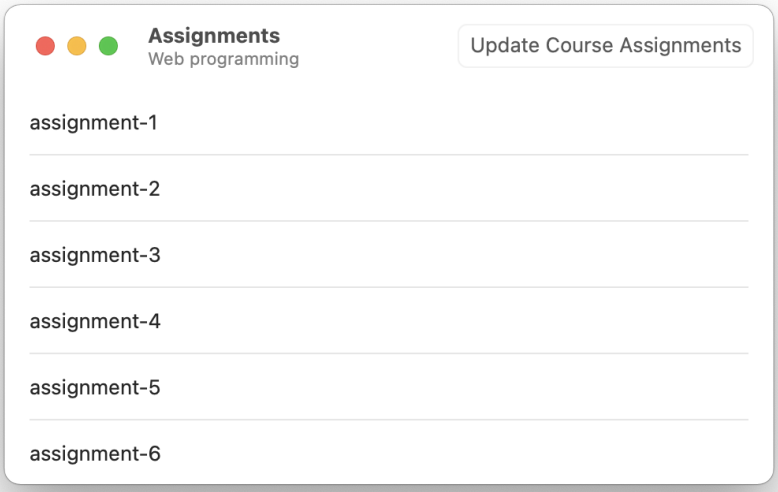
5.5.4 Innleveringer

Det er viktig for en lærer og studentassistent å ha oversikt over de ulike innleveringene i emnet. Hvilke innleveringer som må rettes manuelt og når de forskjellige innleveringene har frist for levering. På applikasjonens side for innleveringer vil brukeren få en oversikt over de forskjellige innleveringene i emnet.

Design

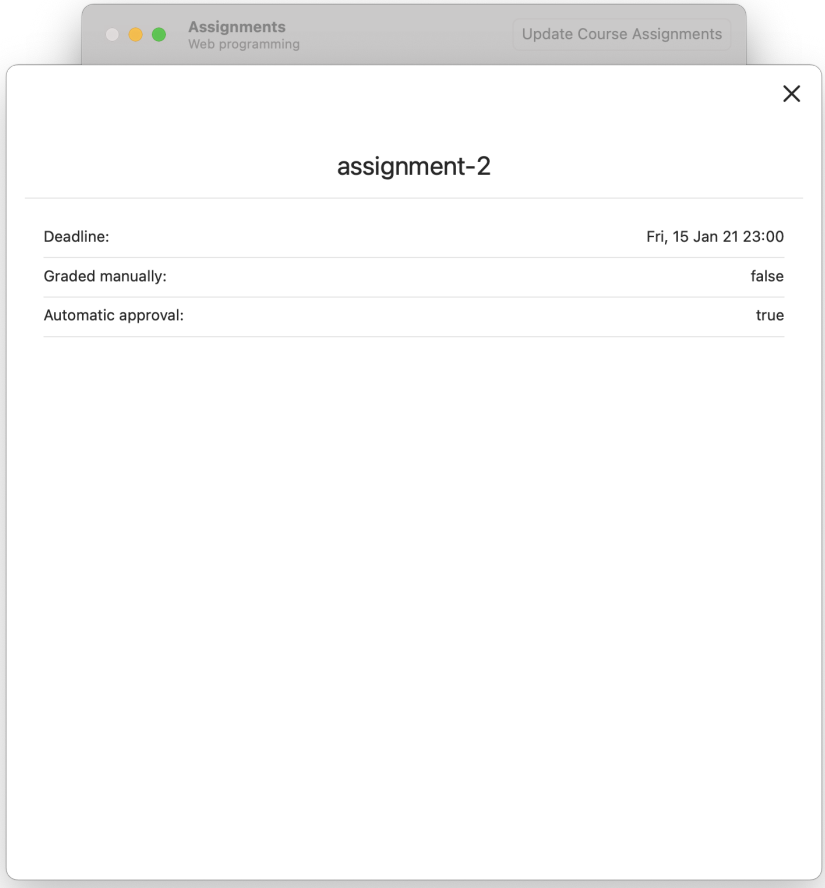
Siden for innleveringer starter på lik måte som de andre grensesnittene for lærere og studentassistenter. Øverst på siden befinner det seg en verktøylinje (figur 5.22). Denne verktøylinjen inneholder sidens overskrift "Assignments" etterfulgt av underoverskriften som inneholder emnets navn. Til venstre i verktøylinjen finner vi en "Update Course Assignments"-knapp. Denne kna-

ppen oppdaterer alle innleveringene på klientsiden hvis det er gjort endringer på serversiden. Hovedsiden for innleveringer er bygd opp av en liste med alle innleveringene i faget. Innleveringene i denne listen er trykkbare.



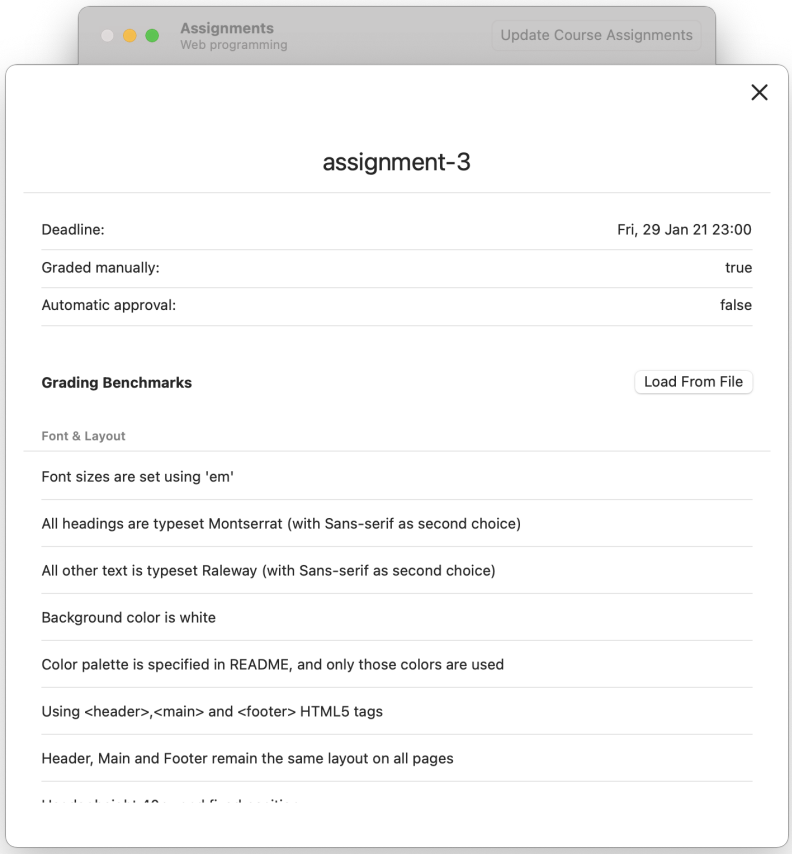
Figur 5.22: Oversikt over emnets innleveringer

Etter en bruker har valgt innlevering vil hen få opp informasjon om den valgte innleveringen. Denne informasjonen blir plassert i et forgrunnsbilde (figur 5.23). Denne informasjonen starter med innleveringens navn. Videre følger en liste med innleveringens leveringsfrist, om innleveringe skal manuelt karaktersettes og om innleveringen skal rettes manuelt.



Figur 5.23: Oversikt over emnets innleveringer

Når innleveringene skal manuelt rettes vil det bli lagt til enda mer informasjon. Som vist i figur 5.24 får brukeren opp en liste med overskrift "Grading Benchmarks". Denne listen beskriver innleveringes kriterier. Dette er kriteriene som skal benyttes når innleveringene skal rettes manuelt.



Figur 5.24: Oversikt over emnets manuelt rettet innleveringer

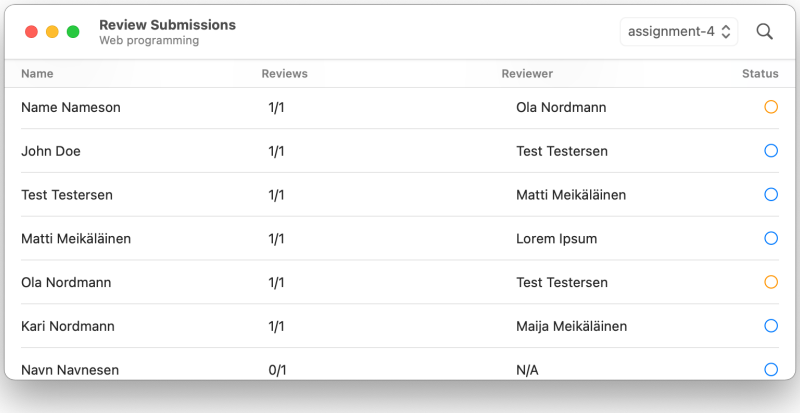
5.5.5 Manuell retting

Noen innleveringer skal rettes manuelt av lærere og studentassistenter. På siden over manuelt rettet innleveringer trenger brukeren en måte for å se hvilke innleveringer som trenger retting, hvilke kriterier som gjelder for innleveringen og en metode for å rette innleveringene.

Design

Øverst på siden for manuelt rettet innleveringer (figur 5.25) finner vi en verktøylinje. Denne verktøylinjen inneholder sidens overskrift "Review Submissions" etterfulgt av underoverskriften som inneholder emnets navn. Til høyre i verktøylinjen finnes det en flervalgsmeny, sammen med et søkefelt. Flervalgsmenyen indikerer hvilken innlevering brukeren skal rette. I søkefeltet kan brukeren søke etter medlemmer som har levert innleveringen som er valgt i flervalgsmenyen. Hovedsiden til manuelt retting er bygd opp av en liste med alle som har levert innleveringen valgt i flervalgsmenyen. Hver rad i listen beskriver et medlem. Hver rad består av fire kolonner. Første kolonne er brukerens navn, andre kolonne er antall personer som har og skal rette innleveringen, tredje kolonne beskriver hvem som har rettet innleveringen og siste kolonner viser et symbol for status til innleveringen. Statussymbolene for manuelt rettet er som følger:

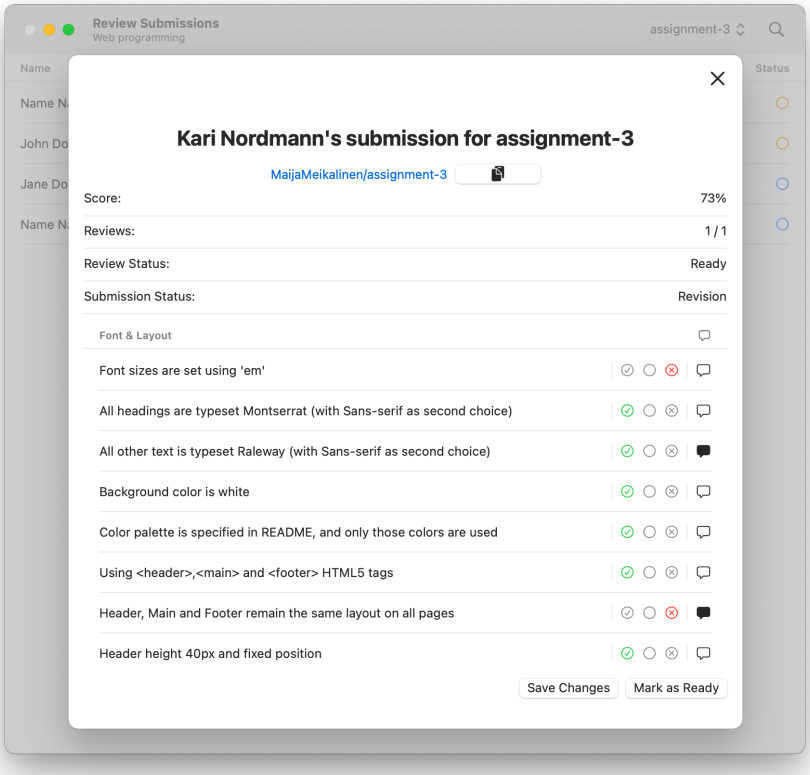
- **Blå ring** - Mangler retting
- **Oransje ring** - Prøvd å få godkjent, et nytt forsøk



Figur 5.25: Liste over medlemmer som trenger manuelt rettet innlevering

For å rette en innlevering må brukeren velge hvilken innlevering som skal

rettes. Dette gjøre med å trykke på raden for den ønskede innleveringen. Etter innleveringen er valgt vil den åpne seg i et forgrunnsvindu. Som vist i figur 5.26 starter dette forgrunnsvinduet med brukerens navn sammen med hvilken innlevering som holder på å rettes. Under dette er det plassert en HTTP lenke sammen med en knapp. HTTP lenken fører brukeren til innleveringens plassering på GitHub. Knappen kopierer brukerens repository navn til utklippstavlen slik at lærere og studentassistenter enkelt kan klone GitHub repositoryet til innleveringen. Etter dette kommer det frem et skjema. Dette skjemaet inneholder hvilken poengsum, i prosent, innleveringen har, hvor mange som har rettet innleveringen, om innleveringen har blitt markert som klar og status på innleveringen. Hva som kommer etter dette skjemaet avhenger av om det er andre brukere som har rettet innleveringen før. Om det er andre som har rettet innleveringen tidligere vil brukeren få opp en tekst hvor det står "This assignment is reviewed by:"etterfulgt av navnet på brukeren som har rettet innleveringen. Om innleveringen ikke har blitt rettet tidligere vil brukeren få opp en "Create Review"-knapp. Med å trykke på denne knappen vil hen starte rettingen av den valgte innleveringen, og få opp listen med kriterier for innleveringen.



Figur 5.26: Manuell retting av innlevering

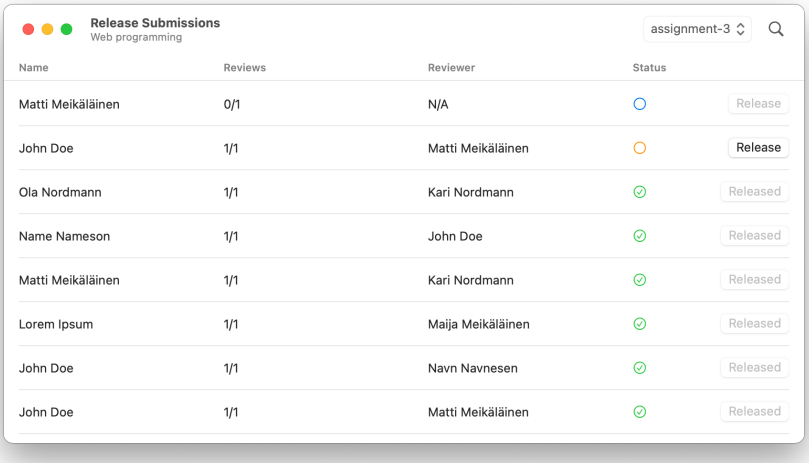
Som vist i figuren over har hvert kriterium en liste med ikoner plassert til høyre. Ikonlisten er delt inn i to seksjoner. Første seksjon er bygd opp av tre ikoner. Et for godkjent, et for ikke karakter og et for ikke godkjent. Her velger brukeren status på de ulike kriteriene. I andre seksjon av ikonlisten finnes det et meldingikon. Ved å trykke på dette ikonet kan brukerene legge til en kommentar til kriteriet. Helt i bunne av listen vil brukeren få mulighet til å legge til en kommentar som gjelder hele innleveringen. Under listen er det plassert to knapper. Den ene knappen vil lagre endringene som har blitt gjort i forbindelse med rettingen. Den andre knappen vil markere innleveringen som ferdig. Da vil innleveringen bli sendt videre til utlevering (seksjon 5.5.6).

5.5.6 Utlevering av innlevering

Etter en innlevering har blitt rettet, og markert som ferdig, blir den sendt til utlevering. Her vil rettelisten gjennomgås av en faglærer. Faglærer vil til slutt sette den endelige statusen på innleveringen. Faglærer kan her gi innleveringen status som godkjent, ikke godkjent og lever på nytt. Dette er siste ledd i behandlingen av en manuelt rettet innlevering.

Design

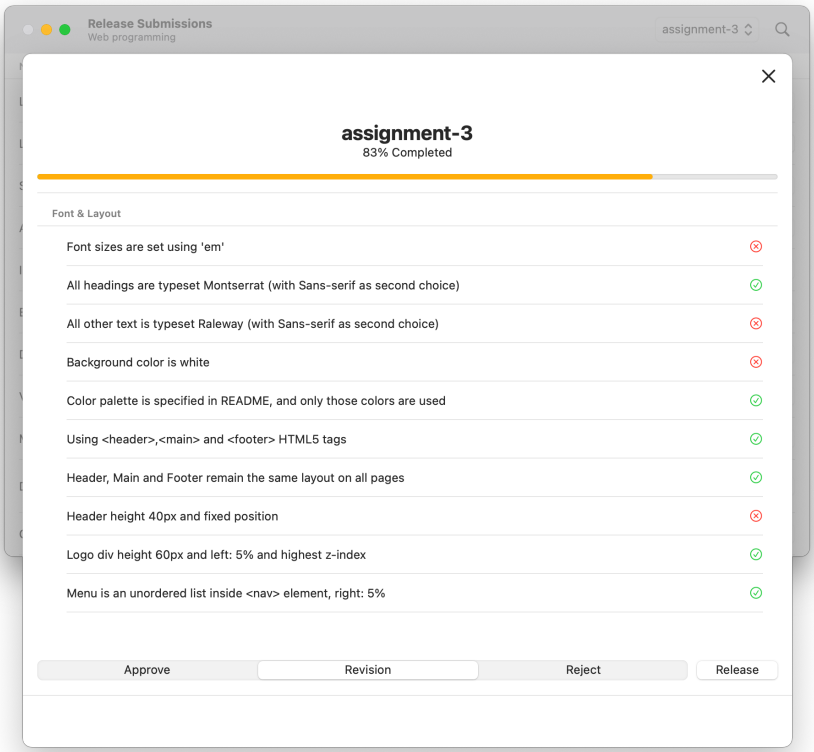
Som vist i figur 5.27 har denne sidene en verktøylinje plassert øverst på siden. Denne verktøylinjen er helt identisk med den forklart i seksjon Manuell retting (seksjon 5.5.5). Flervalgsmenyen i verktøylinjen for utlevering og for manuell retting deler samme innlevering. Om en bruker endrer innlevering på manuell retting siden vil samme innlevering vises på utleveringsiden. Utlevering av innleveringsiden bestå av en liste med innleveringer som er rettet manuelt og som trenger karakter. Hver rad består av hvem som har levert innleveringen, hvor mange som har rettet innleveringen, hvem som har rettet innleveringen, tidligere status for innleveringen og en knapp for å utlevere innleveringen til studenten.



Figur 5.27: Utlevering av innlevering

5.5 Lærer

Faglæreren kan se gjennom rettingen av innleveringen ved å trykke på personens navn. Som vist i figur 5.28 åpnes rettingen i et forgrunnsvindu som vil være lik tilbakemeldingsiden til en student. Under tilbakemeldingslisten finner brukeren en flervalgsmeny. Her kan faglæreren velge hvilken status innleveringen skal ha.



Figur 5.28: Forgrunnsvindu i Utlevering av innlevering

5.6 Administrator

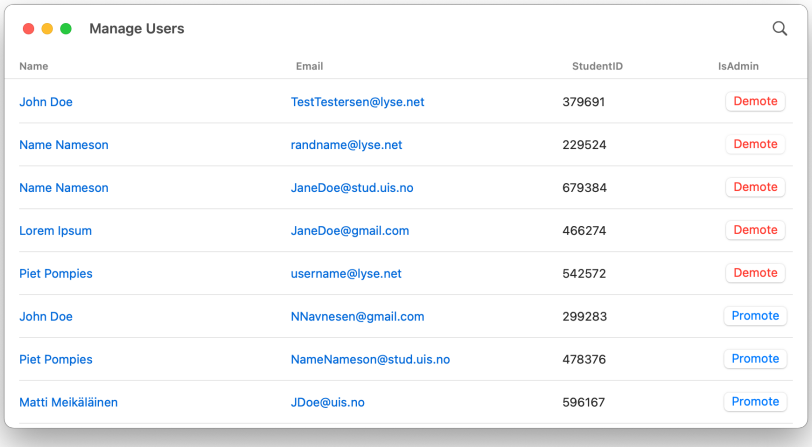
Som sett i seksjon 5.3 får en bruker med brukerrollen Administrator mulighet til å navigere seg inn på to ekstra deler av QuickFeed applikasjonen. Her kan brukeren navigere seg inn til en liste over brukere eller en liste med de forskjellige emnene QuickFeed tilbyr. På disse sidene får brukeren mulighet til å kunne administrere QuickFeeds brukere og emner.

5.6.1 Brukere

En Administrator av QuickFeed vil til enhver tid ha oversikt over hvilke brukere som er registrert på applikasjonen. Hen vil også ha oversikt over hvilke brukere som har administrator rettigheter og hvilke som ikke har.

Design

Siden hvor administratorer får en oversikt over brukerene er bygd opp av en liste hvor hver nye rad beskriver en ny bruker. Hver rad er bygget opp av fire kolonner. Første kolonne er navnet på brukeren, andre kolonne er email-adressen til brukeren, tredje kolonne er brukerens student ID og siste kolonne viser om brukeren er administrator eller ikke. For å oppnå en oversiktlig liste, er listen sortert på navn og administrator rettigheter. Alle administratorer kommer først i listen, før alle ikke administratorer.



Figur 5.29: Grensesnitt for å administrere brukere

Funksjonalitet

Brukerenes navn er markert i blått. Navnene til brukerene er en HTTP lenke som fører administratorer til brukerens GitHub profil. Mail-adressen til brukerne er også bygd opp som en lenke. Ved å trykke på mail-adressen vil QuickFeed automatisk sende deg til din standard mail applikasjon. Her åpnes det en ny e-post hvor din standard mail-adresse står som avsender og brukeren du trykket på står som mottaker.

Administratorer kan også gjøre andre brukere til administartorer (Promote) eller fjerne administrator rettighetene til andre brukere (Denote). Dette gjøre ved at brukeren trykker på knappen i siste kolonne av hver bruker. En administrator kan ikke fjerne sin egen administrator rolle.

Som forklart over i delkapittelet "Design" vil listen over brukere være sortert på brukerenes navn sammen med brukerrolle. Kodevisningen av "filteredUsers()" 5.9 viser at funksjonen først sortere ut i fra brukerrolle. Videre sjekker funksjonen hvilke navn som skal først og sist. Da er listen sortert. Etter dette filtrerer funksjonen ut personer som passer sammen med informasjonen i søkefeltet på siden.

```
1  func filteredUsers() -> [User] {
2      var users = viewModel.users!
3      users.sort {
4          if $0.isAdmin != $1.isAdmin{
5              return $0.isAdmin && !$1.isAdmin
6          } else {
7              return $0.name.trimmingCharacters(in: .
       whitespacesAndNewlines) < $1.name.trimmingCharacters(in: .
       whitespacesAndNewlines)
8          }
9      }
10     return users.filter({ matchesQuery(searchQuery: searchQuery,
        user: $0) })
11 }
```

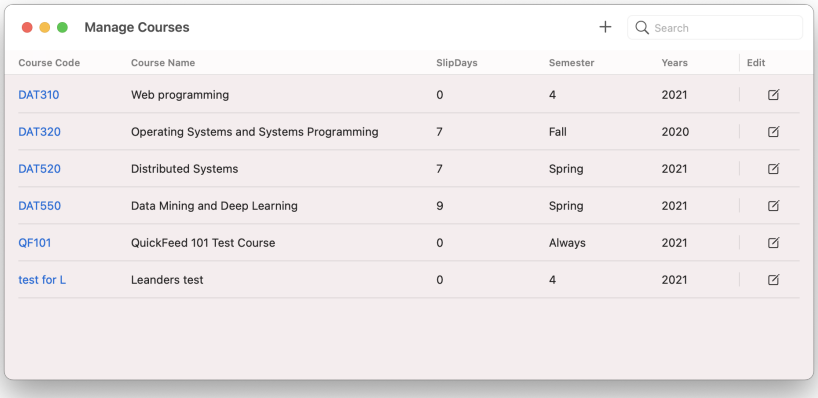
Kode 5.9: Sortere og filtrere brukere hentet fra AdminUsers.swift

5.6.2 Emner

På lik linje med siden for å administrere brukere vil administrator også ha en side hvor hen kan ha oversikt og administrere alle emner på QuickFeed. En administrator vil også ha muligheten til å opprette nye emner for kommende semestere.

Emneoversikt

Som vist i figuren 5.30 får brukeren opp relevant informasjon om de forskjellige emnene QuickFeed tilbyr. Øverst på siden finner vi sidens verktøylinje. Til venstre i verktøylinjen finner man sidens overskrift, som for emneoversikt er "Manage Courses". Til høyre finner vi er knapp merket "+". Denne knappen gir brukeren mulighet til å lage nye emner på QuickFeed. Rett bak "+"-ikonet finner vi et søkefelt. Søkefeltet brukes til å søke etter relevante emner. Selve hoved siden er bygget opp av en list med hver rad som nytt emne. Hver rad er igjen bygget opp av seks kolonner. Første kolonne for hvert emne inneholder emnets kode, andre kolonne er emnets navn, tredje er antall slipDays som vil være tilgjengelig for emnet, fjerde vil beskrive hvilket semester emnet er for, femte er hvilket år emnet er for og til slutt en knapp som kan brukes for å endre på det gitte emnet.



Figur 5.30: Oversikt over alle emner på QuickFeed

Som vist i figur 5.30 er alle emnekodene blå. Alle emnekodene er HTTP lenker som sender brukeren til emnets GitHub Organisasjon.

Når en bruker benytter seg av søkefeltet vil alle andre emner som ikke passer søketeksten forsvinne fra listen. Søkefeltet kan brukes til å søke på emne kode, emne navn, antall slipDays, hvilket semester og hvilket årstall emnet undervises. I kodeutdraget 5.10 vises koden som brukes til å filtrere ut i fra søkefeltet. Her tar funksjonen inn inneholdet i søkefeltet og et emne. Videre sjekker funksjonen om søkefeltet passer emnets kode, navn, slipDays, semester eller år. Hvis ingen av disse matcher returnere funksjonen at emnet ikke passer innholdet søkefeltet.

```
1  func matchesQuery(searchQuery: String, course: Course) -> Bool{
2      if searchQuery == ""{
3          return true
4      }
5
6      if course.code.lowercased().contains(searchQuery.lowercased
       ()){
7          return true
8      }
9
10     if course.name.lowercased().contains(searchQuery.lowercased
       ()){
11         return true
12     }
```

```
13
14      if String(course.slipDays).contains(searchQuery.lowercased()
        ){
15          return true
16      }
17
18      if course.tag.lowercased().contains(searchQuery.lowercased()
        ){
19          return true
20      }
21
22      if String(course.year).lowercased().contains(searchQuery.
        lowercased()){
23          return true
24      }
25
26      return false
27  }
```

Kode 5.10: Søkefelt sjekk for emner hentet fra HelperFunctions.swift

Siden emnekoder er mye brukt til å referere til de ulike emnene, har vi valgt å sortere listen over de ulike emnene etter emnekoden. Dette gjør at brukere raskt kan finne frem til riktig emne. Kodeutdraget 5.11 viser at emnene først sorteres ut i fra emnekode, for å så filtrere ut innhold som passer søkefeltet.

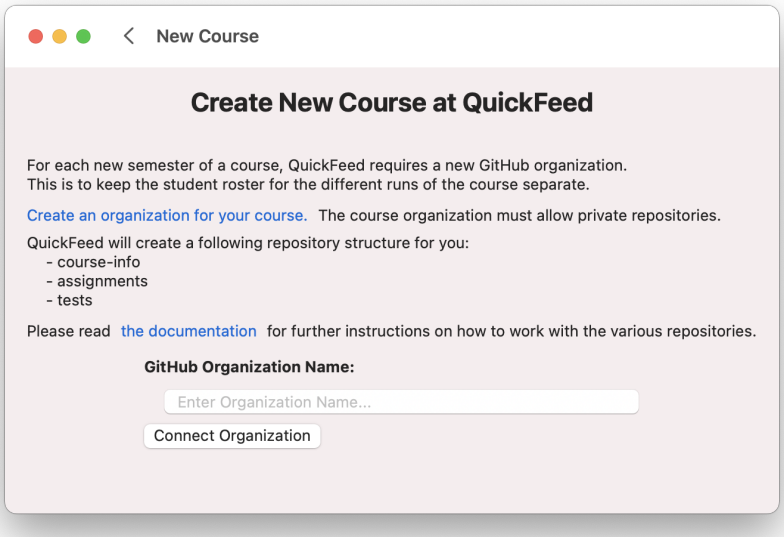
```
1 func filteredCourse() -> [Course] {
2     var courses = viewModel.courses!
3     courses.sort { $0.code < $1.code }
4     return courses.filter({ matchesQuery(searchQuery:
      searchQuery, course: $0) })
5 }
```

Kode 5.11: Sortere og filtrere emner for administrator hentet fra AllCourses.swift

Ved å benytte seg av "+"-ikonet i verktøylinjen eller endre knappen i en av emneradene vil brukeren bli sendt til siden hvor brukeren kan lage nye kurs eller siden for å endre kurs. Mer om hvordan disse sidene er designet, og hva slags funksjoner disse har, vil bli presentert i seksjon "Nytt Emne" og seksjon "Endre Eksisterende Emne".

Nytt Emne

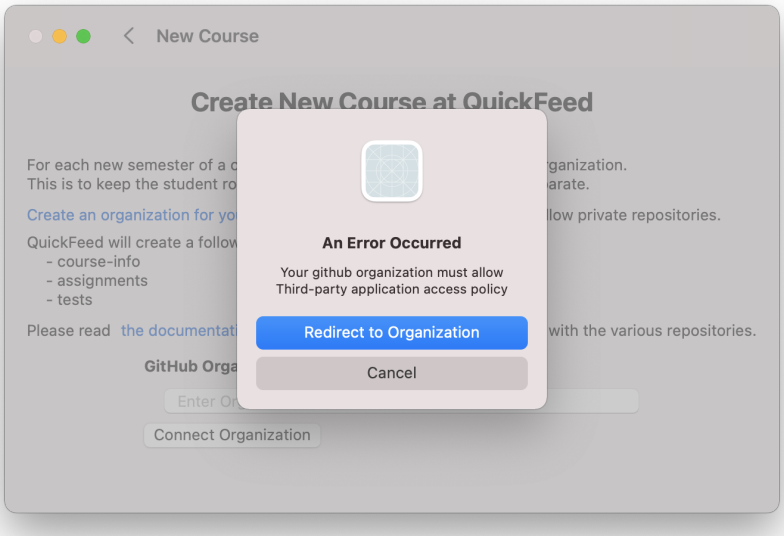
Når en administrator skal opprette et nytt emne er det noe informasjon som administratoren må være klar over. Som du kan se ut i fra figur 5.31 kommer all den relevante informasjonen før administratoren kan lage emnet. Etter denne informasjonen er lest, og administratoren har opprettet en GitHub Organisasjon for emnet, koble GitHub Organisasjonen sammen med QuickFeed.



Figur 5.31: Opprett nytt emne på QuickFeed

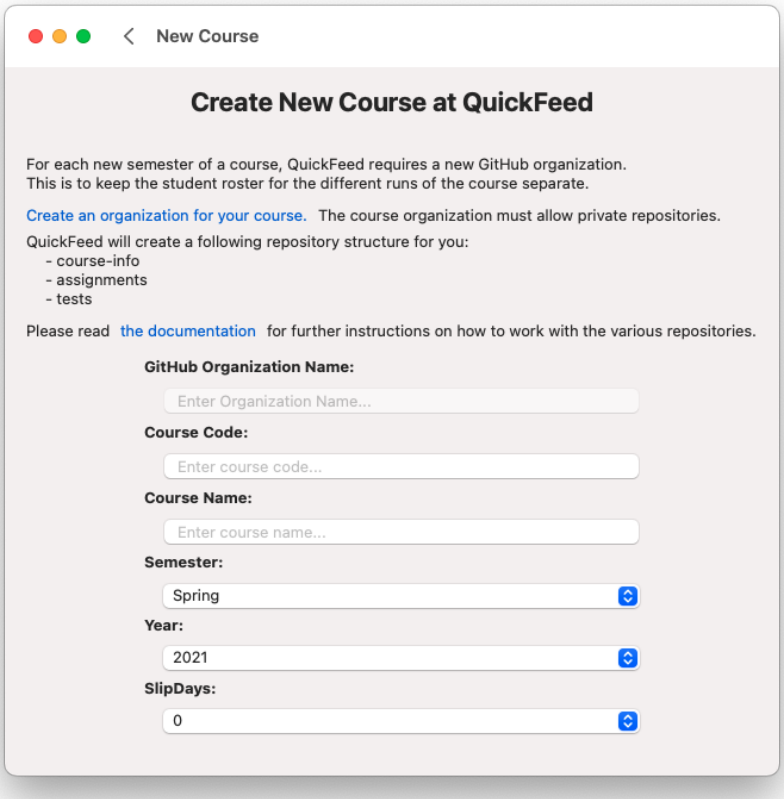
Etter en bruker har opprettet en GitHub organisasjon for QuickFeed emnet sitt må hen huske å skru på at tredjeparts applikasjoner kan bruke organisasjonen. GitHub organisasjoner har som standard deaktivert muligheten for tredjeparts applikasjoner. Hvis GitHub organisasjonen ikke tilater tredjeparts applikasjoner kan ikke QuickFeed oppnå kommunikasjon med organisasjonen. Om dette er tilfelle når en administrator av QuickFeed prøver å koble QuickFeed til GitHub, vil brukeren få opp en feilmelding som forklarer problemet. Som vist i figur 5.32 vil brukeren få to valg i feilmeldingen. Brukeren kan enten velge å kansellerer feilen, da vil hen ikke få

mulighet til å opprette sitt nye emne. Brukeren kan også velge "Redirect to Organization", denne knappen vil føre brukeren direkte til GitHub siden for organisasjonen hvor hen kan aktivere tredjeparts applikasjoner.



Figur 5.32: Tredje parts feil med GitHub organisasjonen

Etter administratoren har fått koblet opp GitHub Organisasjonen sammen med QuickFeed vil GitHub organisasjons feltet bli deaktivert. Hen får da opp et skjema, som vist i figur 5.33, hvor hen må fylle ut relevant informasjon for det nye emnet. Skjemaet starter med to tekst felt som skal inneholde emnets kode og emnets navn. Begge tekstfeltene vil være utfylt med en beskrivende tekst slik at det ikke er noe tvil om hva feltene skal inneholde. Etter tekstfeltene har skjemaet tre flervalgsmenyer. Første flervalgsmeny skal beskrive hvilket semester emnet skal undervises, andre flervalgsmeny beskriver hvilket år emnet skal undervises og siste flervalgsmeny skal brukeren si hvor mange slipDays det skal være for studentene i emnet.



Figur 5.33: Opprett nytt emne etter godkjent GitHub Organisasjon

Flervalgsmenyen for semester inneholder tre valg. Her kan brukeren velge mellom semesterene "Fall", "Spring" og "Always". Vi valgte å løste valg av semester på denne måten for å oppnå en lik skrive måte for semester. I dagens Web-applikasjon er feltet for å fylle inn semester ett tekstfelt. Her godtar applikasjonen alt som kan passe inn i en streng variabel.

Flervalgsmenyen for år inneholder tre valg. Her vil det være mulig for brukeren å registrere emner på QuickFeed to år frem i tid. Det vil ikke være mulig å velge år som har vært. I 2020 vil det altså ikke være mulig å registrere emner for 2019. På samme måte som forklart for semester tilbyr Web-applikasjonen ett tekstfelt for å fylle inn årstall. Her godtar klienten

alt som kan passe inn i en streng. Vi har valgt å finne frem til riktig år med funksjonen i kodevisning 5.12. Her finner funksjonen ut hvilket år vi befinner oss i nå. Etter dette går den gjennom tre runder i en for-loop. Hver runde legges årstallet inn i en liste med årstall for å så øke årstallet med en. Listen med årstall blir så lagre i en @State variabel.

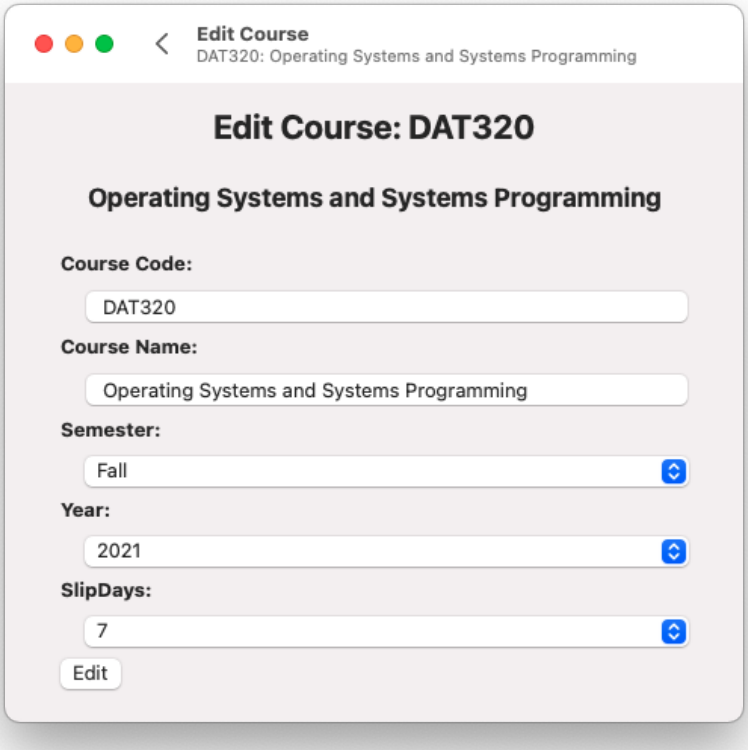
```
var yearsArray: [String] = []
let year = Calendar.current.component(.year, from: Date())
for index in 0...2 {
    yearsArray.append(String(year + index))
}
self.years = yearsArray
```

Kode 5.12: Funksjon for å finne to år frem i tid hentet fra CourseFields.swift

Etter skjemaet er ferdig utfylt vil det komme frem en knapp som gir brukeren mulighet til å registrere emnet på QuickFeed.

Endre Eksisterende Emne

Til tider vil det være behov for administratorer å måtte endre på informasjonen til registrerte emner. Når en administrator trykker på et emne for å rediger det vil det åpnes en ny side i applikasjonen. Som figur 5.34 viser vil tittelen på siden være "Edit Course: " etterfulgt av emnets kode og emnets navn. Med denne informasjonen i tittelen vil brukeren aldri være i tvil om hvilket emne hen er i ferd med å endre. Etter tittelen kommer samme skjema som forklart i delkapittelet "Nytt Emne", men informasjonen om emnet vil allerede være ferdig utfylt. På den måten trenger brukeren bare å endre på det som skal endres, alt annet vil være som det var fra før.



Figur 5.34: Endre eksisterende emne på QuickFeed

Om et emne som skal endres på ikke inneholder alternativene som er tilbutt i flervalgsmenyene vil det som standard bli valgt “Always“ for semester og nåværende år for “Year“.

5.7 iOS-støtte

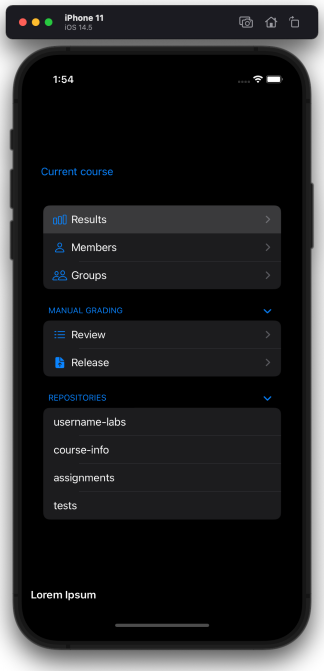
For å kjøre applikasjonen på iOS-plattformen, må iOS legges til som en målarkitektur for Xcode-prosjektet. Ved flere målarkitekturer vil det opprettes egne mapper for arkitekturspesifik kildekode, og en mappe for delt kildekode. For denne applikasjonen er det kun visningslaget som trenger arkitekturspesifike tilpasninger, siden visningsmodellene og hentingen av data fungerer på samme måte uavhengig av plattform.

Tiden ble for knapp for å tilpasse en fullstendig løsning for iOS, men utforsking av mulighetene ble gjennomført i et eget Xcode-prosjekt. Prosjektet ble satt opp som en multiplattformapplikasjon, og kildekoden fra macOS-prosjektet ble kopiert til den delte mappen i multiplattform-prosjektet. Ut av boksen produserte koden en rekke feilmeldinger, som skyldtes bruk av rammeverk som ikke er tilgjengelige for iOS. For eksempel benytter søkefeltet i applikasjonen AppKit, og siden AppKit ikke er kompatibelt med iOS må en egen UIKit-implementasjon tilpasses for iOS. Kodeutdrag 5.13 viser en skisse av hvordan plattformspesifike implementasjoner kan løses med betinget kompilering. I en slik løsning vil *SearchFieldiOS* bruke UIKit, og *SearchField* bruke AppKit.

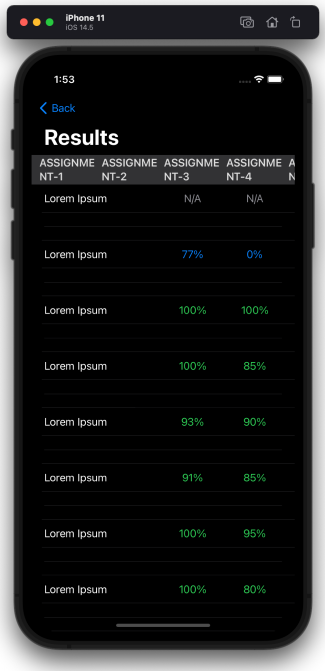
```
1 #if os(iOS)
2     SearchFieldiOS(query: $searchQuery)
3         .frame(minWidth: 200, maxWidth: 350)
4 # else
5     SearchField(query: $searchQuery)
6         .frame(minWidth: 200, maxWidth: 350)
7 # endif
```

Kode 5.13: Betinget kompilering i Swift

Med bruk av betinget kompilering fikk vi fjernet alle feilmeldingene, og kjørt applikasjonen. Figur 5.35 og 5.36 viser grensesnittet for henholdsvis navigasjonen, og resultatsiden ved simulering i iOS, uten tilpasninger av visninger. Som man kan se behøves ytterligere arbeid for å oppnå et brukervennlig grensesnitt.



Figur 5.35: Navigasjonsmeny i iOS



Figur 5.36: Resultatliste i iOS

5.8 Fremtidig arbeid

Applikasjonen i sin nåværende tilstand kan kobles opp mot serveren, og har den samme funksjonaliteten som web-applikasjonen for automatisk rettede innleveringer. Likevel står det igjen en del arbeid for at applikasjonen skal bli klar til distribusjon.

5.8.1 Sikkerhet og autentisering

For at kommunikasjonen mellom klienten og serveren skal være sikret tilstrekkelig må enkelte endringer gjøres både på server-siden, og klient-siden. gRPC-koblingen bør med bruk av SSL og TLS benyttes til autentiseringen av applikasjonens brukere, samt til å kryptere data.

5.8.2 Strømming over gRPC

For å optimalisere brukeropplevelsen bør strømming av enkelte datatyper implementeres. Om klienten og serveren holder en strømningskanal åpen når klienten kjøres, kan serveren ta seg av oppdatering av klientens informasjon når informasjonen oppdateres på serversiden. Med en slik løsning kunne en student lastet opp endringene sine fra en terminal-applikasjon, og fått en notifikasjon fra applikasjonen i det øyeblikket serveren har testet ferdig koden.

5.8.3 Manuell retting

For manuelt rettede innleveringer er brukeropplevelsen enda ikke optimal. Mye av ansvaret for logikken ligger på klient-siden. For eksempel må endringer i hvordan poengsummen utregnes implementeres i begge klient-applikasjonene, noe som gjør kodebasen komplisert å vedlikeholde.

En løsning på dette problemet er å flytte deler av logikken over til serversiden. For eksempel kan serveren ta seg av oppretting av Review-strukturen, etter forespørsel fra klienten. Endringer for hvert enkelt kriterium kan sende en oppdatering til serveren, hvor serveren regner ut ny poengsum, og sender oppdatert informasjon tilbake til klienten. Strømming av data kan også forbedre brukeropplevelsen for manuell retting, ettersom flere brukere ofte retter samtidig. Oppdatert informasjon om hvilke innleveringer som har påbegynte rettinger vil gjøre samarbeidet enklere.

For manuelt og automatisk rettede innleveringer er brukstilfellene veldig ulike. Flere av feltene i strukturene som brukes i begge tilfeller er relevant til kun det ene brukstilfellet. Proto-filen kunne markert skillet mellom en manuelt rettet og automatisk rettet innlevering tydeligere.

5.8.4 Distribuering

For å distribuere en macOS-applikasjon til potensielle brukere har man to overordnede valgmuligheter. Man kan enten stå for distribueringen selv,

eller la AppStore ta seg av distribueringen. Uansett trenger man en betalt utviklingskonto fra Apple.

For å distribuere applikasjonen utenfor AppStore, må den først signeres med den betalte utviklingskontoen. Om applikasjonen distribueres uten signatur vil *Gatekeeper*, en sikkerhetsmekanisme innebygd i macOS, forhindre kjøring av applikasjonen for alle brukere.

For distribuering på AppStore må applikasjonen i tillegg sendes inn for en gjennomgang av Apple. For godkjenning må applikasjonen bestå en lang liste med krav som går på både sikkerhet, forrettnigsmodell, design, lovverk, og ytelse. Fordelen med å distribuere på AppStore er at det er enklere å nå ut til brukeren med fremtidige oppdateringer. I tillegg er det en nødvendighet for å utvide til iOS og iPadOS. Ulempen er at applikasjonen må godkjennes av Apple, i en potensielt tidkrevende prosess.

Kapittel 6

Konklusjon

Hovedmålet som ble definert i innledningen ble til en viss grad oppnådd. Brukergrensesnittet gir den samme funksjonaliteten relatert til automatisk retting som web-applikasjonen. Når det kommer til manuell retting er ikke funksjonaliteten konsistent med web-applikasjonens nåværende funksjonalitet, og videre arbeid må gjennomføres for at applikasjonen pålitelig kan håndtere manuell retting. Videre arbeid kreves også for autentiseringsmekanismen, og for distribuering av applikasjonen.

Optimaliseringen i delkapittel 4.4 angående lasting av resultater er ikke direkte ny funksjonalitet, men kan likevel ses på som oppnåelse av en sekundær målsetning. Resultatene lastes mye raskere, og forbedrer brukeropplevelsen for faglærere og studentassistenter i enkelte emner. Støtte for iOS er utforsket og utprøvd, men en brukbar løsning vil kreve tid å implementere. I tillegg må applikasjonen godkjennes til AppStore for å kunne distribueres på iOS-plattformen.

Bruk av gRPC-rammeverket i Swift-applikasjoner virker til å være lite utbredt, og informasjon om arkitekturmessige retningslinjer var nærmest fraværende. Likevel fikk vi implementert en fungerende løsning, og gRPC-metoder utføres pålitelig.

SwiftUI er et relativt nytt rammeverk, og har enkelte mangler sammenlignet med mer modne rammeverk som UIKit og AppKit. Å inkludere AppKit-

og UIKit-visninger er en oppnåelig prosess, men utgjør en flaskehals i utviklingsprosessen. Spesielt dersom utvikleren ikke har kjennskap til rammeverkene fra tidligere prosjekter. Sammenlignet med mer modne rammeverk er det også lite informasjon å finne i form av forum-diskusjoner på tjenester som *StackOverflow*. Samtidig er SwiftUI-syntaksen enkel å bli kjent med, og man kommer raskt opp på kunnskapsnivået som kreves for å komme i gang med utvikling av brukergrensesnitt.

Bibliografi

[1] Decision tree for how to define your swiftui properties (figur). https://twitter.com/chriseidhof/status/1280433133813456896/photo/1.

[2] The model-view-viewmodel pattern. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

[3] The state of developer ecosystem 2020. https://www.jetbrains.com/lp/devecosystem-2020/.

[4] Swiftnio: Design philosophy. https://github.com/apple/swift-nio#design-philosophy.

[5] Where the world builds software. https://github.com/about.

[6] Vera Yaseneva Daniel Urdal. Experience report - replacing rest with grpc in autograder. Technical report, Universitet i Stavanger, 2019. Bacheloroppgave.

[7] Heine Furubotten. The autograder project: Improving software engineering skills through automated feedback on programming exercises. Technical report, Universitet i Stavanger, 2015. Masteroppgave.