



## Gruppe 5 - Skiltsøker'n

MOB3000 - Applikasjonsutvikling for  
mobile enheter

Daniel Feili - 246992

Endre Kvam - 251876

Håvard Garsrud - 251875

Oscar Skauge Eriksen - 247004

Philip Toan Nguyen - 161218

<b>1. Introduksjon</b>	<b>4</b>
<b>2. Funksjonell beskrivelse</b>	<b>4</b>
2.1 Sammenlign	4
2.2 Favoritter	5
2.3 Logge inn/Logge ut/Registrere bruker	5
2.4 Innstillinger	6
2.5 Søke på et kjøretøy	6
2.6 Søke på et kjøretøy med kamera	7
<b>3. Designvalg</b>	<b>7</b>
3.1 Frontend	7
3.1.1 Jetpack Compose	7
3.1.2 Material Design 3	8
3.1.3 Navigering	8
3.2 Backend	8
3.2.1 Retrofit	8
3.2.2 Firebase	9
3.2.3 Google ML Kit	9
3.2.4 Accompanist-Permissions	9
3.3 Wireframe	9
<b>4. Teknisk systemdokumentasjon</b>	<b>12</b>
4.1 UseCase-diagram	12
4.2 Utlasseringsdiagram	13
4.3 Klassediagram	14
4.4 Teknisk informasjon	15
<b>5. Brukerveiledning</b>	<b>15</b>
5.1 Hjem	15
5.2 Søk	16
5.3 Sammenlign	16
5.4 Logg inn/Register	16
5.5 Profil	17
5.6 Favoritter	17
5.7 Kamera	18

<b>6. Arbeidsprosess og innsats / Individuelt</b>	<b>19</b>
6.1 Daniel Feili	19
6.1.1 Kodebidrag	19
6.1.2 Kodesnutt	20
6.1.3 Dokumentasjon	21
6.2 Endre Kvam	22
6.2.1 Kodebidrag	22
6.2.2 Kodesnutt	23
6.2.1 Dokumentasjon	24
6.3 Håvard Garsrud	24
6.3.1 Kodebidrag	24
6.3.2 Kodesnutt	25
6.3.3 Dokumentasjon	28
6.4 Oscar Skauge Eriksen	29
6.4.1 Kodebidrag	29
6.4.2 Kodesnutt	30
6.4.3 Dokumentasjon	31
6.5 Philip Toan Nguyen	32
6.5.1 Kodebidrag	32
6.5.2 Kodesnutt	33
6.5.3 Dokumentasjon	34
6.6 Arbeidsinnsats	34
<b>7. Læring og refleksjon</b>	<b>34</b>
<b>8. Referanser og kilder</b>	<b>36</b>

# 1. Introduksjon

Skiltsøker'n er en mobilapplikasjon som gir brukere mulighet til å søke på norske skiltnummer, sammenligne kjøretøy og lagre favoritt-kjøretøy. Brukeren skal også kunne søke på skiltnummer ved bruk av kameraet på telefonen sin. Data som vises blir hentet fra Statens Vegvesen. Brukerdata og favoritter er lagret i Firebase. Applikasjonen er utviklet i Android Studio og er skrevet i Kotlin.

Vi er en gruppe på fem studenter ved Universitetet i Sørøst-Norge i Bø som har utviklet denne applikasjonen i høstsemesteret 2023.

# 2. Funksjonell beskrivelse

Alle Composable-klassene som vises på skjermen bruker BottomNav.kt fra mappen Navigation. Dette er for å enkelt kunne navigere til "Start", "KameraSkjerm" og "Settings", uansett hvor i appen brukeren befinner seg.

## 2.1 Sammenlign

Sammenlign-siden er laget for å sammenligne to kjøretøy side om side. Her skriver bruker inn to gyldige skiltnummer, og API-kallet fra Start-siden blir gjenbrukt for å vise frem informasjon fra Statens Vegvesen sitt API.

Funksjonen SammenlignSkjerm tar inn APIViewModel, som bruker funksjonen hentBillInfo(), en funksjon som returnerer variablene til et kjøretøy med funksjonen billInfoVariabler(). Denne ligger i klassen HentBillInfo.kt.

Knappen "Gå" kaller på funksjonen hentInfo(), som tar inn to skiltnummere som parameter, og starter med å hente informasjonen til disse som vist over. Hvis begge skiltnummerene er gyldige vil denne informasjonen vises under.

### Klassene som er brukt:

Sammenlign.kt og APIViewModel.kt, som igjen bruker de andre klassene i mappen DataApi.

## 2.2 Favoritter

Klassen Favoritter.kt sjekker først om brukeren er logget inn eller ikke. Dette skjer med å bruke Auth().currentUser fra klassen Auth.kt. Dersom man ikke er logget inn vises informasjon om hvordan man kan logge seg inn eller registrere ny bruker. Klassen bruker også NavController fra klassen Nav.kt for å navigere til enten "Login" eller "Register".

Dersom bruker er logget inn vises innholdet i FavoritterSkjerm.kt med FavoritterViewModel.kt som argument. Her lages en dropdown-meny med skiltnummer på alle kjøretøy brukeren har lagret som favoritt. Menyen hentes fra Meny.kt. Denne menyen blir populert ved bruk av hentFavoritter() i viewmodelen, som gjør en spørring mot databasen. Samtidig i viewModelen blir en liste med all data fra favorittene laget. Det blir altså ikke gjort API-kall på favoritter-siden, men alle favorittene hentes fra databasen.

Dersom man ikke har noen favoritter vises dette for bruker.

Når bruker trykker på et av skiltnummerene vises all informasjon om dette kjøretøyet under. Dette blir hentet fra listen "allefavoritter" fra FavoritterViewModel.kt. Ved sletting av favoritt brukes slettFavoritt() fra samme viewModel, som også gjør en spørring mot databasen.

### Klassene som er brukt:

Auth, Favoritter, FavoritterSkjerm, FavoritterViewModel, og Nav.

## 2.3 Logge inn/Logge ut/Registrere bruker

Applikasjonen kan brukes enten med eller uten innlogging. Når en person er innlogget, vil hen ha mulighet til å bruke favoritter-funksjonen, for lagring og sletting av favoritt-skilt. Brukerhåndtering er gjort via Firebase. Alle hovedklassen har en viewmodel som har en dataklasse med variabler. Disse er alle mutable og forandres når input i hovedklassene blir forandret. Vi har også registrer- og login-funksjonene i ViewModelene til sidene. Disse kjøres og tar variablene som kommer inn i dataklassen. I klassen Auth ligger hovedfunksjonene som er hentet fra Firebase-biblioteket. Metodene i Auth er lagbruker og login fra Firebase, disse metodene kjøres i ViewModel'ene i en try-catch med riktige variabler som blir skrevet inn av bruker. I hovedklassen kjører vi ViewModel'en med hovedklassen og lager en instance av dataklassen.

**Klassene som er brukt her er:**

Auth

Login: Login/Login og Login/LoginViewModel

Registrer: Registrer/Register og Registrer/RegisterViewModel

## 2.4 Innstillinger

På applikasjonen har vi en profilside. Her skal man kunne gjøre endringer i forhold til profilen sin, som å endre passord, slette bruker, ta på light/dark mode, og endre språk. Vi har brukt komponent struktur der alle komponenter kjøres på en slutt funksjon. Dette gjorde at koden var mye mer ryddigere og at den var lettere å jobbe med siden det var i biter. Vi har tatt i bruk funksjoner fra auth som slettbruker, bytta passord og logg ut bruker.

**Klasser som blir brukt:**

Innstillinger

## 2.5 Søke på et kjøretøy

Alle funksjonene i applikasjonen er basert rundt å søke på et kjøretøy. Denne funksjonaliteten er løst ved å bruke Retrofit. Retrofit bygges først ved hjelp av klassene nevnt under, og det er også laget get request og ViewModel for get request. Dette kalles på i SokerInfo via en LaunchedEffect, og tar teksten personen skrev på hjem skjermen/kamera som input. Responsen blir så satt i en metode som returnerer et objekt med variablene som skal vises frem.

**Klasser som blir brukt:**

Start (Her skrives skilnummeret inn)

DataAPI/RetrofitInstance (Her bygges Retrofit)

DataApi/Interceptor (Legger til API nøkkelen som SVV på RetrofitInstance)

DataApi/DataInterface (For å kjøre API kallet)

DataApi/APIViewModel (ViewModel for APIKallet)

DataModeller (Mappe med et hierarki av dataklasser, som responsen fra API kallet blir satt inn i.)

DataApi/HentBillInfo (Returnerer et objekt med kun de variablene vi trenger fra API responsen)

## 2.6 Søke på et kjøretøy med kamera

Applikasjonen kan bruke telefonen sitt kamera for å søke på et kjøretøy, i stedet for å bruke tekstfeltet. Dette er gjort med AndroidX Camera2, som gir Jetpack Compose tilgang til telefonen sitt kamera, og Google ML-Kit som gjenkjenner tekst fra kameraet. Første gang brukeren går inn på denne funksjonen, vil det bli spurt om å gi tilgang til kameraet. Dette er gjort ved hjelp av Google sitt Accompanist-bibliotek. Når et skiltnr er funnet av kameraet kan man trykke på søkeikonet, og man vil da få opp info om dette kjøretøy. Hvis teksten kameraet finner ikke er et gyldig skiltnummer, vil man få opp skjermen som vises hvis bilskiltet er ugyldig.

### Klassene for denne funksjonaliteten:

Kamera/Kamera (Sjekker om brukeren har gitt tillatelse til kamera. Hvis ikke blir man sendt til IngenTillatelseSkjerm, hvis man har det blir man sendt til KameraSkjerm)

Kamera/KameraSkjerm (Setter opp Ulet, og bruker de andre klassene)

Kamera/TekstGjenkjenner (For å kjenne igjen tekst fra kameraet)

Kamera/IngenTillatelseSkjerm (Skjerm for å tillate kamera)

## 3. Designvalg

### 3.1 Frontend

#### 3.1.1 Jetpack Compose

Vi har valgt å bruke Jetpack Compose for å bygge applikasjonen vår. Det er en relativt ny måte å lage Android-applikasjoner på, og erstatter den gamle måten der man skrev i XML-filer. Compose ble utgitt i mars 2021 (Okan, 2022). Hvis man er kjent med

web-rammeverk, kan Compose ligne på React og Vue. Man lager komponenter, såkalte Composable-funksjoner, som bygger opp UI’et del for del.

### 3.1.2 Material Design 3

Material Design 3 er den nyeste designsystemet fra Google. Det består av en rekke ferdigbygde komponenter, farger og typografi slik at man får en konsekvent stil på komponentene man velger å bruke i applikasjonen sin. Det ble utgitt litt senere enn Jetpack Compose, i slutten av 2021 (Kozłowski, 2023). Noen av endringene fra Material Design 2 inkluderer dynamiske farger basert på brukerens bakgrunn, forskjellige figurer og endrede komponenter (Kozłowski, 2023).

### 3.1.3 Navigering

For å navigere i applikasjonen vår har vi tatt i bruk en NavController, som er det sentrale API’et for å navigere i en Jetpack Compose applikasjon. Man trenger å implementere androidx sin navigation:navigation-compose i build.gradle-filen til applikasjonen for å bruke dette (Android Developers, 2023).

Enhver NavController må være koblet sammen med en NavHost. I NavHost definerer man et valgt antall NavGraphBuilder.composable-funksjoner, som er ruter (routes) som det skal navigeres til. En rute er en ny Composable-funksjon som bygger opp UI’et for den siden det navigeres til.

Vi har valgt å ha en bottomBar med en NavigationBar, som er en komponent tilbudt av Material Design 3. Den består av tre knapper, der man enten kan navigere til “Hjem”, som er startsiden til applikasjonen, “Søk”, der man kan søke på skilnummer med kameraet, eller “Profil”, der man kan registrere seg, logge inn, eller endre språk eller utseende på appen. Denne bottomBaren er synlig på alle sidene i applikasjonen.

## 3.2 Backend

### 3.2.1 Retrofit

Det å hente data fra API til applikasjon kan være krevende. Vi har derfor tatt i bruk biblioteket Retrofit, som er en “typesikker” REST-klient for Android, Java og Kotlin (Ahmed, 2022). Det at den er typesikker, vil si at kompilatoren vil validere typer på kompileringsstadiet, og gi error hvis man gir feil datatype til en variabel (Wheeler, 2008). Retrofit tilbyr metoder som simplifiserer henting av data fra API’er, og denne dataen blir returnert i form av et JSON-objekt.

### 3.2.2 Firebase

Firebase er en gruppe med skybaserte verktøy tilbuddt av Google. Noen av verktøyene det tilbyr er autentisering, sanntidsdatabase og meldinger i skyen (Hanna, 2023). I applikasjonen vår har vi valgt å la Firebase håndtere autentisering og database. Firebase Auth tilbyr ferdiglagde funksjoner som blant annet lar brukeren opprette bruker, logge inn, endre passord, og slette bruker. Det å implementere et autentiseringssystem fra bunnen av er tidkrevende og vanskelig, så disse funksjonene sparer oss for mye tid. Databaseløsningen som vi har valgt å bruke heter Firebase Firestore og er en NoSQL-database.

### 3.2.3 Google ML Kit

For å hente ut skiltnummer i kamerafunksjonen til applikasjonen, har vi brukt Text Recognition v2 fra Google (Google Developers, 2023). Dette er et API som tilbyr funksjoner som lar deg trekke ut tekst fra et bilde. Vi har brukt dette sammen med AndroidX sitt Camera2, som gir Jetpack Compose tilgang til kameraet til enheten som brukes.

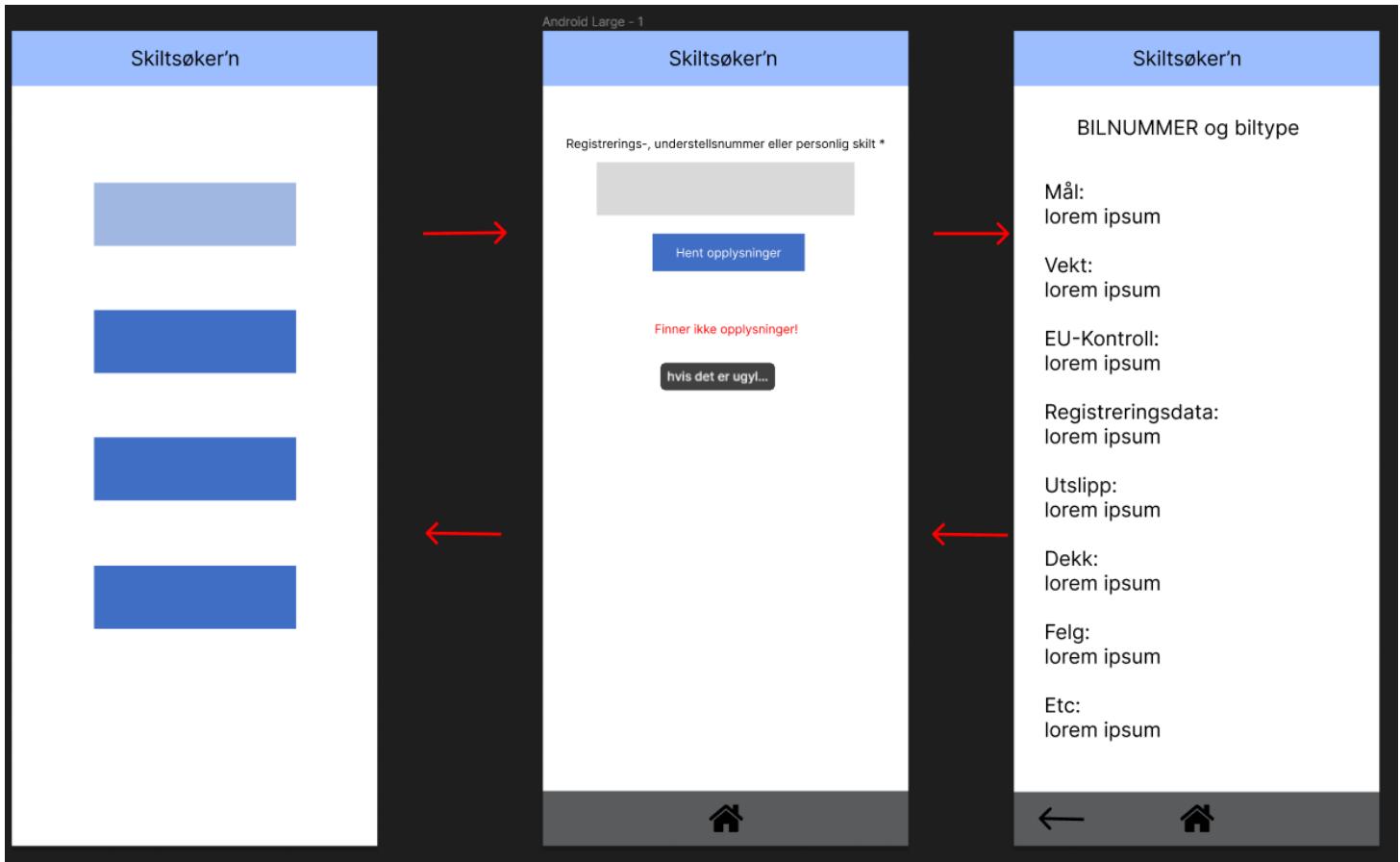
### 3.2.4 Accompanist-Permissions

For å kunne bruke kamerafunksjonen til applikasjonen, må den ha tilgang til kameraet. Dette har vi løst ved å bruke Accompanist fra Google, som er en gruppe med biblioteker som tilbyr vanlige funksjoner for Compose som ikke enda er implementert i Compose. For å håndtere tillatelser, har vi brukt Permissions-biblioteket.

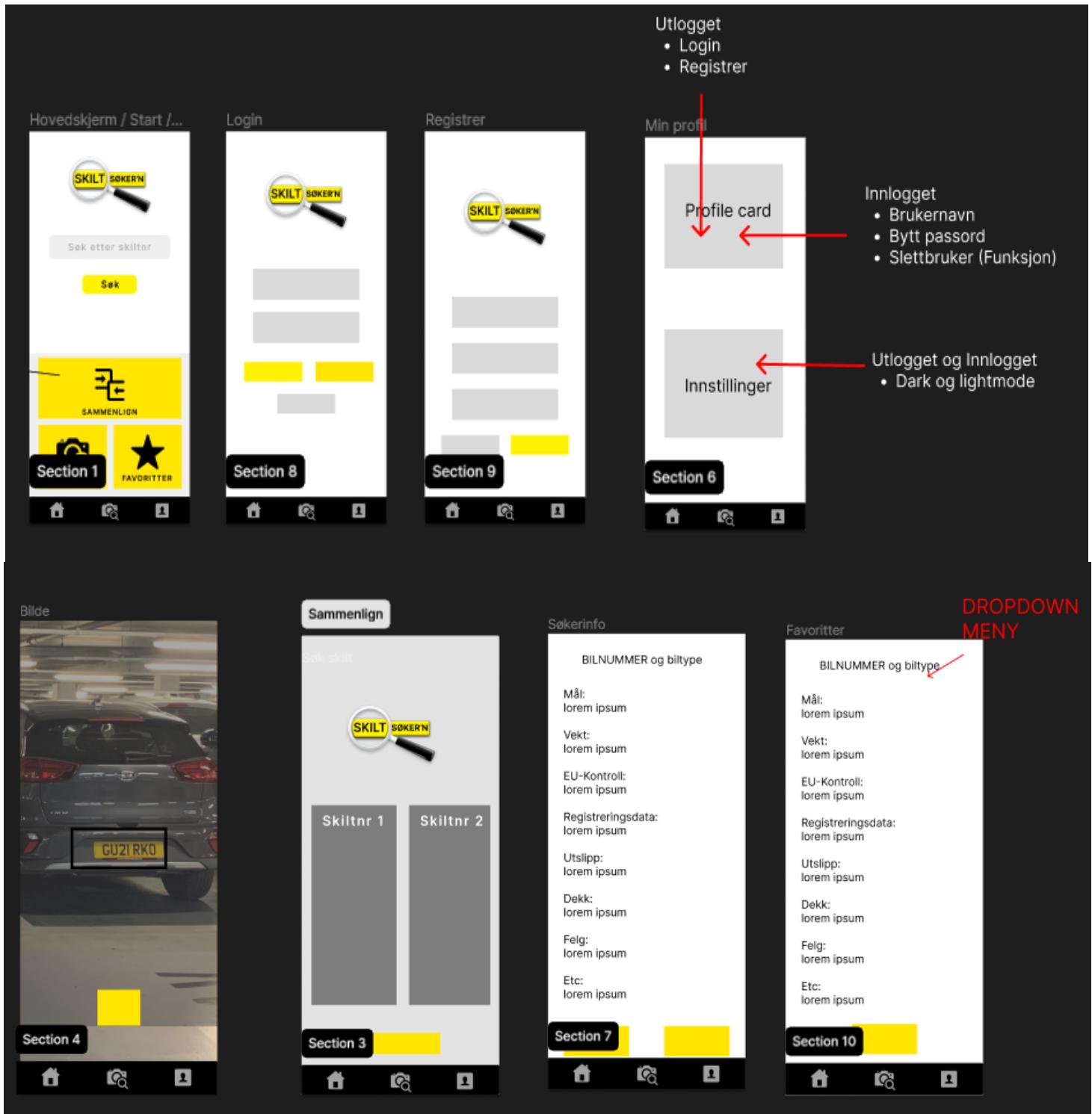
## 3.3 Wireframe

Etter at vi hadde bestemt oss for en oppgave, så ble vi enige i fellesskap å lage en wireframe/prototype. Figma ble tatt i bruk som verktøy. Vi hadde et møte hvor vi fokuserte på dette. Alle fikk uttrykke sine tanker og ideer til applikasjonens design. Vi har hatt ulike meninger, men kommet til en enighet på hvordan applikasjonen skal se ut.

Bildet under viser vårt første utkast av wireframen for applikasjonen. Dette var en relativt simpel wireframe som vi brukte for å komme i gang med applikasjonen.

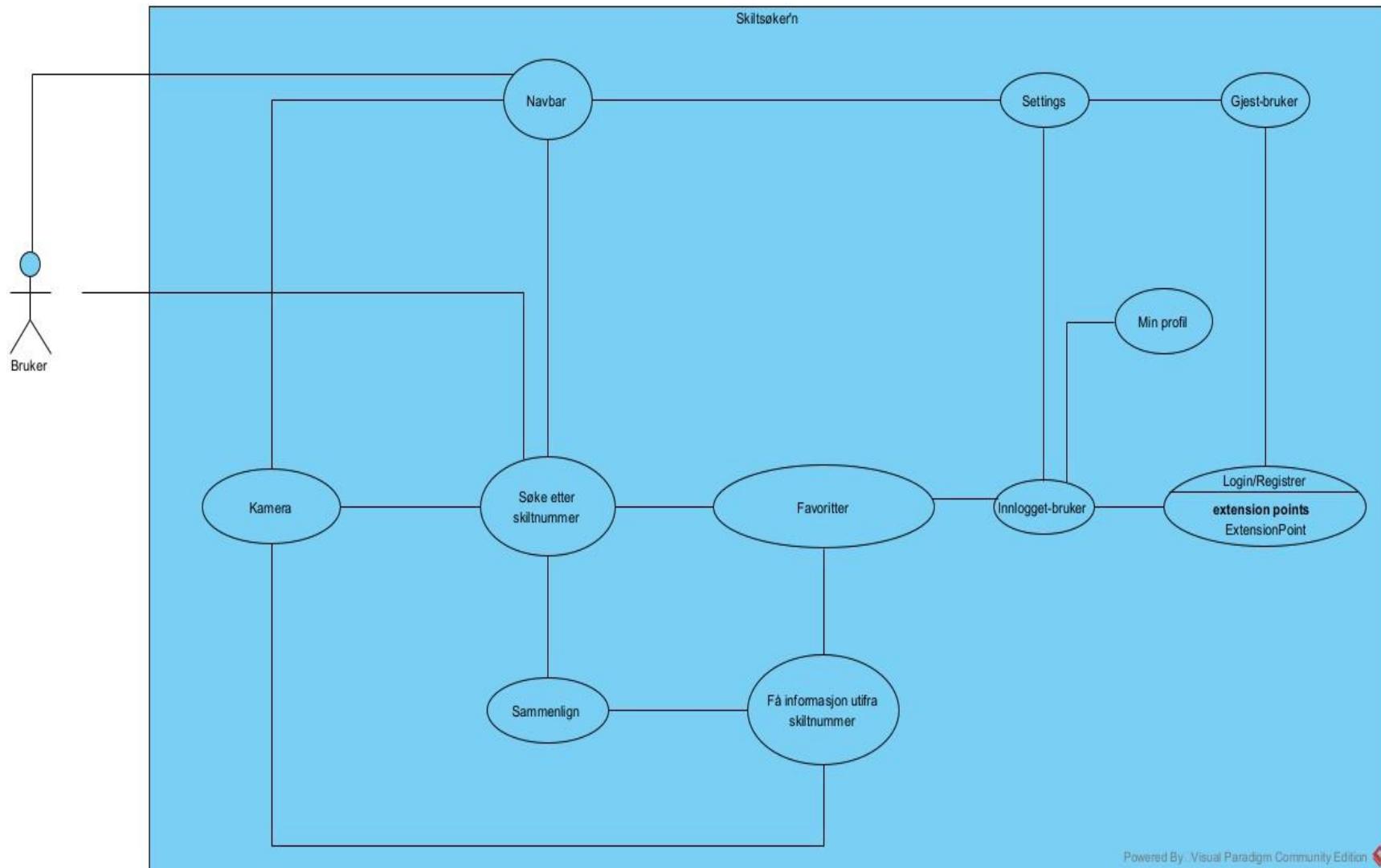


Bildene under viser vårt endelig utkast av wireframen. Dette var hvordan vi planla at den endelige applikasjonen skulle se ut.

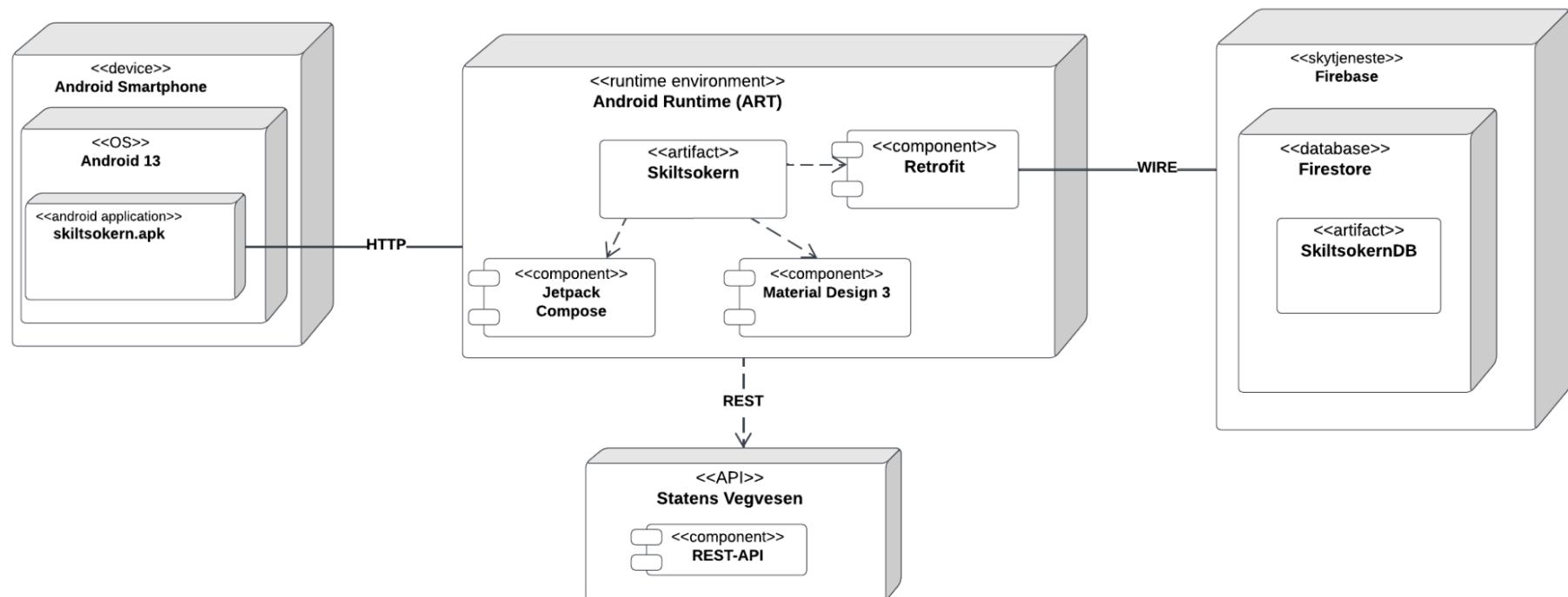


## 4. Teknisk systemdokumentasjon

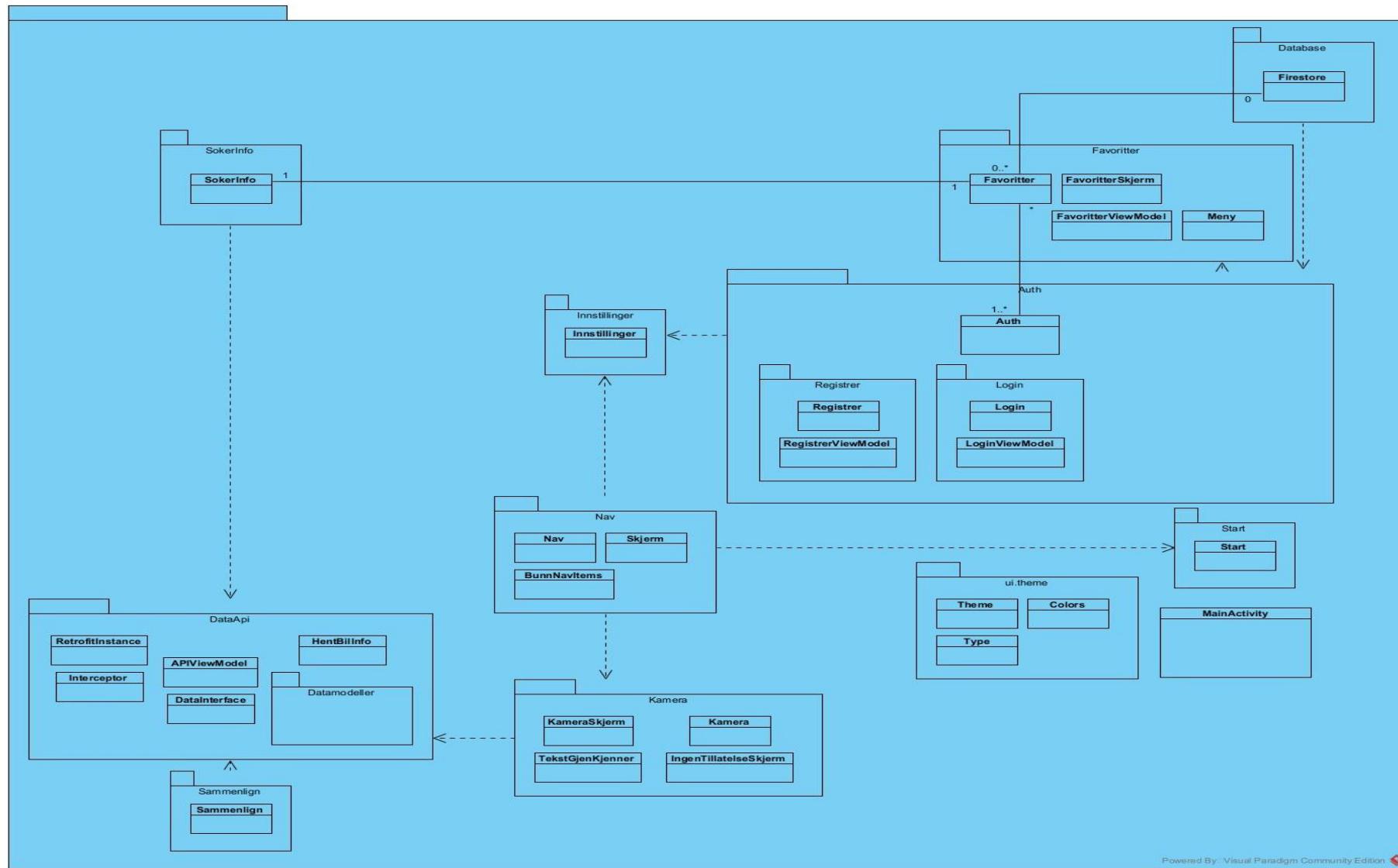
### 4.1 UseCase-diagram



## 4.2 Utplasseringsdiagram



## 4.3 Klassediagram



Powered By: Visual Paradigm Community Edition

## 4.4 Teknisk informasjon

For å kjøre applikasjonen, åpne skiltsøkeren.apk med Pakkeinstallasjon. Applikasjonen blir da lastet ned til din enhet. Mer detaljert veiledning er under Brukerveiledning.

Testbrukere i applikasjonen:

Testbruker 1:

brukernavn: [test1@test.no](mailto:test1@test.no)

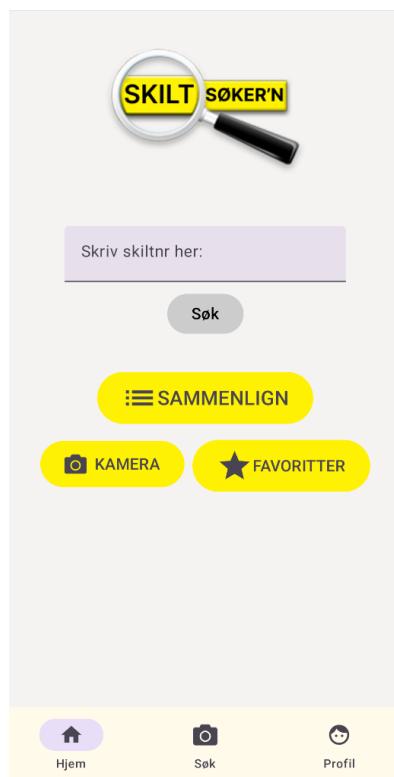
passord: test123

Testbruker 2:

brukernavn: [test2@test.no](mailto:test2@test.no)

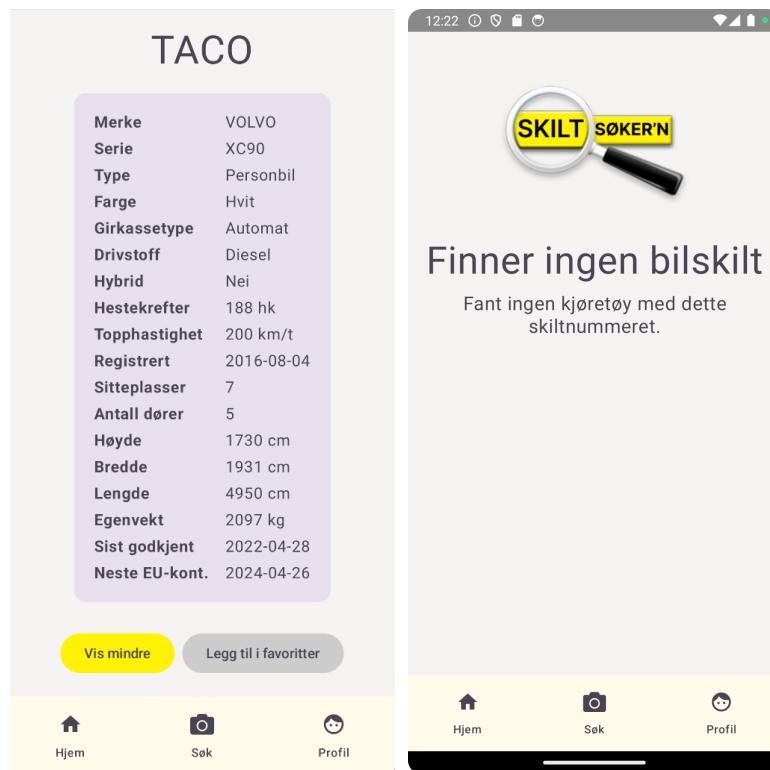
passord: test123

## 5. Brukerveiledning



### 5.1 Hjem

Dette er startsiden til applikasjonen, her kan du søke på skiltnummer og navigere til andre sider i applikasjonen. Det er i denne skjermen du kan komme videre til sidene Sammenlign og Favoritter.



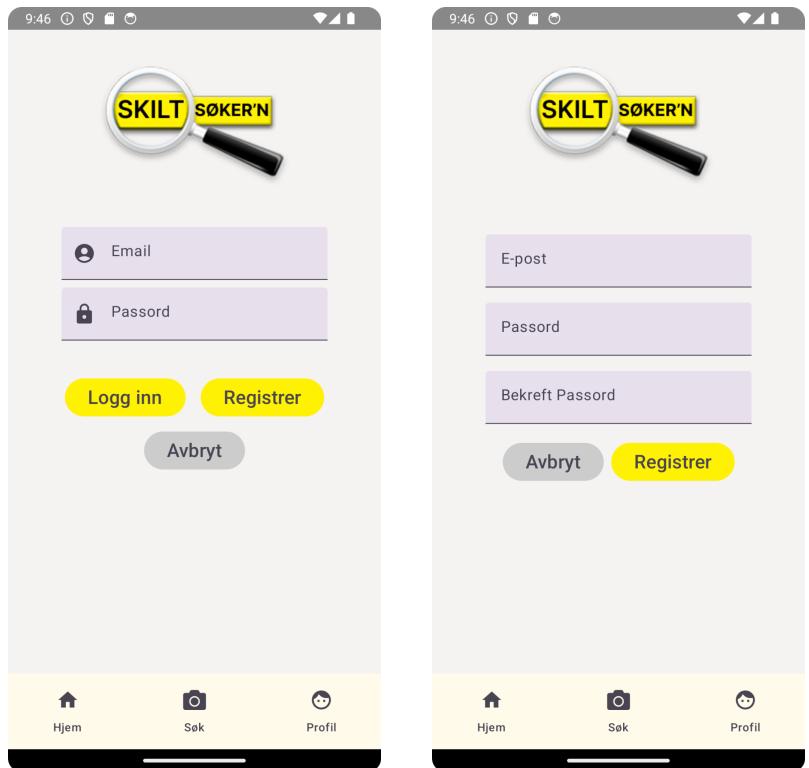
## 5.2 Søk

Dette er Søk-skjermen, denne skjermen vises når du har søkt på et skiltnummer. Her kan du trykke på "vis mer"-knappen for å se mer og legge skiltnummer til i dine favoritter. Legg til favoritter funker kun hvis du er logget inn. Hvis du skriver et ugyldig skiltnummer, så vil det komme opp en error melding om dette.



## 5.3 Sammenlign

Sammenlign-skjermen kommer du til ved å trykke på Hjem, også Sammenlign. På denne siden kan du sammenligne to forskjellige biler, slik at du kan se informasjonen side om side. Dette gjør du ved å skrive inn reg nr til bilene du vil sammenligne i de to forskjellige tekstfeltene, og trykker gå.



## 5.4 Logg inn/Registerer

Logg inn og registrer bruker finner du ved å trykke på Profil i navigasjonsbaren på bunnen. Her kan du bruke testBrukeren, eller velge å registrere en ny bruker.

TestBruker 1:

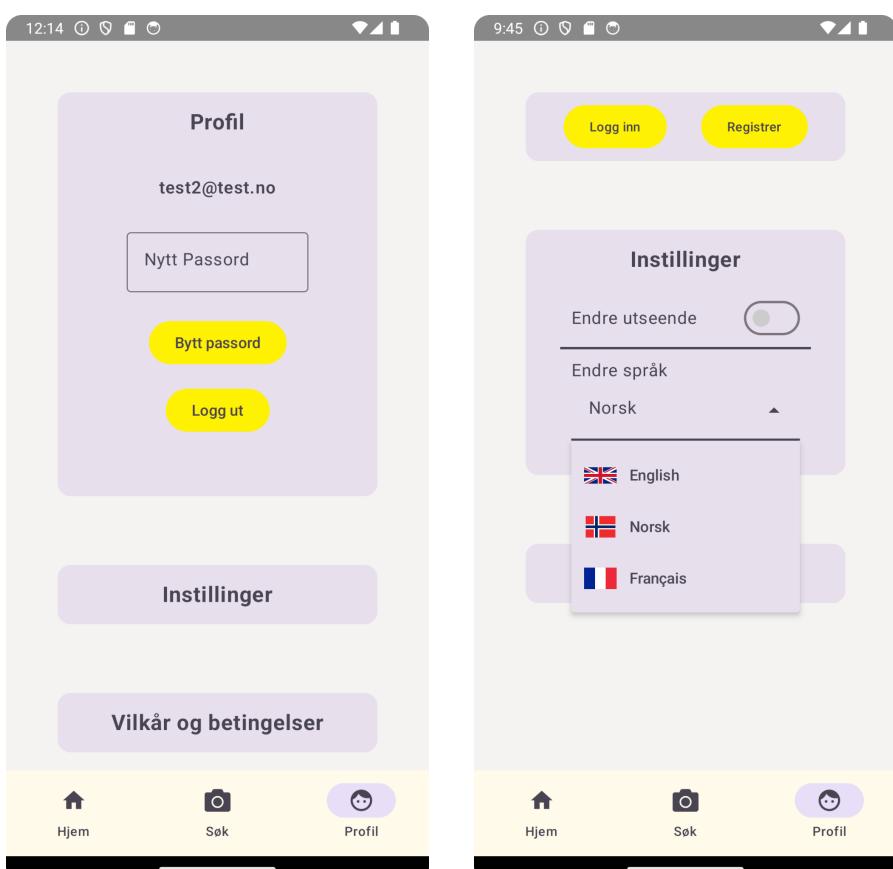
brukernavn: [test1@test.no](mailto:test1@test.no)

passord: test123

TestBruker 2:

brukernavn: [test2@test.no](mailto:test2@test.no)

passord: test123



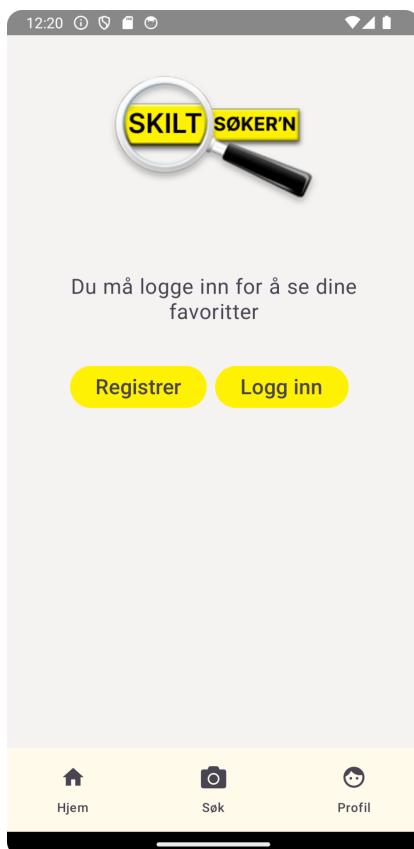
## 5.5 Profil

Her kan innstillingar for bruker og applikasjonen endres.

Denne siden kommer du til ved å trykke på profil navigasjonsbaren på bunnen. Hvis du ikke er logget inn, vil du få spørsmål om å logge inn eller registrere bruker. Når brukeren er logget inn, er det mulig å se hvilken bruker som er logget inn, samt endre passord, eller slette brukeren.

Det er også innstillinger for hele applikasjonen, light/dark mode og endre språk. Disse kommer du til ved å trykke på innstillingen, også kan man endre på hva man ønsker.

## 5.6 Favoritter

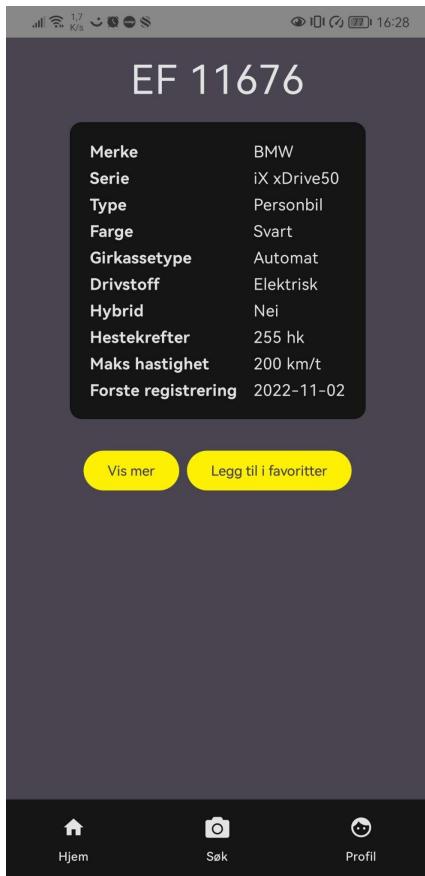


Favoritter-siden er til for å vise frem biler som brukeren har lagt til i favoritter tidligere (via søk-skjermen). Denne funksjonen er kun tilgjengelig for brukere som er logget inn. For å finne favoritter-siden, må man trykke på Hjem i navigasjonsbaren på bunnen, så trykke på Favoritter. Hvis personen er innlogget, og har lagret biler som favoritter, vil disse finnes i dropdown-menyen på toppen. Hvis brukeren ikke har noen favoritter, vil denne dropdown-menyen være tom.



## 5.7 Kamera

Kamerafunksjonen finner man ved å trykke på Søk (med kamera-ikon) på navigasjonsbaren på bunnen, eller ved å trykke på hjem i navigasjonsbaren på bunnen, også velge kamera søk. Her må du gi appen tillatelse til å bruke kameraet. Etter kameraet kan brukes i applikasjonen, er det bare å få teksten du vil søke i kameraet, og trykke på søker. Her er det viktig at det ikke er annen tekst i bildet, ettersom all tekst vil bli søkt på.



Etter man har trykket på forstørrelsesglasset, med tekst i kameraskjermen, blir man videresendt til "SokerInfo" med teksten i skjermen som skiltnummer.

## 6. Arbeidsprosess og innsats / Individuelt

### 6.1 Daniel Feili

#### 6.1.1 Kodebidrag

I dette prosjektet har jeg hatt ansvar for Login, Registrer, Auth og Setting. Jeg har også hjulpet med kode rundt andre steder i appen vår. Jeg lagde klassene Login/Login Viewmodel, Registrer/Registrer Viewmodel, Setting, Auth og lagde interceptor klassen.

#### 1. Auth

I Auth så har jeg alle metodene som skal hjelpe login og registrer klassene. Disse metodene er f.eks lagbruker og login. Det er også mange andre metoder som byttpassord og sjekk for innlogget bruker.

#### 2. Login/Loginviewmodel

Login/Login Viewmodel er klassen hvor jeg fikset alt med login. I Login klassen ligger alt med UI mens i Viewmodel har jeg satt opp alt som skjer under kjøringen og data som blir brukt for å logge inn en bruker. Det er noen funksjoner som er inne og ikke blir brukt, men er der for å vise at jeg kunne gjort mer med login om vi hadde hatt bruk for det.

#### 3. Registrer/ Registrerviewmodel

Disse er veldig lik login funker på samme måte med funksjoner fra Auth som blir brukt i bakgrunnen i registeret for å registrere bruker. Alt av verifisering blir kjørt i viewmodel og alle metodene som blir brukt i Registrer er inne i viewmodellen til klassen.

#### 4. Interceptor

Interceptor-klassen er en metode som fungerer som en mellom mann for api kallet vårt. Vi bruker interceptoren fordi vi vil at alle kallene skal ha SVV-auth og api key. En interceptor gjør det også mer ryddig i stedet for å ha header i hver request. Interceptor er fra okhttp.

## 5. Innstillinger

Innstilling klassen ble laget fordi vi trengte en side på appen hvor brukeren kan se profilen sin og forandre på settings inne på appen. Klassen består av flere Composable funksjoner som kjøres på slutt metoden som er den som vises fram. Du har også mulighet til å logge inn fra settings noe som betydder at jeg måtte sjekke om bruker har logget inn eller ikke før å vise riktig Card.

### 6.1.2 Kodesnutt

Kodesnutt for lagBruker-funksjonen. Denne funksjonen tar i bruk dataklassen og bruker lagBruker funksjonen fra auth klassen. For å lage bruker så setter vi inn variablene som kommer fra dataklassen.

```
fun lagBruker(context: Context) = viewModelScope.launch() {
    try {
        if (!validerRegister()) {
            reguiStatus = reguiStatus.copy(error = "Fyll inn alle
feltene")
        }

        auth.lagBruker(
            reguiStatus.emailReg,
            reguiStatus.passordReg
        )

        reguiStatus = reguiStatus.copy(registrert = true)
    } catch (e: Exception) {
        reguiStatus = reguiStatus.copy(error = "Kunne ikke lage
bruker")
    }
}
```

Kodesnutt for loginBruker. funksjonen funker på samme måte som over og bruker loginviewmodel i stedet for register viewmodel.

```
fun loginBruker(context: Context, navController: NavController) = viewModelScope.launch {
    try {
        if (!validerLogin()) {
            error("Fyll inn alle feltene")
        }
        val loginResultat = auth.login(uiStatus.email,
            uiStatus.passord)

        if (!loginResultat) {
            error("Feil brukernavn eller passord")
        } else {
            uiStatus = uiStatus.copy(loggetInn = true)
            navController.navigate(Skjerm.Start.ruter)
            Toast.makeText(context, "Logget inn",
                Toast.LENGTH_SHORT).show()
        }
    } catch (e: Exception) {
        uiStatus = uiStatus.copy(error = "Kunne ikke logge
            inn")
    }
}
```

### 6.1.3 Dokumentasjon

- Funksjonell beskrivelse
- Teknisk systemdokumentasjon
- Læring og refleksjon
- Referanser og kilder

## 6.2 Endre Kvam

### 6.2.1 Kodebidrag

Jeg har bidratt med klassene “Favoritter”, “Sammenlign” og “Firestore”, oppsett av database og spørninger mot disse. Har hjulpet på oppsett ved henting av informasjon gjennom API-kall. Har også deltatt på felles arbeid på de fleste klasser og funksjoner. Har testet appen grundig og funnet bugs og feil, for så å fikse disse.

#### 1. Database

Jeg satt opp databasen i Cloud Firestore, og lagde spørninger mot denne i klassen Firestore og FavoritterViewModel.kt. Lagde også funksjonen leggInnFavoritt() som brukes i SokerInfo.kt.

#### 2. Favoritter

Jeg lagde nesten alt i Favoritter-mappen. Det som vises når bruker ikke er logget inn, er ikke mitt verk, men alt utenom det har jeg laget selv. Dropdown-menyen lagde jeg ved hjelp av tutorials på youtube og stackoverflow. ViewModellen som henter data fra databasen, hvordan dataene vises og slette favoritt har jeg kodet selv.

#### 3. Sammenlign

Hvordan informasjonen vises har jeg laget, men hvordan dataen blir hentet gjennom API-kall er laget av Oscar og Philip. En tidkrevende variabel her var “hestekrefter” som ble hentet fra API’et. Variabelen ble hentet fra kun én motor, og dataen vi fikk var effekten til denne i kilowatt. Jeg konverterte dette til hestekrefter ved å gange kw med 1,34102209. Vi skjønte fort at appen viste feil antall hestekrefter på biler med flere motorer, så jeg lagde en måte å sjekke hvor mange motorer og hvilke typer drivstoff disse brukte. Kodet for dette ligger i HentBillInfo.kt. Appen viser nå med god nøyaktighet hvor mange hestekrefter et kjøretøy har, men det er mulig dataene fra Statens Vegvesen ikke er helt korrekte. Dette fant vi ut av med å sjekke dataene i SV mot biler vi har kjennskap til. Hestekreftene vi viser i appen har muligens en liten feilmargin mot Statens Vegvesen sine data på enkelte kjøretøy.

Sammenlign-siden ble laget ganske sent, og vi fikk ikke til å legge det inn i en ViewModel med tiden vi hadde igjen. Derfor blir ikke data lagret på skjermen hvis bruker snur på enheten etter hen har søkt på to bilskilt.

## 6.2.2 Kodesnutt

### Hente og slette favoritter i databasen

```
class FavoritterViewModel : ViewModel() {
    val allefavoritter = mutableStateOf<List<String>>(emptyList())
    val favoritterSkilt = mutableStateOf<List<String>>(emptyList())
    val brukerID = Firebase.auth.currentUser?.uid.toString()

    ▲ Endre Kvam
    init {
        hentFavoritter()
    }
    ▲ Endre Kvam
    private fun hentFavoritter() {
        val db = Firebase.firestore
        db.collection( collectionPath: "favoritter" ) CollectionReference
            .whereEqualTo( field: "brukerID", brukerID ) Query
            .get() Task<QuerySnapshot!>
            .addOnSuccessListener { result ->
                val favoritterliste = mutableListOf<String>()
                val dataliste = mutableListOf<String>()
                for (document in result) {
                    favoritterliste.add(document.data["skilt"].toString())
                    dataliste.add(document.data.toString())
                }
                favoritterSkilt.value = favoritterliste
                allefavoritter.value = dataliste
            }
            .addOnFailureListener { exception ->
                Log.w( tag: "Feil", msg: "Fikk ikke tilgang til dokumenter: ", exception)
            }
    }
}
```

Begge disse ligger i FavoritterViewModel.kt. Funksjonen hentFavoritter() legger alle skilt nr. i en liste, og all data om alle favorittene i en annen liste. Dette gjøres med en gang bruker går inn på “Favoritter”. Funksjonen slettFavoritt() finner skilt nummeret og brukerID på kjøretøyet som skal slettes i databasen.

```

  ↗ Endre Kvant
    fun slettFavoritt(valgtFavoritt: String) {
        val db = Firebase.firebaseio
        db.collection( collectionPath: "favoritter" ) CollectionReference
            .whereEqualTo( field: "brukerID", brukerID ) Query
            .whereEqualTo( field: "skilt", valgtFavoritt )
            .get() Task<QuerySnapshot!>
            .addOnSuccessListener { result ->
                for (document in result) {
                    db.collection( collectionPath: "favoritter" ).document(document.id).delete()
                    favoritterSkilt.value = favoritterSkilt.value.filter { it != valgtFavoritt }
                }
            }
            .addOnFailureListener { exception ->
                Log.w( tag: "Feil", msg: "Fikk ikke tilgang til dokumenter: ", exception)
            }
    }
}

```

## 6.2.1 Dokumentasjon

- Innledning
- Funksjonell beskrivelsene
- Mine funksjoner
- Læring og refleksjon
- Referanser og kilder

## 6.3 Håvard Garsrud

### 6.3.1 Kodebidrag

Har hatt hovedansvar for kamerafunksjonaliteten i applikasjonen, mer spesifikt klassene “Kamera”, “KameraSkjerm”, “TekstGjenkjenner” og “IngenTillatelseSkjerm”. Har også lagt til mulighet til å endre tema og språk i Settings.kt, se kodesnutt under. La også til datavalidering på skiltnummer.

#### 1. Innstillinger

Design og implementasjon av tematoggle (mørkt eller lyst tema), og språktoggle.

#### 2. Kamera

Design og implementasjon av sidene under Kamera, IngenTillatelseSkjerm.kt, Kamera, KameraSkjerm.kt og TekstGjenkjenner. Brukte disse videoene for å få til tekstgjennkjenningen i kamerafunksjonen (Yanneck Reiß, 2023a) og (Yanneck Reiß, 2023b).

### 3. Theme.kt

La inn fargene i appen i konstanter, slik at de kan brukes gjennom hele appen i enten mørkt eller lyst tema.

### 4. Generelt

Hjulpet andre der det trengs, primært med design.

#### 6.3.2 Kodesnutt

Composable-funksjonen endrer språket i applikasjonen, man må ha Android versjon 13 (SDK level 33).

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SpråkToggle(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    val språk = arrayOf(
        "English",
        "Norsk",
        "Français"
    )
    val flagg = listOf(
        R.drawable.united_kingdom,
        R.drawable.norway,
        R.drawable.france
    )
    var utvidet by remember { mutableStateOf(false) }
    var valgtSpråk = ""
    when (Locale.current.language) {
        "en" -> valgtSpråk = språk[0]
        "nb" -> valgtSpråk = språk[1]
        "fr" -> valgtSpråk = språk[2]
    }
    ExposedDropdownMenuItem(
        expanded = utvidet,
        onExpandedChange = {
            utvidet = !utvidet
        },
    ),
}
```

```
        modifier = modifier.padding(horizontal = 10.dp)

    ) {

    CompositionLocalProvider(LocalTextInputService provides null) {
        TextField(
            value = valgtSpråk,
            onValueChange = {},
            readOnly = true,
            colors = TextFieldDefaults.textFieldColors(
                textColor = MaterialTheme.colorScheme.onBackground,
                containerColor = MaterialTheme.colorScheme.tertiary,
                focusedIndicatorColor = MaterialTheme.colorScheme.onBackground
            ),
            trailingIcon = {
                ExposedDropdownMenuDefaults.TrailingIcon(expanded = utvidet) },
            modifier = modifier
                .menuAnchor()
                .fillMaxWidth()
        )
    }
}

ExposedDropdownMenu(
    expanded = utvidet,
    onDismissRequest = { utvidet = false },
    modifier = modifier.background(MaterialTheme.colorScheme.tertiary)
) {
    språk.forEachIndexed { index, item ->
        DropdownMenuItem(
            text = {
                Text(
                    text = item,
                    color = MaterialTheme.colorScheme.onBackground
                )
            },
            leadingIcon = {

```

```
Image(  
    painter = painterResource(flagg[index]),  
    contentDescription = null,  
    modifier = modifier  
        .height(20.dp)  
        .width(30.dp)  
)  
,  
onClick = {  
    valgtSpråk = item  
    when (valgtSpråk) {  
        "English" -> {  
            context.getSystemService(LocaleManager::class.java)  
                .applicationLocales = LocaleList.forLanguageTags("en")  
        }  
  
        "Norsk" -> {  
            context.getSystemService(LocaleManager::class.java)  
                .applicationLocales = LocaleList.forLanguageTags("nb")  
        }  
  
        "Français" -> {  
            context.getSystemService(LocaleManager::class.java)  
                .applicationLocales = LocaleList.forLanguageTags("fr")  
        }  
    }  
}  
}  
}
```

### 6.3.3 Dokumentasjon

- Designvalg: Frontend og Backend
- Utlasseringsdiagram
- Rettsskriving og formatering av rapport
- Læring og refleksjon
- Referanser og kilder

## 6.4 Oscar Skauge Eriksen

### 6.4.1 Kodebidrag

I dette prosjektet har jeg bidratt både individuelt og gjennom felles koding. Jeg har laget/vært en del av klassene:

DataApi/APIViewModel, DataInterface, HentBillInfo, Interceptor, RetrofitInstance, SokerInfo, Sammenlign/Sammenlign og mappen DataModeller.

#### 1. API

Her jobbet jeg sammen med Philip for å få tatt i bruk Retrofit for å gjøre API-kall. Vi har fått noe hjelp fra Daniel (Interceptor) og Endre (å finne frem noen av variablene i data klasse hierarkiet). Dette var en oppgave som tok mye tid, ettersom vi hadde problemer med å få det til å ha sin egen coroutine. I første versjon av API kallet vårt, gjorde vi det uten en egen coroutine, og uten ViewModel. I ettertid gjorde jeg endringer på API kallet, som sørget for at det hadde en ViewModel, og kjørte det også i en LaunchedEffect, slik at API kallet har sin egen coroutine. Jeg lagde også en metode (HentBillInfo) som henter ut variablene vi bruker, og returnerer et objekt med de variablene. Dette var for å gjøre koden på SokerInfo lettere å lese, og slik at den kunne brukes i andre klasser, som sammenlign.

#### 2. Datamodeller

For API responsen, trengte vi en dataklasse som strukturerer JSON-filen. Dette er mappen datamodeller, som er et hierarki av dataklasser. Dette ble laget ved hjelp av et Android Studio plugin som heter JSON to Kotlin Class (JsonToKotlinClass, n.d.). Her limte vi bare inn et eksempel på en response fra Statens vegvesen sitt API, og pluginet lagde dataklassehierarkiet vårt.

#### 3. SokerInfo

Denne siden har jeg hatt hovedansvar for, men det er flere som har gjort småting på siden.

#### 4. Design

Design av SokerInfo, som er siden som viser informasjon om en bil etter et søk.

#### 5. Design

Sammen med Endre, hadde jeg ansvar for design av sammenlign.

## 6. Generelt

Utenom mine ansvarsområder, har jeg hjulpet/prøvd å hjelpe andre med ting de sitter fast på. Jeg har også fått hjelp fra andre med mine problemer.

### 6.4.2 Kodesnutt

Koden under henter URL fra tekstfeltet på startsiden, og kjører LaunchedEffecten, hvor jeg kjører API kallet, med API endpointet som input, via en ViewModel.

```
var url by remember { mutableStateOf("") }
url = "kjoretoydata?kjennemerke=$name"
var bilInfo by remember { mutableStateOf<KjoretoyDataListe?>(null) }
LaunchedEffect(url) {
    try {
        val info = viewModel.hentBilInfo(url)
        bilInfo = info
        Log.e(
            "API info",
            "Info: $info"
        )
    } catch (e: Exception) {
        Log.e(
            "API info",
            "Error: ${e.message}"
        )
    }
}
```

Koden under er en ViewModel for Retrofit API kallet, som brukes i SokerInfo. Denne klassen finnes i DataApi/APIViewModel.

```
class APIViewModel() : ViewModel() {

    private val apiKobling =
        RetrofitInstance.api.create(DataInterface::class.java)
```

```
suspend fun hentBilInfo(regNr:String): KjoretoyDataListe? {  
  
    val response = apiKobling.getKjoretoyDataListe(regNr)  
  
    if (response.isSuccessful) {  
  
        return response.body()  
  
    }  
  
    return null  
}
```

#### 6.4.3 Dokumentasjon

- Introduksjon
- Funksjonell beskrivelse
- Læring og refleksjon
- Brukerveiledning
- Referanser og kilder

## 6.5 Philip Toan Nguyen

### 6.5.1 Kodebidrag

Mitt bidrag til koding har både vært individuelt og felles. Vi har vært gode til å fordele arbeidsoppgaver. Derfor nevner jeg ting i lister som jeg har fått fordelt og bidratt med.

#### 1. API

Jeg har jobbet sammen med Oscar for å få til denne delen. API-delen var den som tok lengst tid og mest utfordrende, men vi tok i bruk Retrofit. For å ikke gjenta dette, så har Oscar forklart prosessen vi har gått igjennom. Men vi fant en løsning og brukte Log.d (samme som system.out.print på Java) for å sjekke om vi fikk en kobling.

**Klasser:** Retrofit Instance, DataInterface, Datamodeller(Package)

#### 1. Navigasjon

Jeg har bidratt med å lage navigasjonsbar som alltid skal være på bunnen av applikasjonen. Det er en offisiell dokumentasjon på hvordan man kan lage det. Siden dette var ganske nytt for meg, så tok det tid å sette meg inn i det, men til slutt skjønte jeg hvordan man kunne løse dette til vår applikasjon, ved å prøve mange mulige måter. Med det offisielle dokumentet for navigasjonsbar (Android Developers, 2023), så har jeg klart å tilpasse dette til applikasjonen.

**Klasser:** Bottom Nav.kt og Nav.kt

#### 2. Startsiden: Startsiden har forandret seg siden første gang vi presenterte applikasjonen. Jeg tok på meg ansvaret for å forandre denne siden. (Vi ble enige i fellesskap på forhånd hvordan det skulle se ut sånn ca.)

**Klasser:** Start (Fjernet SøkReg og implementerte den i Start).

#### 3. Generelt: Når vi var ferdige med vår del av arbeidsoppgaven, så var det å hjelpe andre når de trengte hjelp. Som for eksempel legge til glemte funksjoner eller fullføre design på ulike sider, men dette har alle vært med på.

**Klasser:** Auth.kt, Screen.kt, Registrer, Login

## 6.5.2 Kodesnutt

```
Scaffold(
    modifier = Modifier.fillMaxSize(),
    bottomBar = {
        NavigationBar(
            containerColor = MaterialTheme.colorScheme.secondary,
        ) {
            val navBackStackEntry: NavBackStackEntry? by
                navController.currentBackStackEntryAsState()
            val currentDestination: NavDestination? = navBackStackEntry?.destination
            listeAvNavItems.forEach { bunnNavItem: BunnNavItems ->
                NavigationBarItem(
                    selected = currentDestination?.hierarchy?.any { it.route ==
                        bunnNavItem.route } == true,
                    onClick = {
                        navController.navigate(bunnNavItem.route) {
                            popUpTo(navController.graph.findStartDestination().id) {
                                saveState = true
                            }
                            launchSingleTop = true
                        }
                    },
                    icon = {
                        Icon(
                            imageVector = bunnNavItem.icon,
                            contentDescription = null,
                            modifier = Modifier.size(24.dp),
                            tint = MaterialTheme.colorScheme.onBackground
                        )
                    },
                    label = {
                        Text(
                            text = context.getString(bunnNavItem.labelStringId),
                            color = MaterialTheme.colorScheme.onBackground
                        )
                    }
                )
            }
        }
    }
)
```

### 6.5.3 Dokumentasjon

- Designvalg: Wireframe
- Teknisk systemdokumentasjon: UseCase og Klassediagram
- Læring og refleksjon
- Referanser og kilder

## 6.6 Arbeidsinnsats

Arbeidsinnsatsen på prosjektet mener vi er likt fordelt mellom alle gruppemedlemmene. Alle medlemmene har hatt sine ansvarsområder, slik at arbeidet er fordelt likt mellom alle gruppemedlemmer. Vi har ikke dokumentert timer.

Innsatsen er fordelt slik:

Endre: 20%

Philip: 20%

Daniel: 20%

Oscar: 20%

Håvard: 20%

## 7. Læring og refleksjon

Vårt arbeid med prosjektet har vært en positiv opplevelse for alle gruppemedlemmene. Dette var det første prosjektet til gruppen sammen, og vi har fått til et bra samarbeid. Gruppens arbeidsfordeling har vært bra, hvor alle gruppemedlemmene har hatt sine egne arbeidsområder.

Det endelige produktet vi har laget, er et produkt vi alle er fornøyde med. Applikasjonen gjør det meste som var planlagt. Det var enkelte ting vi håpet på å få implementert i applikasjonen, men som vi ikke fikk tid til. På sammenlign-siden hadde vi en plan om å kunne velge fra favoritter eller søke opp kjøretøy man vil sammenligne. I vårt produkt er det bare mulig å søke, så å kunne velge fra favoritter ble altså ikke implementert.

Kamera-funksjonen vår skulle også bare kunne scanne skiltnummer i en avgrenset boks på midten av skjermen, slik at ikke all tekst på kameraet blir hentet. Dette fikk vi ikke tid til.

Vi har jobbet i en SCRUM-metodikk, hvor vi har hatt møter hver uke. Gruppen har ikke hatt noen store problemer underveis, grunnet de ukentlige møtene våre. Et problem vi møtte på var å få til API koblingen via Retrofit til Statens vegvesen sitt API. Dette ble løst ved hjelp av våre ukentlige møter, hvor gruppen sammen fikk til en løsning.

Som gruppe har vi vært flinke på kommunikasjon, noe som har gjort arbeidsprosessen lett og lærerik.

Generelt så er vi fornøyde med det vi har fått til, men det er par ting som kunne blitt gjort bedre. Det er god kommunikasjon innad i gruppa, men når det kommer til organisering så har vi vært litt for dårlige til å planlegge ordentlige møter. Alt i alt så syns vi at vi har jobbet bra sammen, og har fått levert et produkt som vi kan være stolte av.

## 8. Referanser og kilder

- (n.d.). Figma: The Collaborative Interface Design Tool. Retrieved November 29, 2023, from <https://www.figma.com>
- Ahmed, U. (2022, August 3). *Retrofit Library in Android*. Topcoder. Retrieved November 29, 2023, from <https://www.topcoder.com/thrive/articles/retrofit-library-in-android>
- Android Developers. (n.d.). *Side-effects in Compose | Jetpack Compose*. Android Developers. Retrieved November 30, 2023, from <https://developer.android.com/jetpack/compose/side-effects>
- Android Developers. (2023, November 23). *Navigating with Compose | Jetpack Compose*. Android Developers. Retrieved November 29, 2023, from <https://developer.android.com/jetpack/compose/navigation>
- Contrast Checker. (n.d.). WebAIM. Retrieved November 29, 2023, from <https://webaim.org/resources/contrastchecker/>
- Google Developers. (2023, May 2). *Text recognition v2 | ML Kit*. Google for Developers. Retrieved November 30, 2023, from <https://developers.google.com/ml-kit/vision/text-recognition/v2>
- Hanna, K. T. (2023, May). *What is Google Firebase?* TechTarget. Retrieved November 30, 2023, from <https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase>
- JsonToKotlinClass. (n.d.). *wuseal/JsonToKotlinClass: 🚀 Plugin for Android Studio And IntelliJ Idea to generate Kotlin data class code from JSON text ( Json to Kotlin )*. GitHub. Retrieved November 30, 2023, from <https://github.com/wuseal/JsonToKotlinClass>
- Kozłowski, M. (2023, January 9). *From Material Design to Material You (Material 3)*. Espeo Software. Retrieved November 29, 2023, from <https://espeo.eu/blog/the-evolution-of-material-design/>
- Okan, E. (2022, December 9). *Introduction to Jetpack Compose. Jetpack Compose is a modern UI toolkit... | by Ecem Okan*. Medium. Retrieved November 29, 2023, from <https://medium.com/@ecemokan/introduction-to-jetpack-compose-c042e2544db6>
- Philipp Lackner. (2023, April 5). *Firebase Google Sign-In With Jetpack Compose & Clean Architecture - Android Studio Tutorial*. YouTube. Retrieved November 30, 2023, from <https://www.youtube.com/watch?v=zClfBbm06QM>
- Square. (n.d.). Interceptors - OkHttp. Retrieved November 30, 2023, from <https://square.github.io/okhttp/features/interceptors/>

- Square. (n.d.). *Retrofit*. Square Open Source. Retrieved November 30, 2023, from  
<https://square.github.io/retrofit/>
- Statens vegvesen. (n.d.). *API for kjøretøyopplysninger*. Statens vegvesen. Retrieved November 30, 2023, from  
<https://www.vegvesen.no/kjoretoy/kjop-og-salg/kjoretoyopplysninger/api-er-for-tekniske-kjoretoyopplysninger/api-for-tekniske-kjoretoyopplysninger/>
- Wheeler, P. (2008, November 4). *language agnostic - What is Type-safe?* Stack Overflow. Retrieved November 30, 2023, from  
<https://stackoverflow.com/questions/260626/what-is-type-safe>
- Yanneck Reiß. (2023a, June 25). *How to Effortlessly Integrate CameraX with Jetpack Compose in Android | In-App Camera Tutorial*. YouTube. Retrieved November 30, 2023, from <https://www.youtube.com/watch?v=pPVZambOuG8>
- Yanneck Reiß. (2023b, August 4). *Integrating Google's ML Kit with CameraX and Android Jetpack Compose: Text Recognition in Real-Time*. YouTube. Retrieved November 30, 2023, from <https://www.youtube.com/watch?v=wCADCaeS8-A>