Planet Pluss – regnetreningsapp for barn

Mappeeksamen – innlevering 1 av 3

Emne: DAVE3600 Apputvikling

Dato**:** 28. september 2025

Innholdsfortegnelse

Innledning

Hensikten med regnetreningsappen *Planet Pluss* er å lære bort addisjon til barn. Barn kan trene på oppgaver gjennom å spille spillet som er tilgjengelig fra startmenyen. I spillet får brukeren tilfeldige regnestykker, og svaret legges inn via et talltastatur. Brukeren kan velge mellom 5, 10 eller 15 spørsmål per runde, og ingen spørsmål gjentas i samme runde. Applikasjonen har støtte for både norsk og tysk, avhengig av innstillingene på enheten eller emulatoren.

I denne rapporten dokumenterer jeg oppbygningen av Android-applikasjonen, med fokus på arkitektur, implementasjon og design. Underveis viser jeg til kodeeksempler og skjermbilder for å illustrere valg, begrunnelser og hvordan appen fungerer i praksis.

Arkitektur og teknologi

Applikasjonen er utviklet i Kotlin, som er det språket Google anbefaler for Android-utvikling. Kotlin er mer konsist en Java, og har i tillegg innebygd null-sikkerhet som bidrar til å unngå en av de vanligste feilene i Java, nemlig NullPointerException.

Jetpack Compose blir tatt i bruk som rammeverk for å bygge brukergrensesnittet (UI). Rammeverket gjør det enklere å lage dynamiske og reaktive grensesnitt ved å bruke Composable Functions, markert med @Composable. Composables er modulære byggeklosser som kan settes sammen og gjenbrukes på tvers av skjermbilder. Compose er også deklarativt og state-drevet, noe som betyr at UI automatisk oppdateres når data endres.

Et prinsipp innen app-arkitektur er å skille brukergrensesnitt (UI) og ViewModel. Mens UI-laget bestående av Composable-funksjoner er ansvarlig for skjermbilder og reagerer på brukerinteraksjoner, håndterer ViewModel-laget tilstanden i spillet, logikk og sørger for å bevare data. Dette skillet er viktig i Android, fordi livssyklusen til aktiviteter og composables kan bli avbrutt når som helst. Ved å legge tilstanden i en ViewModel, kan vi sikre at brukeren ikke mister progresjon i spillet, og at appen oppfører seg stabilt. Samtidig gjør denne oppdelingen det enklere å teste logikken separat fra UI, samt å isolere feil (Android, 2024).

UI og Composables

Applikasjonen Pluss Planet sitt brukergrensesnitt består av fire hovedskjermer, som er strukturert i hver sin fil under ui/screens; hjemskjerm (se figur 1), selve spillet, forklaring av spillet og preferanser (se figur 2). En slik separasjon gir en tydelig ansvarsfordeling og bidrar til bedre oversikt når kodebasen vokser. Det gjør vedlikehold enklere, siden endringer på én skjerm kan gjøres uten å påvirke de andre, og det legger til rette for gjenbruk av komponenter på tvers av skjermbilder.

Figur 1: Skjermbilde for hjem med startmeny

Figur 2: Skjermbilde for preferanser

Filene består av flere nested Composables, som lar oss bygge komponenter hierarkisk og kombinere dem på fleksible måter. En typisk struktur er en Column som inneholder tekst, knapper og bilder. I figur 3 vises et utsnitt av en nested Composable-funksjon, som bygger opp skjermbildet for informasjon om spillet (figur 4).

```
@Composable
fun AboutScreen(navController: NavController, modifier: Modifier = Modifier) {
    Column(
        modifier = modifier
            .fillMaxSize()
            .systemBarsPadding()
            .background( color = DarkBlueBackground)
            .padding( all = 16.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.CenterHorizontally

    ){
        Spacer(modifier = Modifier.height( height = 10.dp))
        Image(
            painter = painterResource(id = R.drawable.mappe1_logo),
            contentDescription = "App Logo",
            modifier = Modifier.size( size = 200.dp),
            contentScale = ContentScale.Fit
        )
        Spacer(modifier = Modifier.height( height = 20.dp))
        Text(text = stringResource( id = R.string.about_game_description),
            fontSize = 30.sp,
            lineHeight = 38.sp,
            textAlign = TextAlign.Center,
            color = White
        )
        Spacer(modifier = Modifier.height( height = 30.dp))
        Button(onClick = { navController.navigateUp() },
            modifier = Modifier
                .fillMaxWidth(),
            colors = ButtonDefaults.buttonColors(
                containerColor = BrightPink,
                contentColor = LightYellow
            )
            ) {
            Text(text = stringResource( id = R.string.back_button),
                fontSize = 40.sp
            )
        }
    }
```

Figur 3: Eksempel på nested Composables

Figur 4: Skjermbilde for informasjon om spillet

For å illustrere hvordan Composables fungerer i praksis, kan vi se på hva som skjer når brukeren interagerer med selve spillet. Når brukeren sender inn et svar via tekstfeltet, så skjer det to ting med skjermbildet. Spørsmålet oppdateres i bakgrunn gjennom en endring i en enkel tekst-Composable. I tillegg dukker det opp en dialogboks, og dette er fordi en ny Composable-funksjon legges oppå de eksisterende elementene. Se figur 5 og 6 nedenfor.

Dette eksempelet viser hvordan Composables gjør det mulig å oppdatere kun de delene av skjermen som faktisk endres, uten å måtte gjenskape hele skjermbildet. Resten av UI-komponentene beholdes uendret. Composables kan legges oppå hverandre i en “stack”, og fjernes enkelt når de ikke lenger trengs. Dette gjør grensesnittet både enklere å vedlikeholde og mer ressursvennlig.

Figur 5: Skjermbilde fra spillet

Figur 6: Spillet etter bruker trykker på tallet 5 og "Svar!"

Dialogbokser brukes for å vise tilleggsinformasjon eller når brukeren må ta et valg før de kan fortsette. Dersom brukeren forsøker å gå tilbake midt i spillet, blir de spurt om de virkelig ønsker å avslutte. Ved å forhindre utilsiktede handlinger kan brukeropplevelsen forbedres.

I prosjektet har jeg to filer med dialogkomponenter under ui/components: én som håndterer tilbakemelding underveis og sluttscore (FeedbackDialog), og én som håndterer avslutning midt i spillet (ExitGameDialog). Det å separere koden for dialog ut i egne filer gir en tydelig ansvarsfordeling, gjør koden mer oversiktlig, muliggjør gjenbruk og forenkler vedlikehold.

Brukerinteraksjon i GameScreen styrer når og hvilken dialogboks som vises. Dersom brukeren for eksempel klikker på tilbakeknappen, illustrert som et konevensjonelt ikon øverst til venstre i figur 5, så trigges ExitGameDialog. I denne funksjonen sjekkes det om bruker trykker "Nei" eller utenfor boksen, som avbryter operasjonen, eller om bruker bekrefter med å trykke på "Ja". GameScreen mottar svaret og navigerer brukeren til hjemskjerm dersom avslutning bekreftes.

State og ViewModel

ViewModel brukes blant annet til å lagre data som skal bevares selv om en aktivitet blir ødelagt og gjenoppbygd av Android-rammeverket. Slike livssyklusendringer kan oppstå for eksempel ved skjermrotasjon eller ustabil nettverksforbindelse. Man tar da vare på data i UI state. UI er det brukeren ser, mens UI state er det appen sier at brukeren skal se. Dersom det skjer endringer i UI state, reflekterer disse seg umiddelbart i UI (Android, 2023).

I dette prosjektet håndteres spillets tilstand i filen GameViewModel, hvor informasjon som spørsmål, poengsum, antall besvarte spørsmål og det siste svaret lagres. GameScreen mottar UI state via collectAsState(), noe som gjør at grensesnittet oppdateres automatisk når spillets tilstand endres. Dette sikrer at brukeren beholder progresjonen sin, og at forrige svar kan behandles korrekt i dialogboksene selv om et nytt spørsmål vises på skjermen.

UI state går imidlertid tapt dersom applikasjonen startes helt på nytt. For data som skal overleve både livssyklusendringer og app-omstarter, brukes SharedPreferences. Dette er en nøkkel–verdi-lagring hvor innstillinger lagres med en nøkkel, i dette tilfellet "num_questions", som kan ha verdiene 5, 10 eller 15. Når brukeren endrer innstillingen, lagres verdien under nøkkelen. Når appen startes på nytt, hentes verdien igjen. På denne måten bevares preferansene. Standardverdien settes til 5 dersom ingen innstilling finnes fra før. I figur 7 vises funksjonene som lagrer og henter antall spørsmål ved hjelp av SharedPreferences.

```
fun saveNumQuestions(value: Int) {
    val sharedPreferences = getApplication<Application>()
        .getSharedPreferences(prefsName,  mode = Application.MODE_PRIVATE)
    sharedPreferences.edit {
        putInt(keyNumQuestions, value)
    }
}
2 Usages
fun getNumQuestions(): Int {
    val sharedPreferences = getApplication<Application>()
        .getSharedPreferences(prefsName,  mode = Application.MODE_PRIVATE)
    return sharedPreferences.getInt( key = keyNumQuestions,  defValue = 5)
}
```

Figur 7: Viser hvordan brukerens valg av antall spørsmål lagres og hentes, selv om appen lukkes eller roteres

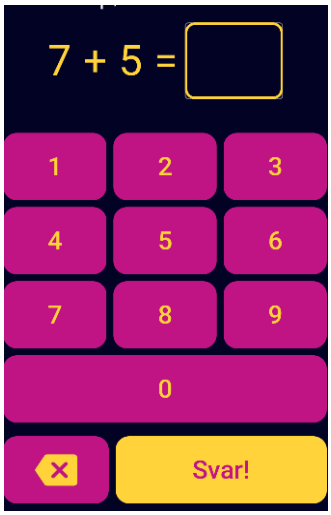
Navigasjon

Når et Android-prosjekt starter, kjøres onCreate-funksjonen i MainActivity. Her opprettes en NavController som styrer navigasjonen, og denne kobles til en navigation graph. I dette prosjektet ligger navigation graph i AppNavHost, som er en composable-funksjon hvor alle ruter og skjermer defineres. Start-destinasjonen er satt til home, slik at hjemskjermen vises først når appen åpnes. Hver composable i AppNavHost representerer en rute og knyttes til den skjermen som skal vises når brukeren navigerer dit (Android, 2025). Ved å samle rutene i en navigation graph får man enkel oversikt over appens flyt. Det gir også en modulær struktur, som gjør det lettere å legge til nye skjermer i ettertid uten å endre logikken i selve aktivitetene.

Design

Applikasjonen tar i bruk sterke kontraster og store fonter for god lesbarhet. Store knapper og lite tekst er spesielt viktig ettersom små barn er målgruppen. Appikonets oransje katt og gulfargen til applikasjonsnavnet *Planet Pluss* står i komplementær kontrast mot den mørkeblå bakgrunnen, noe som gjør elementene ekstra synlige (se figur 8). Fargevalg er gjort med tanke på tilgjengelighet, slik at tekst og viktige elementer er tydelige for brukere med nedsatt fargesyn (Android, 2023).

Figur 8: Appikonet som går igjen i appen



Figur 9: Et tydelig talltastatur hvor gulfargen retter fokus mot oppgaven

Siden målgruppen ikke nødvendigvis kan lese, har jeg lagt vekt på å skape forståelse gjennom det visuelle. Spillskjermen har derfor et markert inputfelt som viser hvor svaret skal

fylles inn. Svarknappen i samme gulfarge som oppgaven og inputfeltet, knytter elementene visuelt sammen, og hjelper brukeren å forstå at svaret skal bekreftes der. Den gule fargen skaper samtidig blikkfang, og retter oppmerksomheten til det som skal utføres (se figur 9).

For å skape et ryddig og oversiktlig layout har jeg lagt til luft mellom elementer, brukt gjenbrukbare størrelser og konsekvent fargebruk. Dette følger prinsipper fra Material Design, og gir et mer enhetlig og navigerbart grensesnitt (Android, 2024).

Da jeg valgte appikon tenkte jeg på hva som appellerer til barn. Jeg hentet inspirasjon fra Pinterest, og lot meg inspirere av søte katter og verdensrom-tema. Logoen ble skissert utfra en kombinasjon av egne ideer og inspirasjonskilder, og deretter tegnet i Clip Studio Paint. Planetringene er formet som et plusstegn, som refererer til addisjon og passer til appens verdensrom-tema.

Avgrensninger

På grunn av plassbegrensninger er det ikke mulig å inkludere alle kodeeksempler fra prosjektet. Spesielt GameViewModel og GameScreen inneholder mye av applikasjonens kjernefunksjonalitet, som spørsmålsgenerering, poengoppdatering, state-håndtering og kobling mellom UI og spilllogikk. Kommentarene i kodefilene dekker flere detaljer som ikke er vist her. Mørkt og lyst tema via Theme.kt er ikke implementert, og farger er definert direkte i Color.kt.

Konklusjon

Pluss Planet er utviklet som et verktøy for å lære barn addisjon. I rapporten har jeg fokusert på separasjonen av UI og ViewModel, hvordan Composables er strukturert etter ansvarsområder, og hvordan UI state og SharedPreferences brukes for å bevare data på ulike nivåer. Med kodeeksempler og skjermbilder forsøker jeg å begrunne valgene som er tatt.

Applikasjonen følger oppgavens krav når det gjelder spilllogikk, selv om det ikke har vært mulig å gå i detalj på selve koden for logikken i denne rapporten.Designet er tilpasset barn, med universell utforming, pedagogiske og positive tilbakemeldinger, samt et barnevennlig visuelt uttrykk.

Mulige videreutviklinge kan være implementering av lyst og mørkt tema, flere grafiske elementer som stjerner i bakgrunnen for å komplementere tema, bruk av forskjellige fonter for større kontrast mellom titler og brødtekst, samt utvidelse med flere spørsmål.

Litteraturliste

Android (2023, 8. mai). *Accessibility*. https://developer.android.com/design/ui/mobile/guides/foundations/accessibility

Android. (2023, 21. september). *4. Learn about app architecture*. https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state#3

Android. (2024, 18. januar). *Jetpack Compose basics*. https://developer.android.com/codelabs/jetpack-compose-basics#0

Android (2024, 18. oktober). *Material Design for Android.* https://developer.android.com/develop/ui/views/theming/look-and-feel

Android (2025, 24. september). *Navigation*. https://developer.android.com/guide/navigation