IDATT2506 Applikasjonsutvikling for mobile enheter Arbeidskrav 08 - Kryssplattformprosjekt

John-Ivar Eriksen *jierikse@stud.ntnu.no*
Norwegian University of Science and Technology,
Department of Computer Science,
NO-7491 Trondheim, Norway

1 Introduksjon

Denne rapporten sammenligner noen kryssplattformteknologier for applikasjonsutvikling, spesielt med hensyn på mobile applikasjoner.

For oppsett og hvordan kjøre applikasjonen, se `README.md` som ligger sammen med kildekoden.

1.1 Oppgave og problemstilling

Oppgave: Utvikle en app for handle- og gjøremålslister.

> Oppgaven går ut på å lage en enkel app for handle/todoliste. "Enkel" i det henseendet at det skal være raskt å føre opp nye innslag, slik at appen gjør jobben sin fortest mulig og lar deg fortsette hva enn du holdt på med.

Problemstilling i utvikling: Hvilke teknologier velger man, og av hvilken årsak.

> Faget er laget for ingeniørstudenter og da skal vi forvente at man kan gjøre kloke valg basert på fakta. Man skal ikke velge teknologi fordi man vil eller "kan den", men fordi det er den beste for formålet.

2 Interaksjonsdesign

For en kort gjennomgang av appens grafiske funksjoner, samt begrunnelser og forklaring av designvalg, se rapportens vedlegg "A08-01 Interaksjonsdesign".

3 Kryssplattformløsninger

3.1 Tekniske spesifikasjoner

- IDE: Android Studio
- Programmeringsspråk: Dart
- GUI-rammeverk: Flutter
- Android API 35 (Android 15, "Vanilla Ice Cream")

Appen er testet på en emulert enhet av overnevnt API, i Android Studio.

3.2 Sammenligning av teknologier

Oppgaven presenterer et knippe kjente kryssplattformløsningene, som alle har sine fordeler og ulemper, i tillegg til å oppfordre studenten til å undersøke og sammenligne teknologier de selv synes er interessante.

Av de som sammenlignes her er React Native, sammen med Flutter+Dart, de mest utbredte. Kirigami er lite brukt utenfor utvikling til Linux, da gjerne spesifikt til KDE-miljøet. C# med Avalonia UI har en økende brukermasse, men er fortsatt ikke i nærheten av populariteten til React og Flutter+Dart på kryssplattform. Kotlin Multiplatform (KMP) med Compose Multiplatform UI er under utvikling av JetBrains og kan raskt bli en sterk kandidat, særlig for utviklere som fokuserer på Android.

3.2.1 C# med Avalonia UI

C# (C-sharp) er et moderne og effektivt programmeringsspråk med tett integrasjon mot Microsofts .NET-økosystem. Det er mye brukt til utvikling på desktop, og har et omfattende utvalg av biblioteker og verktøy, i tillegg til å være godt dokumentert.

Avalonia UI er et open source rammeverk for grafiske brukergrensensitt (GUI), designet for kryssplattform. Det er inspirert av Windows Presentation Foundation (WPF) og Xamarin, og bruker den velkjente Model-View-ViewModel-arkitekturen. Dette gjør Avalonia UI godt egnet for desktoputvikling, spesielt i .NET-økosystemet. På mobilutvikling derimot, fremstår imidlertid Avalonia som mindre modent.

Fordeler

- C# er kraftig, lett å lese og godt dokumentert, med mange verktøy og biblioteker.
- Open source, under aktiv utvikling og et voksende fellesskap gir fleksibilitet.
- Ressurseffektivt og tilbyr god ytelse på desktop med relativt lav ressursbruk.
- MVVM-arkitektur gir en strukturert og testbar måte å utvikle GUI på.

Ulemper (mest tilknyttet Avalonia UI):

- Mobilstøtten er umoden og kan være ustabil.
- Mangler dokumentasjonen og stabiliteten som finnes hos f.eks Flutter og React Native.
- Ennå under utvikling, og endringer kan føre til at koden ikke fungerer ved oppdatering.

3.2.2 Kotlin Multiplatform (KMP) med Compose Multiplatform UI (CMPUI

KMP gjør det mulig å dele kode mellom Android, iOS, web og desktop. Sammen med Compose Multiplatform UI (CMPUI) har det en deklarativ tilnærming til GUI-utvikling, kjent fra Jetpack Compose i moderne Android-utvikling. KMP er spesielt egnet for utviklere som allerede har erfaring med Kotlin, og kan gi en nær-native opplevelse, særlig på Android. Offisiell støtte fra Google[1] og JetBrains gjør rammeverket til et lovende og fremtidsrettet valg. Ettersom KMP fortsatt er et relativt nytt rammeverk, er fellesskapet mindre sammenlignet med konkurrenter som Flutter. Dette kan gjøre det vanskeligere for nye utviklere å finne ressurser, tutorials og tredjepartsbiblioteker.

Fordeler

- Moderne, deklarativ, UI-utvikling med gjenbrukbare komponenter.
- Native ytelse.

[1] "(...) and today at Google I/O we are excited to announce we are supporting Kotlin Multiplatform on Android, (...)" (14 May 2024) URL: `https://android-developers.googleblog.com/2024/05/android-support-for-kotlin-multiplatform-to-share-business-logic-across-mobile-web-server-desktop.html`

- Kan dele business logic mellom plattformer, samtidig som UI kan være spesifikt for hver plattform.
- Lavere læringskurve for Android-utviklere.

Ulemper

- Støtte for iOS er ikke like godt uviklet som for Android enda.
- Kompleksitet i oppsett er utfordrende for nybegynnere og mindre prosjekter.
- Begrenset dokumentasjon, sammenlignet med Flutter eller React Native.

3.2.3 Flutter + Dart

Flutter er et rammeverk utviklet av Google for bruk til kryssplatformapplikasjoner for Android, web, desktop og iOS. Flutter bruker programmeringsspråket Dart (også utviklet av Google), og skal ha fokus på god ytelse. Det har støtte for "hot reload" av kode og funksjonalitet, noe det er kjent for, hvilket gjør at det kan testes fortløpende på en kjørende emulator. Flutter har en deklarativ tilnærming til UI, likt som Jetpack Compose. Alt av GUI skal da håndteres av Flutter, uten avhengigheter til plattformens egne komponenter, hvilket gjør at Flutter skal fungere konsistent på alle plattformene en app kjøres på. Apper kompileres til flere plattformer basert på én kodebase, altså en slipper å skrive appen flere ganger eller gjøre større individuelle tilpasninger til plattformene.

Fordeler

- Høy ytelse i forhold til konkurrentene.
- Hot reload.
- Omfattende bibliotek med ferdige UI-komponenter.
- Bruker samme kodebase for alle plattformer
- Designet for å være lett å lære, spesielt for de med erfaring i objektorientert programmering.

Ulemper

- Dart kan være ukjent for mange utviklere som kommer fra Java eller Kotlin.
- Størrelsen på appene kan bli stor, sammenlignet med native apper pga. Flutter-bibliotek og ressursfiler.
- Mindre "native" følelse grunnet at Flutter og Dart bruker sine egne ressurser.
- Feil eller inkompatibilitet i dependencies kan føre til frustrerende feil.

3.2.4 React Native

Utivklet av Facebook (Meta), for å bygge mobilapplikasjoner ved bruk av JavaScript eller TypeScript. Det er basert på React, som er et populært bibliotek for webutvikling, og er deklarativ og komponentbasert. RN oversetter koden til native komponenter, hvilket gir apper et mer "autentisk" preg på hver plattform. Dette gjør også at man kan skrive plattformspesifikk kode om nødvendig, hvilket kan være en fordel om det jobbes med komplekse apper. React Native utmerker seg ved å være mer tilgjengelig for utviklere med erfaring fra webutvikling, men lavere ytelse og kompleksiteten i integrasjon med native kode gjør det mindre egnet for hvis ytelse og konsistens er prioritert.

Fordeler

- Bygget på React og JavaScript gjør at det er velkjent for mange webutviklere.

- Bruker native komponenter, som gir et mer OS-autentisk uttrykk.
- Hot reload.

Ulemper

- Generelt lav ytelse, sammenlignet med Flutter og native apper.
- Integrasjon mellom native kode og tredjepartsbiblioteker kan føre til høy kompleksitet i koden, og kan være utfordrende å holde styr på.
- Feilsøking i JavaScript, i tillegg til JS-interaksjon med native kode, kan være svært krevende, siden JS ofte gir lite informative feilmeldinger.
- Appene blir ofte store pga. avhengighet til React Native-biblioteket.
- Ikke fullt kryssplattform, siden det ofte må gjøres tilpassninger til den enkelte plattformen.

3.2.5 Kirigami

Utviklet av KDE for apper som skal fungere sømløst både på desktop og mobil. Det er bygget på Qt, og bruker QML (Qt Modeling Language) for å bygge brukergrensesnittet. Kirigami er først og fremst rettet mot Linux, og da spesielt brukt i KDE-miljøet. Dette gjør det til et veldig godt valg om en ønsker god integrasjon mot Linux og KDE Plasma. Dessverre er mobilstøtten ikke like godt utviklet som hos andre, mer utbredte rammeverk som Flutter og React Native. Kirigami, som Linux, er i stor grad for spesielt interesserte, selv om det er både stabilt, godt dokumentert og har veldig god ytelse. Kirigami har ikke noen "store" aktører i ryggen, slik som Flutter og React Native, og har dermed heller ikke samme gjennomslagskraft eller ressursene til å ha like hurtig fremgang på utviklingen for ulike plattformer. Selv om Kirigami er godt egnet for Linux-applikasjoner, er det lite hensiktsmessig for dette prosjektet som fokuserer på Android.

Fordeler

- Stort og veletablert rammeverk innen Linux og KDE-miljøet.
- Velegnet for utvikling på Linux og KDE Plasma.
- Fleksiblet, med gode muligheter for å tilpasses.

Ulemper

- Begrenset støtte for mobil, sammenlignet med konkurrenter som Flutter og React Native.
- Begrenset community, siden det er mindre utbredt. Det har dermed færre ressurser enn de mer populære løsningene.
- Qt og QML kan være komplekst for nybegynnere å sette opp.
- Begrenset utvalg av tredjepartsbiblioteker, sammenlignet med Flutter eller React Native.
- Ikke mainstream, og lite brukt utenfor KDE og Linux-miljøet. Dette alene gjør det til et lite attraktivt for de fleste, som ofte heller vil bruke de mer populære og store navnene.

4 Resultat

Flutter tilbyr en god balansen mellom ytelse, brukervennlighet og dokumentasjon, noe som gjør det velegnet for denne oppgaven. Sammenlignet med React Native er Flutter bedre egnet for ytelseskrevende applikasjoner, og det gir mer konsistens på tvers av plattformer. Til tross for at Avalonia UI og Kirigami har interessante egenskaper, gjør den begrensede mobilstøtten dem mindre aktuelle for dette prosjektet.