Alexander Shipunov

Innlevering 2 “Et kall!”

Beskrivekse

I min løsning har jeg prøvd å implementere alle tekniske krav som er nevnt i oppgaveteksten. Løsningen består av 5 activities og 1 dialog (fungerer som en slags HOW TO for applikasjonen). Alle activities unntatt Maps Activity har menu for navigering slik at man kan lett åpne nødvendig side.

Når man åpner Main Activity gjennomføres det skriving til/oppdatering av databasen med data om aktuelle lokasjoner hvor man kan finne NFC tags og ID koder som trenges for å fange pikachu.

I Maps Activity plasseres det markører i samsvar med data fra databasen. Lokasjonene hvor pikachu ble ikke fanget markeres med blått. Røde markører viser lokasjoner hvor pikachu ble fanget.

I Catch Activity får bruker to muligheter:

1. man kan skrive koden fra tag og trykke på “CHECK!” knappen
2. man kan skanne NFC tag, da trigges det automatisk sjekk av info fra tagen

Som resultatet får man enten beskjed at pikachu ble fanget (hvis det skjer første gang), ble allerede fanget før, at ID var feil, autentiserings info er fail eller at serveren kan ikke finne pikachu (hvis man sener tom string). Alle disse resultatene får man som svar på http request. Hvis man har fanget pikachu første gang åpnes det Result Activity hvor man kan se alle pikachu som ble fanget i form av ListView med tilsvarende bilder og pikachus navn.

Man kan også se hvilke scores har forskjellige brukere ved å åpne Scores Activity.

Alt nedlasting av informasjon fra internett skjer i separate tråder med sikkerhets mekanismer som forhindrer krasj av applikasjonen hvis man har ikke nett tilkobling.

Forbedrings potensial

Oppdatering av database skjer nå hver gang man åpner MainActivity. Dette er ikke nødvendig og kan implementeres på en mer smart måte slik at det skjer om et jevnt tidsintervall. Selve oppdaterings algoritme kan bli forbedret, slik at applikasjonen sjekker både om det dukket opp nye lokasjoner og om de gamle ikke ble endret (nå sjekkes det bare om det ikke dukker opp nye).

For å øke bruker vennlighet kan det også forbedres flytt mellom lodd- og vannrettet stilling. Nå forsvinner det brukerens input i Catch Activity om man endrer stillingen av telefonen mens man skriver inn kode.

ListView med bilder som vises i Result Activity kan også bli endret. Nå lastes det ned bilder fra nettet og når man scroller ned/opp i listen forsvinner det de punktene som ikke er synlige. Dette fører til at når man åpner (scroller til) de punktene, må applikasjonen laste bildene på nytt (noe som tar tid og ressurser).

Teori

1. For å koble seg til SQL databasen på en mobil enhet trenges det ikke noen tillatelser fra bruker sin side. Man kan gjøre det ved bruk av SQLiteOpenHelper som man kan utvide ved å overskrive metoder onCreate og onUpgrade. I disse metodene kjøres det SQL spørringer som oppretter/endrer tabeller i database. Videre kan man lage en klasse som gjenspeiler data som finnes i tabellen, hvor felter svarer til kolonner. Man lager også en datasource klasse som "kobler" klassen med respektive tabellen. Ved hjelp av datasource får man skrive/lese fra den respektive tabellen i databasen.
2. AsynkTask er en klasse som har en ThreadFactory som i sin tur kaller på new Thread når man instansierer tasken. Man bruker det får å kjøre ressurs/tidskrevende prosesser for å forhindre ANR feil og ikke overbelaste

hovedtråden som er ansvarlig for UI. En av eksemplene på det er kobling til nett som kan være treig eller ressurskrevende (når man laster ned bilder for eks). De tre vanlige metodene som man kan overskrive i AsynkTask er:

2.1. onPreExecute hvor man gjør noe i hoved tråden før man starter eksekvering av kode i den nye tråden. Her kan man for eksempel instansiere progress bar o.l.

2.2. doInBackground hvor man gjennomfører tung oppgave og returnerer resultatet.

2.3. onPostExecute hvor man gjør noe i hoved tråden med resultatet som man fikk fra den forrige metoden.