Utvecklingsmanual

Kodstruktur
Koden för appen är uppdelat i fem paket: core, achievements, profile, network och utils. Klassen MainActivity i paketet core är en knutpunkt som kopplar ihop paketen. Till exempel anropar MainActivity klassen NetworkHandler i paketet network, vilken sedan anropar MainActivity när den är färdig med nätverksuppgiften. Från början hade vi tänkt använda designmönstret Observer/Observable till detta. I paketet achievements används fortfarande observermönstret men på alla andra ställen där vi tänkt ha det använder vi oss nu istället av direkta anrop mellan klasserna.

De flesta Activities & Fragments innehåller både kod för grafik och logik. Uppdelningen av vårt projekt följer alltså inte MVC-mönstret. Ett stort undantag är dock utmärkelsesystemet, där grafiken hanteras i paketet core och logiken i paketet achievements.

- Core-paketet håller i det mesta av grafiken, men också en del logik. Det hanterar också överföringen av information mellan alla paketen. Det visar achievements, hanterar QR-scanning, sköter betygssystemet och en del annat.
- Achievements-paketet hanterar all logik kring skapandet, användandet och upplåsningen av utmärkelser. Även användarens framsteg och poäng hanteras här. Det innehåller inga grafiska komponenter.
- Profile-paketet hanterar dels matchningen av två profiler, samt skapandet av användarens egna profil. Det innehåller både logik och grafiska komponenter.
- Network-paketet har två huvudsakliga uppgifter. Den första är att utföra alla dataöverföringar, både data mellan telefoner samt mellan telefon och server (dessa beskrivs i detalj under rubriken Nätverksprotokoll nedan). Den andra är att lyssna efter och reagera på förändringar i wifi-anslutningar.
- Utils-paketet innehåller verktyg för att läsa och skriva filer, läsa från ElectriCitys och Icomeras APIer, samt kryptera text. Dessa verktyg används av flera av de andra paketen. Utils innehåller bara logik, inga grafiska komponenter.

För mer ingående information om paketen och deras klasser samt kopplingar, se UML och Javadoc.

UML / Javadoc
Se separata dokument.

Nätverksprotokoll
När man skannat en QR-kod (som innehåller enhetens IP-adress), kan den som skannat (A) ansluta till den skannade (B). A öppnar direkt upp en nätverkssocket och ansluter till B, som konstant lyssnar efter inkommande anslutningar. A skickar sedan sin data och tar därefter emot motsvarande data från B, varefter anslutningen bryts. Datan som skickas är en lista bestående av användarens profil samt användarkoden.

När användaren ger betyg (tumme upp/ner) skickas information till servern. Telefonen öppnar en nätverkssocket och skickar data som talar om vem som blivit betygsatt och vilket betyg

denne fick. Servern lagrar datan, som sedan efterfrågas av den betygsattes telefon vid ett senare tillfälle.

I nuläget skickas alltså Javaobjekt i både peer-to-peer-protokollet och client/server-protokollet. Om vi i framtiden utvecklar appen för iOS eller Windows Phone behöver vi skicka datan på annat sätt.

Designval
Vi valde att använda Android API 15, till stor del för att det var den rekommenderade nivån från Android Studio. På nivå 15 når man enligt Android Studio ca 94% av användarna, vilket vi tyckte lät tillräckligt. I efterhand har vi insett att det nog hade varit värt att gå upp några nivåer, t.ex. till 22, där såkallade floating action buttons tillkom, som vi gärna hade använt. Dessutom bör användarprocenten av de högre APIerna stiga under utvecklingstiden, så att när vi väl är klara kan stora delar av Androidanvändarna ändå köra appen.

I efterhand har vi insett att vi borde ha gjort en lite annorlunda paketuppdelning. Paketet core är just nu väldigt stort och har många olika ansvarsområden och det finns dessutom enskilda klasser som har mycket ansvar. Man bör kanske göra en större uppdelning mellan de klasser som sköter grafik och de som sköter logik.

Tester
Det finns ett test-paket som innehåller några testklasser baserade på JUnit. Dessa testar funktionalitet i achievement-paketet, profile-paketet, samt klassen APIHandler. Uppenbarligen finns mycket annat att testa, och vi borde ha skrivit testklasser för resten av funktionerna i appen, men till stora delar är det svårt eller till och med omöjligt att testa utan att utföra såkallade live-tester. Se Javadoc för mer detaljer.