**Design rationale**

Vi har valt att utveckla vår app i Android Studio och mot android-enheter. Det har vi valt att göra dels på grund av våra tidigare kunskaper inom Java, men också på grund av att Android-plattformen är öppnare och tillåter uppdateringar snabbare[[1]](#footnote-1). Det finns också fler billigare varianter av android-telefoner vilket ökar chansen att det är det som kan användas om företagen väljer att köpa in tjänstetelefoner i framtiden. Vi har valt API-nivå 23 som “targeted”, eftersom det är rekommenderad praxis enligt Androids senaste uppdatering. Vi har även valt att ha täcka upp en viss bakåtkompatibilitet genom att använda API-nivå 15 som ”minimum”. Genom detta har vi kunnat täcka in nästan 98% av alla Android-användare[[2]](#footnote-2) och vi tror att om applikationen skulle implementeras hos bussbolaget kommer äldre versioner än så inte att användas.

Vi har försökt minimera externa beroenden för att minimera risker och hållit oss till de standardbibliotek som java och android tillhandahåller. De är väldigt heltäckande och har inte hämmat oss vid designval eller funktionaliteter. Det enda externa beroendet i övrigt är Electricity-plattformen, vilken har tyvärr inte har fungerat, men det hade inte varit möjligt att få tillgång till önskad data på annat rimligt sätt.

Gällande val av databasstruktur valde vi att använda sqlite som är den inbyggda standarden i android. Detta gjordes främst på grund av säkerhet i kompabilitet, enkelheten och minimerad tid som krävdes för implementation. Ett alternativ som vi diskuterade var att sätta upp en egen server och en online-databas, men valde bort det för vi kände att det inte tillförde något värde för vår demo-app. Om den skulle utvecklas och tas i bruk, hade en databas online varit ett naturligt nästa steg. Vi ville dock inte riskera att demonstrationen skulle kunna misslyckas om uppkopplingen till servern inte fungerade just då och valde därför att använda en lokal databas i telefonen.

Överlag har vi försökt följa praxis och de vedertagna designmönster som vi har lärt oss från tidigare kurser och litteratur. Vi har också i den mån det går försökt hålla nere komplexiteten, för att både förenkla för framtida påbyggnad av funktionalitet, men även för att minska risker för buggar. Detta var något som nämndes på föreläsningen med Volvo Cars, vilket vi också därför försökte ta till oss. Dessutom är det en fördel med låg komplexitet om projektet ska lämnas över till en annan part efter dess slut. En låg komplexitet i koden har också underlättat testning under projektets gång.

Vi valde att bygga två separata applikationer, en för busschaufförerna (sändaren) och en för trafikledning/service (mottagaren). Detta gjorde vi för att vi ville täcka in hela felrapporteringsprocessen och inte bara den första aspekten. Mottagarsidan hade kunnat utvecklas i annan form, exempelvis för en dator, men vi ansåg att hög mobilitet kan skapa nya förutsättningar att jobba på ett annat sätt än man idag gör i trafikledning och service. Att vi separerade mottagarappen från sändarappen är för att varje applikation inte ska ha mer funktionalitet än de behöver, för att underlätta för användaren. Dessutom har trafikledning och busschaufförer olika behörigheter och kunskap, vilket kräver olika hantering av vissa delar.

1. http://savvyapps.com/blog/android-vs-ios-which-platform-to-build-for-first [↑](#footnote-ref-1)
2. https://developer.android.com/about/dashboards/index.html [↑](#footnote-ref-2)