Byggprocess

Se till att miljövariabeln ‘ANDROID_HOME’ är satt till sökvägen där Android SDK ligger på din dator.

Öppna terminalen och gå till projektmappen

På Windows kör kommandot ‘gradlew.bat assembleDebug’

På Linux/Mac kör kommandot ‘chmod +x gradlew’ därefter kommandot ‘./gradlew assembleDebug’

APK filen ligger i [ProjektMappen]/app/build/outputs/apk

För att köra tester med robolectric kan denna tutorial med fördel följas:
http://blog.blundell-apps.com/how-to-run-robolectric-junit-tests-in-android-studio/

Standard sms app

För att en app ska kunna väljas som standard sms app i Android måste den uppfylla följande kriterier.

1. Hantera SMS_DELIVER_ACTION för att kunna ta emot SMS
2. Hantera WAP_PUSH_DELIVER_ACTION för att kunna ta emot MMS
3. Hantera ACTION_SENDTO för att andra appar ska kunna skicka SMS via våran app
4. Hantera ACTION_RESPONSE_VIA_MESSAGE för att användaren ska kunna svara ett inkommande samtal med ett textmeddelande

För att hantera de två första åtgärderna är 2 lyssnare implementerade SmsReceiver respektive MmsReceiver. Just nu är inte MmsReceiver implementerad men i grunden ska den funka på samma sätt som SmsReceiver. När någon av dessa får ett inkommande meddelande ser de till att användaren får en notifikation om att ett nytt sms/mms har blivit levererat samt ser till att det sparas ner i telefonens minne. Den tredje åtgärden sköts av den aktivitet som hanterar att skicka sms i appen kallad SendMessage. Den använder modellen för att dela upp meddelandet i flera delar om det är för långt (Ett sms får vara max 160 tecken) och skickar sedan iväg det. Den fjärde åtgärden är ett specialfall som hanteras av en tjänst (service i Android) som heter HeadlessSmsSendService den lyssnar efter åtgärden och ser till att meddelandet blir skickat.

KatlaSpeechToText

KatlaSpeechToText är ett paket som fungerar som en service till activities som vill kunna använda sig av diktering. Paketet innehåller två interface, en factory, en parameterklass och en implementation av ett av interfacen. KatlaRecognitionListener interfacet är vad en activity som vill lyssna måste implementera för att få meddelanden om när KatlaSpeechToText har uppfattat ord osv. KatlaSpeechToText använder sig av android SpeechRecognizer för att sköta den faktiska tal till text funktionaliteten. Anledningen för att

servicen finns är för att på ett smidigt sätt möjliggöra ett utbyte av androids SpeechRecognizer mot ett annat alternativ.

KatlaTextToSpeech

KatlaTextToSpeech är ett paket som fungerar som en service till activities som vill kunna läsa upp en text. Det innehåller ett interface, en implementation av det interfacet, en factory och en klass med parameterar. Servicen använder sig av androids TextToSpeech för den faktiska uppläsningen. Anledningen till att servicen finns är för att tillåta ett enkelt byte till en annan implementation av text to speech som inte android tillhandahåller.

KatlaSmsManager

KatlaSmsManager är ett paket som sköter skickning av sms via androids SmsManager och fungerar som en service. Paketet består av en factory, ett interface och en implementation av det interfacet. Anledningen till att servicen finns är för att enkelt kunna sköta skickning av meddelanden på ett sätt som inte går via en Default Sms Application eller på något sätt förändra hur skickningen går till.
För närvarande fungerar det inte som tänkt att skicka sms som är längre än 160 tecken. Det löses genom att flera separata sms skickas som är max 160 tecken långa.

API nivå

Appen använder sig av Android API nivå 19, kallat KitKat eftersom det var först då som konceptet att ha en standard sms app introducerades. Före KitKat fanns det inget sätt för användaren att välja vilken applikation som skulle hantera text meddelanden, detta tvingade utvecklarna att använda API klasser som inte officiellt var stödda av Android. Om vi hade valt att istället använda dessa icke officiella klasser så hade våran applikation inte funkat på telefoner med API nivå 19 eller högre. Även XML parsningen kräver API nivå 19.

Katla - singleton

Modelklassen Katla är en singleton, detta framförallt för att när man skapar en app som är tänkt att agera som telefonens inbyggda sms app måste den kunna hantera anrop (Så kallade intents) från andra appar. Dessa anrop kan gå ut på att en annan app vill att våran app ska utöra en specifik uppgift som till exempel att skicka ett sms. På grund av att vi inte har någon kontroll över dessa anrop måste vi se till att varje del av programmet alltid kan komma åt modelklassen utan att behöva bry sig om den är instansierad på korrekt sätt eller inte.

Model, View & Controller

Appen är byggd efter designmönstret ‘Model-view-controller’ som går ut på att man separerar appens funktionalitet och användargränssnit så långt som möjligt. Dock är det svårt att strikt följa MVC mönstret i Android eftersom kontrollern och vyn är så hårt sammankopplade. I Android motsvarar kontrollern en aktivitet, men i aktiviteten måste man även lägga kod som på olika sätt modifierar användargränsnittet då det laddas in från en statisk XML fil.

EventBus

Appen använder sig av en EventBus för att möjliggöra lösa kopplingar mellan delar i appen som behöver få information från andra delar. Det är endast AGAListener som skickar events via EventBussen och Katla och SendMessage lyssnar.

Applikationsdata - XML

De fördefinerade meddelanden som användaren kan välja i appen är sparade i XML format. Alla meddelanden är indelade i kategorier, dessa kategorier har ett namn och ett antal meddelanden. Meddelanden i sin tur innehåller en text samt en input tag, input taggen innehåller en kommaseparerad lista med input typer. De två input typer som stöds för närvarande är time och date som kommer visa en dialog där användaren kan välja tid respektive datum att ha med i det fördefinerade meddelandet. Det finns ingen gräns på hur många av dessa input typer som kan vara definerad för ett meddelande men för varje input typ måste textsträngen “%s” finnas med i texten. När användaren sedan väljer att skriva

meddelandet kommer varje förekomst av strängen “%s” ersättas av datan från input typen i den ordning de är skrivna i input taggen.

Aga

Projektet kräver AGA bibliotek för att kunna köras. De bibliotek som används är:

- SDP
- VIL
- Automotive-API
- slf4j-api samt slf4j-simple

Dessa bibliotek möjliggör interaktion med en fysisk(eller virtuell) lastbil vilket projektet kräver för att kunna ändra sitt gränssnitt då lastbilen befinner sig i olika stadium. Det är endast signalen DriverDistractionLevel som projektet för närvarande reagerar på.

Robolectric

Robolectric är ett bibliotek som möjliggör testning av androidspecifika saker utanför en avm. Det krävs i projektet för att kunna skriva tester som testar funktionalitet som beror på androidimplementeringar, till exempel testas skickningen av ett meddelande via robolectric. Detta är saker som hade gått att testa under en körning av applikationen men med hjälp av Robolectric går det att generera flera tester som kan fort kan kontrollera att funktionalitet bibehålls vid förändringar etc. Tyvärr stödjer Robolectric ännu inte API nivå 19 vilket gör att vissa delar i applikationen, särskilt de som nämns under API rubriken, inte går att testa via Robolectric.

För att köra tester med robolectric kan denna tutorial med fördel följas:
http://blog.blundell-apps.com/how-to-run-robolectric-junit-tests-in-android-studio/