Av: Saga Hassellöf
Datum: 27/8 -20
Kurs: 5DV209

Bettbok

- en loggbok för dina fästingbett.

Innehållsförteckning

1. Introduktion

1.1. Målgrupp

Bettbok är till för de som vill hålla koll på fästingbett de fått. Bettbok hjälper användaren att upptäcka en eventuell förändring av huden runt bettet som kan fungera som en indikation på en Borreliainfektion.

1.2. Beskrivning av applikationen

Få hjälp med att skydda dig mot Borrelia med Bettbok - en loggbok för att hålla koll på dina fästingbett.

Även om det finns många hälsofördelar med att vara ute i naturen, så finns det också risker. En av dessa risker utgörs av fästingar. Om en människa blir biten av en fästing och denna fästing bär på en smitta så finns det en risk att smittan överförs till människan. En av dessa smittor är Borrelia. Denna smitta visar symtom efter 2 till 4 veckor och kan ge problem med värk i leder, påverka av centrala nervsystemet och i enstaka fall även hjärtat. Borrelia drabbar årligen 10 000 människor i Sverige och går inte att förebygga genom vaccination. Däremot finns andra sätt att skydda sig, såsom att leta på kroppen efter fästingar när man har varit ute och att plocka bort fästingar, som har bitit sig fast, så fort som möjligt.[1] Vid ett fästingbett som leder till en Borreliainfektion hos människan, finns även en 50% chans att en röd ring bildas kring bettet i början av sjukdomsförloppet.[2] Denna röda ring kan användas som en indikator för att tidigt söka vård och få eventuell behandling.

Bettbok är en loggbok som hjälper dig att hålla koll på dina fästingbett och upptäcka eventuella förändringar kring bettet. För varje bett kan du beskriva vart på kroppen fästingen satt, när du fick det, vilket stadie av livscykeln fästingen var i och du kan även ta bilder på bettet. Du kan sedan gå tillbaka till varje bett för att påminna dig om vart det satt, hur lång tid som har gått sedan du fick det och kolla upp om en förändring har skett. Bettet kan kompletteras med en bild efter 2 veckor och en efter 4 veckor för att hjälpa till med att se en eventuell förändring.

Betyg jag siktar på: G

2. Säkerhet

2.1. Applikationens rättigheter

Om en applikation begär att få åtkomst till viss data eller vissa funktioner på telefonen måste den också hantera denna information på ett säkert sätt. Om applikationen till exempel har rättighet att använda telefonens GPS och att spara information på en extern plats, skulle det vara oansvarigt att spara den senaste positionen på ett sådant sätt som andra appar kan komma åt informationen. För att få dessa rättigheter behöver applikationen få ett godkännande från användaren. Med varje godkänd rättighet kommer ansvar och för att minska säkerhetsriskerna så rekommenderas det att en applikation endast ska begära rättighet för de funktioner som den verkligen behöver.[3]

Bettbok använder kameran för viss funktionalitet i applikationen och dessa bilder ska ses som privat data som ska hållas säker. Men applikationen begär aldrig någon rättighet för att varken ta kort eller för att spara informationen. Detta beror på att Bettbok skickar en förfrågan till telefonens kamera om att ta ett kort. I och med att kameran redan har tillåtelse till att ta kort så behöver inte applikationen denna rättighet. Bilderna sparas sedan i Bettboks egna sandbox som är en privat lagringsplats andra appar inte kan komma åt. Men att spara bilderna i den egna sandboxen gör att kameran inte har tillgång till att spara bilder där. Detta löses genom att använda en FileProvider, vilket är en subklass till ContentProvider. Denna subklass används för att styra över hur applikationen kan dela med sig av sina filer.[4] För att skydda sin applikation från att inte dela med sig av mer filer än det var tänkt, är exporten i manifestet satt till false. Denna FileProvider används nu för att ge kameran tillåtelse att spara ner en bild under en given fil i applikationens sandbox. Rättigheten tas sedan bort när kameran är färdig med sin uppgift.

2.2. Användarens information

En applikation ska spara så lite information som möjligt om användaren. All information en användare ger, delas med ett förtroende. Ju mer information applikationen kräver, desto mer förtroende kräver man av användaren. Detta är ett förtroende som inte får svikas. Den information som applikationen samlar kan äventyras på olika sätt. Antingen genom att applikationen sparar information på ett felaktigt sätt så att andra applikationer kan komma åt data, genom att applikationen skickar information över ett nätverk utan kryptering, genom att lämna log-meddelanden som kan avslöja information om användaren eller genom att telefonen blir stulen eller borttappad vilket kan leda till att information hamnar i fel händer.[3][5]

Bettbok behöver inte åtkomst till någon annan information på telefonen (så som kontakter) än den som användaren själv anger för varje bett. Denna information sparas i en SQLite-databas och som bildfiler i applikationens sandbox, vilket gör att den inte kan kommas åt av andra appar. Bettbok sänder inte heller information över nätverk och är därför inte exponerad för risker kopplade till detta. Däremot kan applikationens information

kommas åt om telefonen blir stulen eller borttappad. Användaren bör därför hålla noga uppsyn över sin telefon och gärna skydda den med någon sorts lås såsom pin-kod eller liknande. Om intresse finns för att skydda informationen ytterligare mot denna typ av fara, kan detta ses över i en uppdatering av Bettbok. I denna uppdatering skulle användaren kunna få möjligheten att ha en inloggning som används för att ge åtkomst till information kopplad till en användare.

2.3. Databasen

Bettbok använder en SQLite-databas för att spara information som matas in av användaren. För att hämta information från databasen används ett query. Detta gör att databasen potentiellt är känslig för attacker genom SQL-injection. Detta är attacker där SQL-kod skickas in som användarinput. Denna input kan då förändra hur queryt används.[6] Bettbok har skyddats mot detta genom att alltid tolka användarens input som en sträng. Detta gör att den eventuella SQL-kod som attackeraren matar in inte tolkas som kod och därför inte kan förändra hur queryt används.

3. Manual för Bettbok

3.1. Lista över alla bett

Den första vy som möter användaren är en lista över alla sparade bett som användaren har gjort (vy A, figur 1). Varje bett visas med en placering som rubrik, hur många dagar det gick sen man fick bettet som underrubrik och om det finns en tillgänglig bild visas denna till vänster om placeringen och antalet dagar. Bilden är till för att visa hur bettet såg ut precis när användaren fick det. Placeringen återger vart på kroppen fästingen hade bitit sig fast. Antalet dagar används för att kunna avgöra om bettet behöver ses till. Listan är sorterad så att det bett man fick senast ligger överst.

Längst ner till höger finns en floating action button med en bild av en fästing som kan användas för att skapa ett nytt bett. När användaren använder detta alternativ kommer ett nytt bett att skapas och användaren kommer sedan att tas till vyn för att redigera detta bett (pil A1, figur 1).

3.2. Skapa nytt bett

När ett nytt bett skapas möts användaren av en vy för redigering (vy B, figur 1). Här finns alternativ för att lägga till vart på kroppen man har fått bettet, när man fick bettet, fästingens stadie i sin livscykel och om man vill ta kort på bettet.

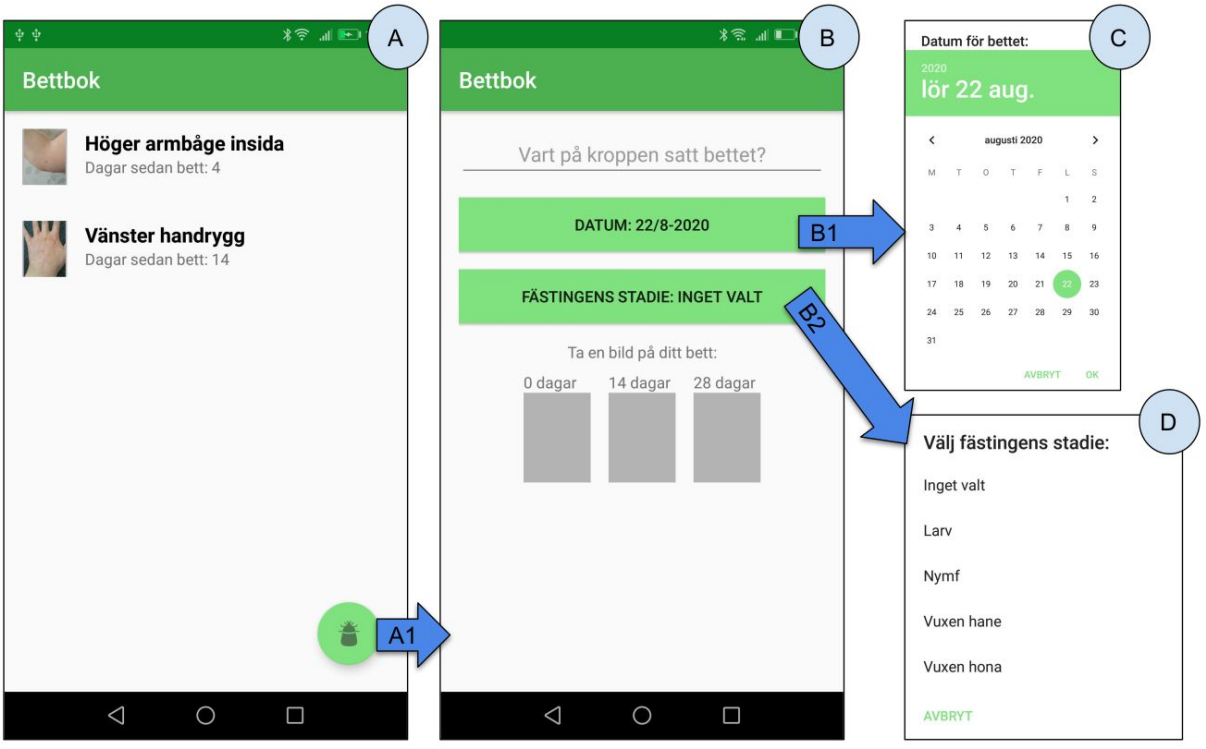
Det datum då man fick bettet är per default inställt på dagens datum. För att ändra detta trycker man på det aktuella datumet (pil B1, figur 1). Användaren möts då av dialogen “Datum för bettet:” (vy C, figur 1). Här visas en månad åt gången och användaren kan leta upp rätt datum för bettet. När rätt datum är markerat klickar man på “OK” för att spara

datumet och gå tillbaka till vyn för att fortsätta editera bettet. Om användaren ångrar sig och inte vill lägga till ett nytt datum kan man trycka på antingen “AVBRYT” eller på bakåt-navigeringen.

Fästingens stadie är per default inställt på “Inget valt”. Om användaren vill lägga till ett stadie för fästingen kan detta göras genom att trycka på det aktuella stadiet (pil B2, figur 1). Användaren möts då av dialogen “Välj fästingens stadie:” (vy D, figur 1). Denna vy innehåller en lista av olika alternativ. Här finns alternativen att inte välja något stadie, larv, nymf, vuxen hane eller vuxen hona. Användaren väljer det alternativ som gäller för just detta bett genom att klicka på alternativet. Detta kommer då sparas och användaren tas tillbaka till vyn för att fortsätta editera bettet. Om användaren ångrar sig och inte vill ändra alternativet kan alternativet “AVBRYT” användas, alternativt bakåt-navigering eller trycka utanför dialogen.

Användaren kan även välja att ta kort på sitt bett. Detta görs genom att trycka på det foto man vill ska bytas ut. Det finns tre möjliga foton att ta, där det första är precis när man fick bettet och det andra och tredje fotot är 14 respektive 28 dagar efter att man fick bettet. Om ett foto inte har lagts till visas endast en grå ruta. När användaren klickar för att ta ett foto öppnas kameran. När användaren tagit en bild denne är nöjd med och bekräftat detta sparas bilden och visas upp på vald plats.

När användaren är klar med redigeringen av bettet används bakåt-navigeringen för att komma tillbaka till vyn med listan av bett, det nya bettet sparas automatiskt.

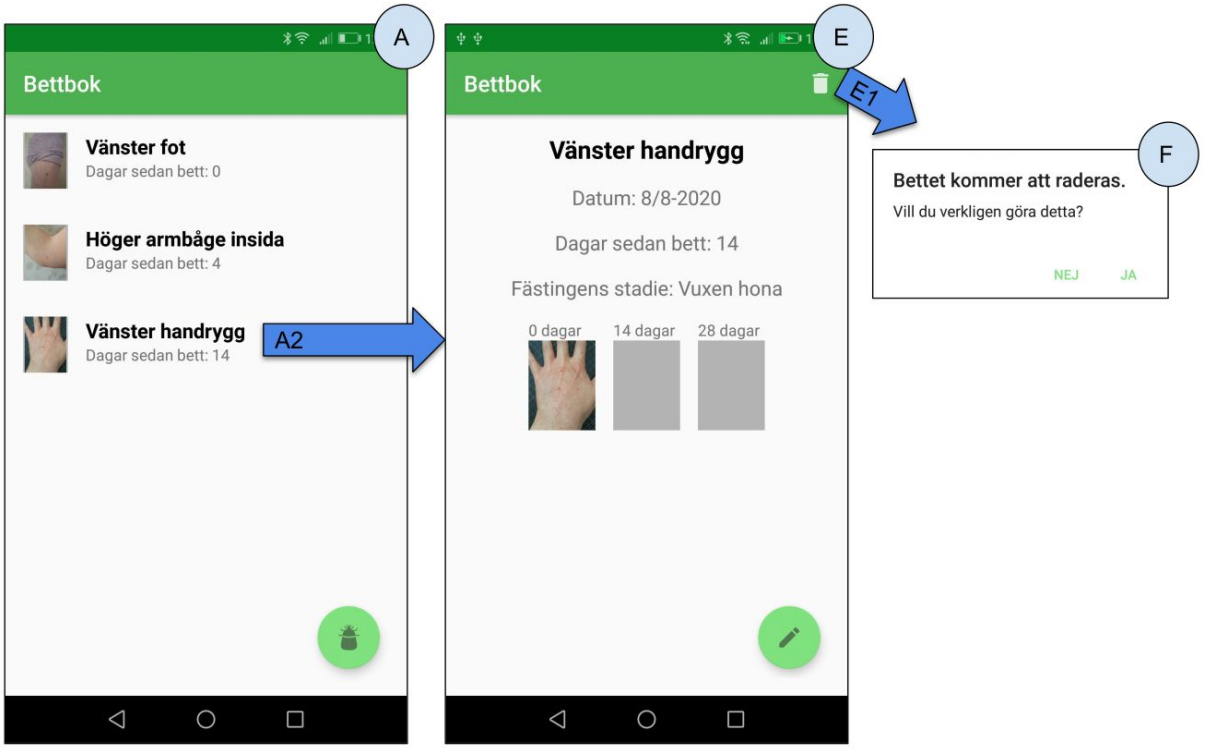


Figur 1: Visar vy för listan av bett och vyer för att skapa ett nytt bett samt övergångar mellan dessa.

3.3. Översikt över ett bett

För att komma till vyn där ett bett visas (vy E, figur 2), trycker man på det aktuella bettet i listan över bett (pil A2). Överst i vyn visas vart på kroppen bettet satt. Nedanför visas det datum och hur många dagar som har gått sedan man fick bettet. Därefter följer det stadie i livscykeln fästingen var som gav bettet. Tillslut visas de bilder som användaren har tagit för att dokumentera bettet. Om ett foto inte har tagits till någon av de tre bilderna visas en grå ruta istället.

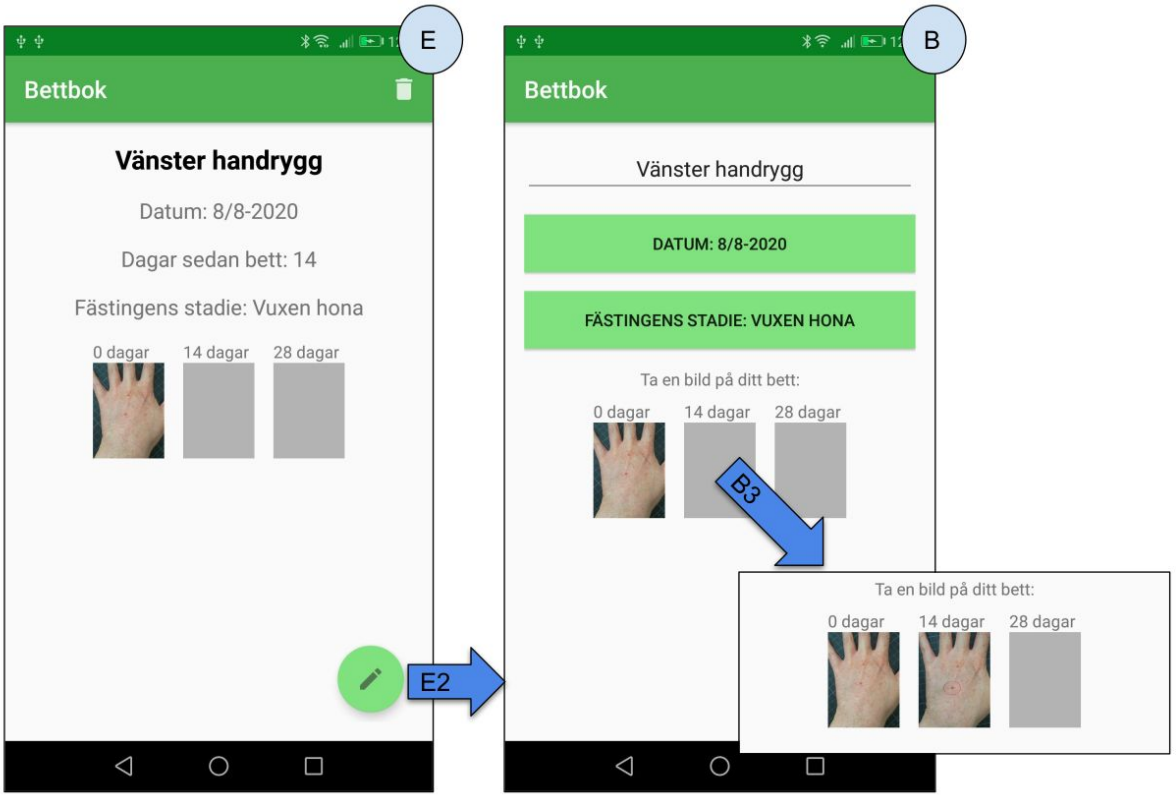
Förutom att granska bettets uppgifter, kan användaren också välja att antingen radera eller att redigera bettet. För att redigera bettet trycker användaren på den floating action button med en penna som finns i det nedre högra hörnet (pil B3, figur 3). Om användaren istället vill radera bettet kan detta göras genom att trycka på soptunnan i applikationens action bar (pil E1, figur 2). En dialogrutan kommer då be användaren att bekräfta att bettet ska tas bort (vy F, figur 2). Om användaren trycker ”NEJ” kommer denne tas tillbaka till vyn för bettet utan att något händer. Om alternativet “JA” används kommer bettet att tas bort och vyn med listan över alla bett visas. När användaren känner sig nöjd med granskningen av bettet kan denne gå tillbaka till listan av alla bett genom att använda bakåt-navigeringen.



Figur 2: Visar hur ett existerande bett väljs och visas upp i en egen vy och alternativet för att radera detta bett.

3.4. Redigera ett befintligt bett

Om användaren väljer att redigera ett befintligt bett, kommer samma vy visas som när ett nytt bett skapas med skillnaden att den information som redan finns om bettet kommer vara ifyllt (vy B, figur 3). Användaren kan ändra vart på kroppen bettet satt, vilket datum denne fick det och fästingens stadie. Användaren kan även välja att byta ut någon av de bilder denne redan tagit eller lägga till en ny bild (pil B3, figur 3). När användaren är klar med sina ändringar trycker denne på bakåt-navigeringen för att komma tillbaka till vyn som visar bettets information.



Figur 3: Visar hur ett redan existerande bett kan redigeras.

4. Uppbyggnad av kodbasen

4.1. Model

De bett som användaren lägger till görs till instanser av klassen Bite. Ett Bite sparar information såsom tiden bettet ägde rum med hjälp av en Calendar och bettets placering och stadiet på fästingen i en sträng. Varje bett har även ett id i form av ett UUID. Klassen Bite har två constructors för att antingen skapa ett bett med ett random id eller ett bett med ett givet id. Förutom getters och setters finns också en metod för att få en sträng som ska kunna användas som filnamn. Denna sträng bygger på bettets id och den int som är ett argument till metoden. Det finns även en metod för att få antalet dagar från bettets datum fram till det aktuella datumet.

För att andra klasser ska kunna få tillgång till betten används klassen BiteLab. Detta är en singleton-klass som kan kommunicera med applikationens databas för att hämta ett bett med givet id eller alla sparade bett. BiteLab hjälper också till med att lägga till, ta bort och uppdatera bett i databasen. Hur BiteLab använder sig av databasen kan ses i figur 4. Det finns även metoder för att få filer som kan användas för att spara foton kopplade till ett bett. Uppbyggnad av model och hur klasser i andra paket interagerar med BiteLab och Bite kan ses i figur 4.

4.2. Database

Bettbok använder en SQLite-databas för att spara Bites i applikationens sandbox. Tre klasser används för att hantera databasen. BiteDbSchema definierar de kolumner som databasen använder. BiteBaseHelper ansvarar för att skapa databasen med hjälp av detta schema. BiteCursorWrapper tillhandahåller en metod för att plocka ut information för en specifik rad ur databasen och göra om denna rad till ett Bite. Uppbyggnad av database och hur interaktion sker mellan model och database kan ses i figur 4.

4.3. Controller och view

Navigeringen genom applikationens vyer sker med hjälp av olika fragment i controllern. Navigeringen mellan huvudfragmenten BiteListFragment, BiteFragment och BiteEditFragment sker med hjälp av en NavController. En översikt över navigeringen mellan fragmenten kan ses i figur 6. För en mer detaljerad bild används figur 4 och figur 5. I figur 4 beskrivs hur klasserna använder sig av applikationens model och i figur 5 visas vad för xml-filer som varje klass använder.

4.3.1. BiteListFragment

När applikationen startas används fragmentet BiteListFragment. Detta fragment använder sig av fragment_bite_list från viewn. Här kontrolleras dels den RecyclerView som används för att visa upp Bites och även en floating action button som används för att skapa ett nytt

Bite. I fragmentet finns Adapter och ViewHolder för att hantera RecyclerViewn och adaptern använder sig av list_item_bite från viewn. I list_item_bite sätts information till de olika fälten som representerar ett Bite och klassen AsyncImageScaler används för att läsa in bilder. Om en användare trycker på representationen av ett Bite i listan skapas en Bundle med det aktuella bettets id som argument. Denna Bundle läggs sedan till när navigering sker till fragmentet BiteFragment. Om användaren istället väljer att skapa ett nytt bett läggs ett nytt Bite till i databasen genom BiteLab. Sedan läggs detta betts id till i Bundlen som skickas med vid navigering till fragmentet BiteEditFragment.

4.3.1. BiteFragment

BiteFragment använder fragment_bite från viewn. När navigering sker till BiteFragment hämtas id, för det Bite som ska visas upp, från Bundlen. Detta id används för att hämta information för bettet och fylla i detta i tillhörande fält i fragment_bite. BiteFragment kontrollerar även uppvisningen av tre foton kopplade till bettet. Klassen Image används för detta (mer information under Image). BiteFragment använder även en Action bar, menu_bite. I denna meny finns alternativet att ta bort det aktuella bettet. Om användaren väljer detta kommer en AlertDialog att visas för att bekräfta att detta är något användaren verkligen vill göra. Ifall användaren väljer “JA” i denna dialog kommer BiteFragment använda BiteLab för att radera bettet ur databasen och sedan navigera tillbaka till BiteListFragment. Väljs “NEJ” stängs dialogen ned. BiteFragment kontrollerar även en floating action button som låter användaren navigera till BiteEditFragment. Om detta välj skickas bettets id med som ett argument i Bundeln.

4.3.1. BiteEditFragment

När navigering sker till BiteEditFragment hämtas idt, för det Bite som ska visas upp, från Bundlen. Detta id används sedan för att hämta tillhörande Bite från databasen. BiteEditFragment kontrollerar de fält i fragment_edit_bite som används för att redigera ett bett. Det Bite som har hämtats används för att fylla i den information som redan är sparad för bettet. Klassen använder sig även av Image för att visa upp foton om dessa finns tillgängliga. Image används också för att göra det möjligt att trycka på fotona och därmed starta kameran (mer information under Image). För att ändra datumet för bettet startas en DatePicker med hjälp av klassen DatePickerDialog som är en subklass till DialogFragment. Bettets datum skickas med till dialogen med en Calendar för att dialogen ska kunna sätta detta som det initiala datumet. Om användaren väljer ett nytt datum skickas även detta datum som en Calendar tillbaka till BiteEditFragment. För att välja ett nytt stadie för fästingen används klassen StagePickerFragment som också är en subklass till DialogFragment. De möjliga alternativen presenteras i en lista där användaren kan välja ett alternativ. Detta alternativ skickas tillbaka som en sträng.

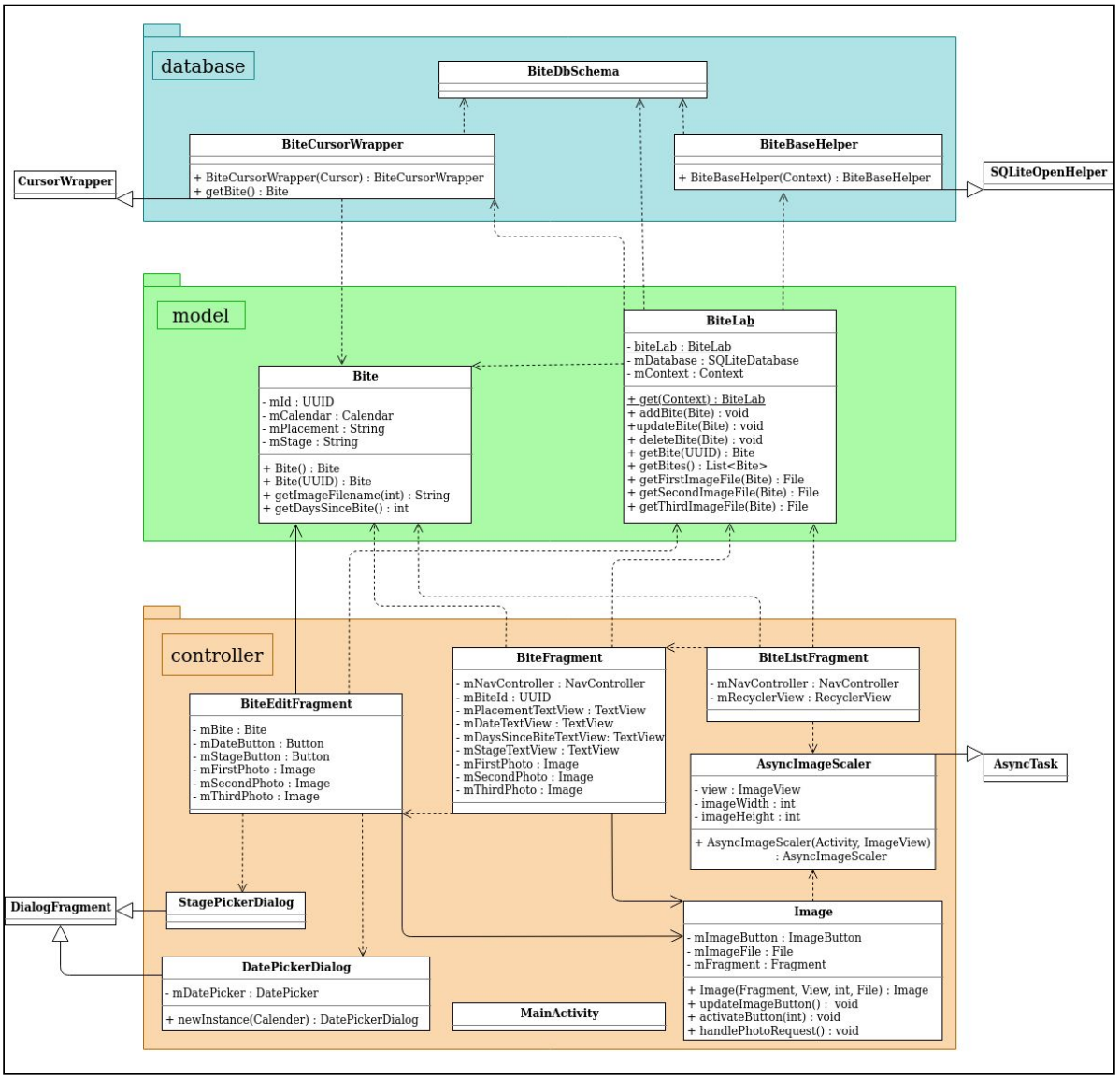
4.3.4. Image

Klassen Image hanterar en fil och en ImageButton som används för att visa upp bilden filen innehåller. Om filen inte existerar kommer denna ImageButton endast visas i grått. För att ge möjligheten att trycka på bilden för att ta ett foto, innehåller Image metoden activateButton.

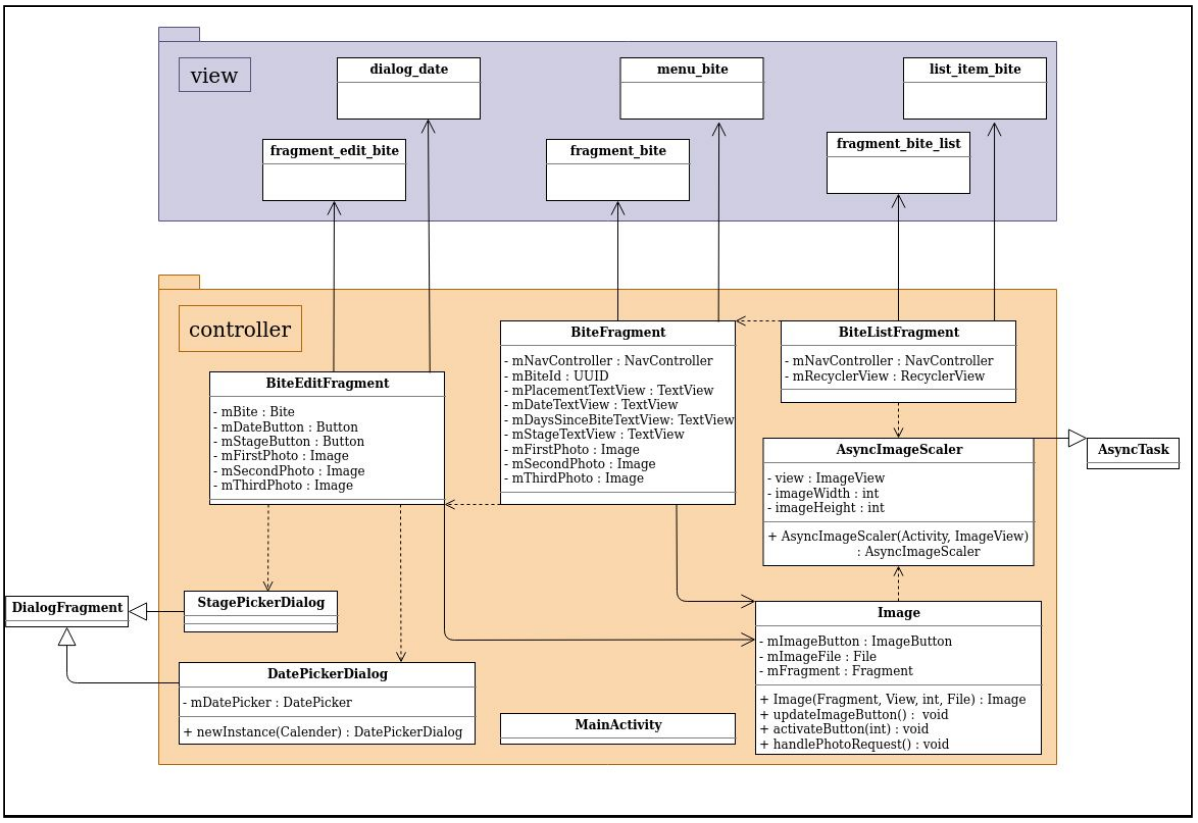
Denna metod tar som argument den int som används för att hämta svaret från kameran. Om denna funktion aktiveras sätts en onClickListener som ger kameran tillåtelse att skriva till den givna filens uri och sedan starta kameran med angiven int som requestCode. Image innehåller även metoden handleRequest som används för att återkalla kamerans rättighet att skriva till filens uri.

4.3.5. AsyncImageScaler

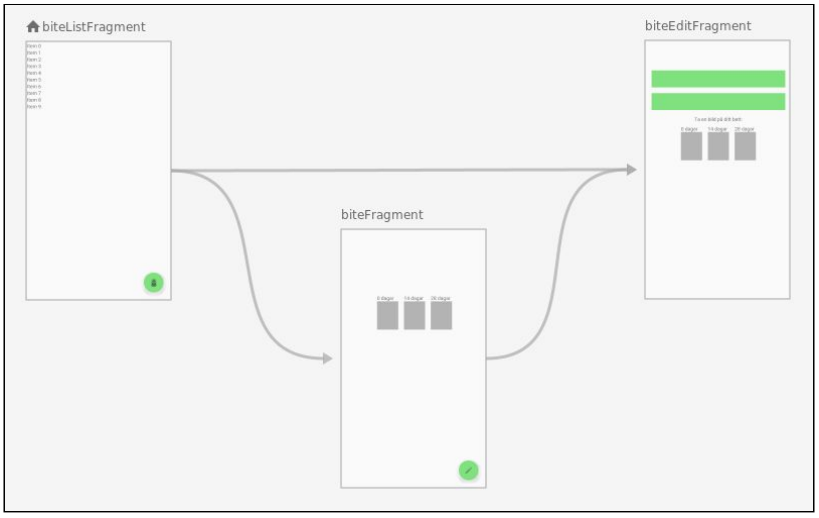
För att en bild ska kunna visas upp i Image och för att visa bilder i recyclerViewn i BiteListFragment används klassen AsyncImageScaler. Här skapas en egen tråd med hjälp av AsyncTask. Denna tråd är till för att läsa in och skala om en bild för att passa en viss ImageView. När detta är gjort sätter också AsyncImageScaler bilden som en Bitmap i den givna ImageViewn.



Figur 4: Visar klasserna i paketen database, model och controller och deras interaktion.



Figur 5: Visar klasserna i paketen controller och view och deras interaktion.



Figur 6: Visar en översikt över navigeringen genom applikationen.

5. Diskussion

5.1. Intressanta problem

I början av utvecklingen hade jag en aktivitet kopplad till varje fragment. När jag skulle byta från att ha flera aktiviteter till endast en uppstod en bugg. Problemet uppstod när BiteEditFragment sparade undan ändringar av bettet till databasen i sin onPause() och navigation sedan skedde tillbaka till exempelvis BiteFragment. När BiteFragment startade upp igen hämtades inte den nya informationen. Detta trots att BiteFragment använde onResume() för att uppdatera sin information mot databasen. Det visade sig att efter att ha loggat onPause() och onResume() så kördes onResume() i BiteFragment innan onPause() i BiteEditFragment, vilket inte var fallet när varje fragment hade sin egen aktivitet. Detta resulterade i att BiteFragment hämtade informationen från databasen innan BiteEditFragment hade hunnit uppdatera den. Detta löstes genom att uppdatera bettet i databasen direkt när något ändras i BiteEditFragment.

Problem uppstod också när AsyncImageScaler skulle försöka ta ut bredd och höjd på den view som skickades in. Det visade sig att storleken på alla layouts inte sätts förens layout pass händer för första gången, vilket är vid ett senare tillfälle än när AsyncImageScaler försöker hitta måtten. Lösningen på detta blev att kolla upp hur stor skärmen är och använda detta som ett sorts maxmått för hur stor viewn kommer vara. Detta är en lösning som presenteras i Android Programming - The Big Nerd Ranch Guide. De beskriver lösningen dock som mindre effektiv så uppskattningen av storleken kan vara dålig. En annan lösning skulle istället kunna vara att vänta med att hämta ut måtten tills första layout pass äger rum.[7]

Jag har även försökt använda Exif för att hantera rotation av foton men har tyvärr haft stora problem. Länge ville det inte fungera alls med konstiga RunTimeExceptions om att metoder för Exif inte existerade trots att Exif var implementerat i build.gradle. Vid ett senare tillfälle började det fungera dock, tyvärr oklart varför problemet försvann. Då uppstod istället problem med att skalan på bilderna blev konstig. Detta löste sig dock genom att använda en kombination av BitmapFactory.Options för att hitta den optimala skalningen och använda dessa mått med den roterade Bitmapen i Bitmap.createScaledBitmap(). Tyvärr fick jag istället problem med att BitmapFactory.decodeFile() åt upp allt minne. Detta hann jag aldrig lösa, vilket leder till att den lösning jag nu använder inte hanterar rotering av bilderna. Det är inte helt väsentligt för applikationens funktionalitet att bilderna på betten är roterade korrekt. Den är väldigt retligt att inte ha löst detta med tanke på all den tid som jag lagt ner på Exif och då det förstås är önskvärt för applikationen.

Jag hade svårt att hitta en lösning för att hantera flera bilder kopplade till ett bett. Lösningen jag använder har bara tre “låsta” bilder. Istället hade jag velat ha ett flexibelt antal bilder som användaren har mer kontroll över själv. Detta hade kanske kunnat hanteras med någon sorts RecyclerView som skapar ett galleri kopplat till varje bett.

5.2. Förslag på funktionalitet inför framtiden

- Det kan vara svårt att förklara exakt vart ett bett satt på kroppen för att flera dagar senare kunna gå tillbaka och kolla på exakt samma plats. En bättre lösning hade varit att ha ett antal bilder som representerar en kropp från olika håll. På någon av dessa bilder hade man sedan kunnat markera med en prick exakt vart bettet satt.

- När applikationen visar listan med de stadier som är möjliga att välja för fästingen ska det även finnas möjlighet att få information om hur en fästing från de olika stadierna kan se ut.

- Användaren kan även ha intresse av att veta vart de var när de fick bettet. Detta hade kunnat sparas genom att användaren kan använda sig av en karta och gps för att markera vart de tror de var. Detta visas sedan som en pin på en karta.

- Användaren skulle kunna vilja kolla på de bilder de har tagit i mer detalj. För att tillhandahålla detta skulle bilderna i EditFragment kunna visas i helskärm om man trycker på en viss bild.

- Användaren bör kunna spara så många bilder de vill för ett bett. Dagens datum för när fotot tas skulle kunna användas för att visa hur många dagar som har gått istället för att detta ska vara låst till 0, 14 och 28 dagar.

- För att användaren inte ska glömma bort att kolla till ett bett bör de påminnas ett antal dagar efter bettets sparade datum. Hur ofta de påminns bör kunna avgöras av användaren själv. Alternativ skulle kunna vara att inte påminnas alls, en gång i veckan i 3 veckor, efter att 4 veckor gått eller liknande.

- Användaren bör kunna sortera listan av bett som de önskar. Betten skulle kunna sorteras i både fallande och stigande ordning. Man ska även kunna välja att bara visa bett från en viss månad, vecka eller liknande.

6. Slutsats

I denna rapport beskrivs applikationen Bettbok som ska fungera som en loggbok för en användares fästingbett. Syftet med detta är att hjälpa användaren att hålla koll på eventuella hudförändringar runt bettet som kan tyda på en Borreliainfektion. Rapporten innehåller en introduktion av Bettbok samt en manual för hur applikationen är tänkt att användas. Detta följs av en förklaring av hur kodbasen är uppbyggd och hur olika klasser och paket interagerar. Rapporten har också en diskussion med de problem som uppstod under utvecklingen av applikationen och vad som gjordes för att försöka lösa problemen. I diskussionen finns även några exempel på funktionalitet som skulle kunna utveckla och förbättra applikationen.

7. Källförteckning

[1]: Fästing.nu. *Borrelia.*
https://www.fasting.nu/borrelia#
(Hämtad 2020-08-27)

[2]: Borreliakollen.
https://borreliakollen.se/?gclid=CjwKCAjw1ej5BRBhEiwAfHyh1CaoePQSQnl6LoxEYVHzEuBOwLS-ynrqPZIwLAR0b-4qZiiLDNjyxRoCvPEQAvD_BwE
(Hämtad 2020-08-27)

[3]: Larimer, Jon. Root, Kenny. 2012. *Google I/O 2012 - Security and Privacy in Android Apps*
https://www.youtube.com/watch?v=RPJENzweI-A&feature=player_embedded
(Hämtad 2020-08-27)

[4]: Developers. *FileProvider.*
https://developer.android.com/reference/androidx/core/content/FileProvider
(Hämtad 2020-08-27)

[5]: Developers. 2010. *Best Practices for Handling Android User Data.*
https://android-developers.googleblog.com/2010/08/best-practices-for-handling-android.html
(Hämtad 2020-08-27)

[6]: PortSwigger. SQL injection.
https://portswigger.net/web-security/sql-injection
(Hämtad 2020-08-27)

[7]: Philips, Bill; Stewart, Chris; Marsicano, Kristin. 2017. *Android Programming: The Big Nerd Ranch Guide.* 3 uppl. Indianapolis: Pearson Technology Group.