Qiwi Quiz

Krav- och analysdokument för “Qiwi Quiz”

Carl Bergman, Erik Blomberg, Henrik Johansson, Sara Persson, Louise Tranborg

2020-10-23

1. Introduktion
Syftet med detta projekt är att skapa en inlärnings-applikation. Appen ska vara ett frågesportspel där användaren kan öva upp sina färdigheter, primärt inom det grundläggande ämnet matematik. Användaren skall kunna följa sin utveckling med hjälp av en statistiksida där deras historik sparas ner efter varje spelad quiz. De ska även kunna justera spelet utifrån behov med hjälp av en inställningssida, där bland annat olika spellägen ska vara tillgängliga.

Projektet syftar också stort till att göra applikationen framtidssäker och utbyggnadsbar, där nya ämnen, nya quiz och nya funktioner enkelt skall kunna att läggas till. Applikationen ska även vara justerbar och personlig för användaren, där exempelvis egna quiz ska kunna läggas till.

Applikationens riktar sig huvudsakligen till grundskoleelever men även kan även tilltala andra personer som har ett intresse att utmana sina kunskaper. En skolelev med svårigheter i matematik kan vända sig till denna applikation för att komma ikapp på ett lite roligare sätt. Men även en yrkesverksam kan få nytta av applikationen genom att exempelvis öva upp sina historiekunskaper för att kunna briljera för sina arbetskamrater. Kunskap är makt, men kunskapen är inte alltid lättillgänglig och underhållande att arbeta sig igenom. “Qiwi Quiz” är därmed ett sätt göra möjligheten att lära sig mer tillgänglig och framför allt roligare att genomföra.

1.1. Definitioner, akronymer och förkortningar
Användarberättelser - En dokumentationsteknik som utgår från användaren och dess sätt att uttrycka sina behov och önskemål. De gör det enklare att hålla arbetssättet, från start till färdig produkt, organiserat och fokuserat [1]. Varje användarberättelse innehåller en beskrivning som följer en viss mall, där användarens önskemål och anledning till detta önskemål presenteras. Berättelsen innehåller utöver detta olika former av uppgifter som utvecklaren skall utföra. Sedan har de även en identifikation (id), namn och en eventuell svårighetsgrad för att kunna dokumenteras och kategoriseras på ett smidigt sätt.
Sprint - En relativt kort arbetsgenomgång där ett antal användarberättelser har valts ut för att genomföras. Tidsramen för en sprint brukar ligga mellan en till tre veckor. Varje sprint leder idealt till ett körbart program. Efter varje sprint granskas koden och följande sprint planeras [2].

2. Kravspecifikation
De övergripande kraven för “Qiwi Quiz” är sammanfattade nedan. Denna sammanfattning låg sedan som grund för skrivandet av användarberättelser. Dessa berättelser hittas under punkt 2.1 och 2.2 i detta dokument.

Spelaren skall kunna:
1. Logga in/skapa ett konto.
2. Spela ett quiz.
    a. Detta quiz ska gå att välja bland en rad olika kategorier.
3. Se resultat efter spelat quiz.

    a. Detta även nersparat i en statistikvy som spelaren kan besöka.
4. Besöka en inställningsvy.
    a. Där skall exempelvis olika spellägen kunna väljas.
5. Skapa egna quiz.

2.1 Genomförda användarberättelser

Nedan listas de användarberättelser som slutfördes under detta projekt. De ligger organiserade enligt deras prioritet, från högst till lägst prioritet.

Det användes en svårighetsgrad på varje berättelse för att göra det enklare att organisera varje sprint, så att arbetet fördelades lika mellan utvecklarna och samtidigt var genomförbart under tidsramen. Varje sprint speglade en arbetsvecka genom hela projektet. Dock så uppstod ibland svårigheter att bedöma svårighetsgrad dels då det är en subjektiv bedömning och dels för att oförutsägbara problem kan uppstå för utvecklaren under själva implementationen av en användarberättelse. Detta gjorde att ett fåtal användarberättelser fick vara aktiva under två sprintar, alltså under två arbetsveckor. Användarberättelserna med identifikation 09 och 07 var sådana fall där svårigheter uppstod under implementation. Dessa fick hänga kvar i en halv eller en hel sprint extra innan de blev klara. Berättelsen med identifikation 07 visade sig vara beroende av ytterligare implementation och blev därmed en mycket större uppgift än väntat.

ID: 01.
Namn: Val av ämne.
Svårighetsgrad: 3.
Beskrivning: Som användare vill jag kunna välja vilket ämne jag ska plugga på för att hitta passade quiz för mig.
Funktionellt:

- Jag kan se en meny av knappar (alternativt bara en; "Matte"-knapp).
    - Ifall jag trycker på en av knapparna förs jag till en "placeholder" sida.
- Högst upp i "Toolbar:en" står det ämnet, till exempel Matte och en bakåtpil för att gå tillbaka till förra aktiviteten.

ID: 02.
Namn: Fråga/Svarsalternativ
Svårighetsgrad: 4-5
Beskrivning: Som en användare behöver jag presenteras av en fråga samt fyra olika tillhörande svarsalternativ för att kunna välja det alternativ som jag tror är rätt.
Funktionellt:

- Jag kan se fyra knappar,
- Knapparna presenterar de svarsalternativ som hör till frågan.
- Jag kan trycka på en av knapparna för att registrera det svar som jag tror är korrekt.
- Jag kan se en fråga över de fyra knapparna.

ID: 03.

Namn: Resultat efter quiz.
Svårighetsgrad: 5.
Beskrivning: Som användare vill jag få ett resultat när jag är färdig med mitt quiz för att se hur bra jag gjorde.
Funktionellt:

- Sidan har en rubrik “Resultat”.
- Sidan har ett resultat speglat hur många rätt av alla jag fick, ex. 7/10.
- Sidan har knapp för att stänga ner resultatsidan för att kunna komma tillbaka till startsidan.

ID: 04.
Namn: Startsida.
Svårighetsgrad: 1-2.
Beskrivning: Som användare vill jag ha en startsida, för att kunna navigera i appen.
Funktionellt:

- En första sida dyker upp när appen startas.
- I denna förstasida, kan man navigera till fler sidor, en som sen leder till “spelet”.
- Jag ska förstå vilka sidor jag länkas till.

ID: 05.
Namn: Val av underkategori.
Svårighetsgrad: 5.
Beskrivning: Som användare vill jag kunna välja underkategori till mitt valda ämne för att specificera mitt inlärande.
Funktionellt:

- När jag klickat på matematik och kommer till kategorisidan ska jag kunna se en rad olika kategorier.
- Dessa skall vara addition, subtraktion, multiplikation och division.

Icke funktionellt:

- Dessa kategorier kommer endast upp om jag klickar på matematik innan.

ID: 06.
Namn: Direkresultat.
Svårighetsgrad: 6-7.
Beskrivning: Som en användare vill jag få direkt respons efter val av svarsalternativ för att veta om mitt svar var korrekt eller inte.
Funktionellt:

- Vyn uppdateras dynamiskt vid val av fråga, markerar om rätt eller fel
- Svarsalternativen markeras efter rätt eller fel: rätt: ditt svar->grön, andra grå. fel: ditt svar->röd, rätt svar->grön, andra grå. (mer specifikt på figma).
- Efter svar skall användaren tas till nästa fråga.

ID: 08.

Namn: Avsluta/Pausa i pågående quiz.
Svårighetsgrad: 3.
Beskrivning: Som användare vill jag bekräfta om jag ska avsluta mitt i ett quiz, för att inte råka avsluta av misstag.
Funktionellt:

- När jag klickar bakåt mitt ett quiz skall det komma upp en bekräftelse-vy.
- På denna vy ska det finnas en avsluta-knapp.
- På denna vy skall det också finnas en fortsätta-quiz-knapp.
- När jag klickar på avsluta skall jag komma tillbaka till hem.
- När jag klickar på fortsätt så kommer jag tillbaka där jag var i det pågående quizet.

ID: 10.
Namn: Resultatsida med kopplat resultat.
Svårighetsgrad: 6.
Beskrivning: Som användare vill jag kunna se mitt resultat som x/y efter mitt quiz, för att kunna se hur många rätt jag fick.
Funktionellt:

- Resultatet x/y beror på hur många rätt jag fick på quizet.
- Alltså efter mitt additions-quiz på y antal frågor så kan jag sedan i resultatet se hur många av dessa jag svarade rätt på.

ID: 15.
Namn, Avslutningsbekräftelse.
Svårighetsgrad: 2.
Beskrivning: Som användare vill jag bekräfta om jag vill stänga ner appen, för att inte råka stänga av den av misstag.
Funktionellt:

- När jag klickar bakåt på startskärmen skall det komma upp ett fönster.
- På detta fönster ska det finnas en avsluta- och en fortsätt-knapp.
- När jag klickar på avsluta skall appen stängas ner.
- När jag klickar på fortsätt så stängs fönstret ner.

ID: 09.
Namn: 10 frågor.
Svårighetsgrad: 8-9
Beskrivning: Som användare vill jag kunna spela 10 frågor inom addition för att kunna förbättra mina färdigheter inom denna kategori.
Funktionellt:

- När jag startar mitt additions-quiz kommer jag mötas av 10 frågor inom denna kategori.
- Frågorna har följande struktur (x + y = z) där x och y är slumpade och olika efter varje fråga/quiz.
- När jag är färdig med de 10 frågorna skickas jag vidare till resultat-viewn.

Icke funktionellt:

- Frågorna skall senare istället kunna hämtas från databas/fil, så gör denna framtida ändring möjlig utan större svårigheter.

ID: 12.
Namn: Inställningar.
Svårighetsgrad: 2-3.
Beskrivning: Som en användare vill jag kunna nå en vy där jag kan ändra olika alternativ för ett quiz, detta för att kunna anpassa quiz efter förmåga/intresse.
Funktionellt:

- Det ska finnas en vy från huvudvyn där man kan göra inställningar till spelet.
- En valfri inställning skall implementeras, exempelvis “nattläge”, redigering av textstorlek eller möjlighet för implementering av ledtråd eller svårighetsgrad.

ID: 17.
Namn: Antal frågor.
Svårighetsgrad: 2-3.
Beskrivning: Som användare vill jag kunna välja hur många frågor det är i varje quiz, detta för att få en lagom mängd frågor att besvara.
Funktionellt:

- Användaren skall kunna specificera hur många frågor som ska ingå, inom ett rimligt spann (t.ex. 5-25).
- Quizet ska innehålla så många frågor som användaren specificerar.
- Resultatvyn ska visa hur många frågor som quizet innehåller.

ID: 07.
Namn: Statistikvyn.
Svårighetsgrad: 5-8.
Beskrivning: Som användare vill jag kunna se min historik för att kunna om jag har blivit bättre.
Funktionellt:

- Jag kan se en tabell av tidigare spelade resultat.
- I denna tabell så kan jag se: Resultat och datum.
- Jag kan ta mig till denna sida i appen.
- Jag förstår vilken sida jag är på med hjälp av en titeltext mm.

ID: 18.
Namn: Val av ämne och underkategori på samma sida.
Svårighetsgrad: 8.
Beskrivning: Som användare vill jag kunna välja bland två ämnen och sedan bland några kategorier inom dessa ämnen för att specificera lärandet.
Funktionellt:

- Det ska finnas två ämnen: Matematik och Historia.
- Matematik ska vara kopplat till Addition, Subtraktion, Multiplikation och Division.
- Historia i sin tur skall vara kopplat till "Europas historia" och "Sveriges historia".

Icke funktionellt:

- Det ska finnas koppling mellan kategori och quiz. Så att man exempelvis kan visa passande quiz utifrån vad för kategori man valt.
- Det ska finnas hyfsad enkel möjlighet att bygga ut detta i "framtiden", alltså att lägga till nya kategorier och ämnen skall gå att göra ganska enkelt i koden.

ID: 20.
Namn: Spellägen.
Svårighetsgrad: 8.
Beskrivning: Som användare vill jag kunna välja ett annat spelläge för att göra mitt lärande roligare.
Funktionellt:

- Jag vill kunna välja ett spelläge där jag har tre liv. Jag tappar ett liv varje gång jag svarar fel och när jag har slut på liv så avslutas quizet.
- Jag ska kunna välja spelläget under inställningar.

ID: 24.
Namn: Tidpress som spelläge.
Svårighetsgrad: 5-6.
Beskrivning: Som användare vill jag kunna ha ett spelläge där jag får n tid på mig att besvara x antal frågor för att göra quizet mer spännande.
Funktionellt:

- Användaren ska få ett visst antal sekunder på sig att besvara så många frågor som den hinner med.
- Tid ska gå då användaren inte har besvarat en fråga.
- Vid svar och väntan på nästa fråga ska tiden vara pausad.
- Användaren ska kunna specificera antalet sekunder i inställningar.

ID: 23.
Namn: Se hur många frågor man har kvar inne i ett quiz.
Svårighetsgrad: 1.
Beskrivning: Som användare vill jag kunna se vilken fråga jag är på och hur många jag har kvar väl inne i ett quiz, så att jag kan lokalisera mig inne i ett pågående quiz.
Funktionellt:

- Inne i ett quiz så skall det finnas en text där det står vilken fråga man är på och hur många frågor quizet innehållet.
- Texten skall vara i formen “Fråga x av y”.

ID: 19.
Namn: Logga in användare.
Svårighetsgrad: 7.
Beskrivning: Som användare vill jag ha ett eget konto för att appen skall kännas personlig.
Funktionellt:

- Det ska finnas en inloggningssida.
- Här ska det finnas "Användarnamn"- , "Lösenord"-fält och en "Logga in"-knapp.
- Man kan skapa ett konto.

Icke funktionellt:

- Det ska finnas fördefinierade användare.
- Inloggning ska ske genom en service som endast exponerar ett gränssnitt och en fabrik.
- Den aktiva användaren kan man komma åt genom en publik Singleton.
- Lösenord behöver ej krypteras utan kan lagras/skickas som en vanliga sträng i nuläget.

ID: 22.
Namn: Fler quiz än endast addition
Svårighetsgrad: 6
Beskrivning: Som användare vill jag kunna spela fler quiz än addition för att kunna plugga på just det jag behöver.
Funktionellt:

- Kunna spela addition, subtraktion, multiplikation, division.
- Kunna spela Sveriges historia och Europas historia.
- kunna spela ett user-quiz.

Icke funktionellt:

- Ingen databas just nu.
- Användarquizet är i nuläget hårdkodat.

ID: 11
Namn: Ledtråd
Svårighetsgrad: 5.
Beskrivning: Som en användare vill jag kunna få en ledtråd om jag inte kan besvara en svår fråga, detta för att jag inte ska bli frustrerad.
Funktionellt:

- Det ska finnas en knapp på quiz-sidan som man kan trycka på för att få hjälp.
- Vid tryck skall svar gråas ut ifall de är fel, max två st ledtrådar kan användas per fråga.
- Kunna ta bort möjlighet till ledtråd i inställningar.

ID: 25.
Namn: “Använt ledtråd” syns i ditt sparade resultat.
Svårighetsgrad: 1-2.
Beskrivning: Som användare vill jag se om mitt sparade resultat använde ledtråd, så jag kan jämföra hur mycket bättre jag har blivit.
Funktionellt:

- Mitt sparade resultat ska markeras om jag har använt ledtråd.
- Denna markering skall synas i resultatvyn och i statistiksidan.
- Markeringen kan exempelvis vara en liten text eller en stjärna.

2.2 Icke genomförda användarberättelser

Nedan listas de användarberättelser som ej genomfördes under detta projekt. De är listade i prioritet, från högst till lägst prioritet.

Om ytterligare en sprint hade varit möjlig så hade alltså databaser implementeras. Detta hade gett ett stort lyft av applikationen, då information som exempelvis statistik för en användare hade sparats ner och varit tillgängligt även efter omstart av applikationen. Det hade gjort applikationen mer verklighetstrogen och komfortabel för en användare.

Senare hade skapandet av egna quiz med egna frågor implementerats på riktigt. I nuläget är koden anpassad för att detta skall vara lätt verkställbart men det skulle krävas ytterligare en sprint för att få det fullständigt implementerat och redo att presenteras som en funktion i applikationen.

Den sista användarberättelsen handlar om att implementera en möjlighet att ändra svårighetsgraden på frågorna. Detta är en spännande funktion men som ej är nödvändig för applikationen i nuläget, därav dess låga prioritet.

ID: 21.
Namn: Implementering av Databaser.
Svårighetsgrad: 8
Beskrivning: Som användare vill jag att min historik och användare sparas så att jag kan komma tillbaka till denna information senare, exempelvis när jag stänger ner appen och startar igen.
Funktionellt:

- Implementera databas till användaren, så att historiken sparas ner.
- Implementera databas så att användarens egna kategorier och quiz kan sparas.

Icke funktionellt:

- Room databas skall användas.

ID: 13.
Namn: Skapa eget quiz.
Svårighetsgrad: 5
Beskrivning: Som användare vill jag kunna skapa eget quiz, för att kunna plugga på exakt det jag vill.
Funktionellt:

- Jag kan se en knapp.
- kan t.ex. stå “skapa eget quiz” på den.
- Ifall jag trycker på knappen så skapas ett quiz.

ID: 14.
Namn: Lägga till frågor i eget quiz.
Svårighetsgrad: 7-8.

Beskrivning: Som användare vill jag kunna lägga till egna frågor i mitt egna quiz, för att kunna specificera de frågor som jag vill träna på.
Funktionellt:

- Jag kan se en knapp.
- kan t.ex. st “Lägg till fråga/alternativ” på den.
- Ifall jag trycker på knappen så skapas en fråga i quizet och jag kan skriva in en fråga och tillhörande alternativ.

ID: 16.
Namn: Svårighetsgrad av quiz.
Svårighetsgrad: 4-5.
Beskrivning: Som användare vill jag kunna bestämma svårighetsgrad på mitt valda quiz för att utmana mig själv.
Funktionellt:

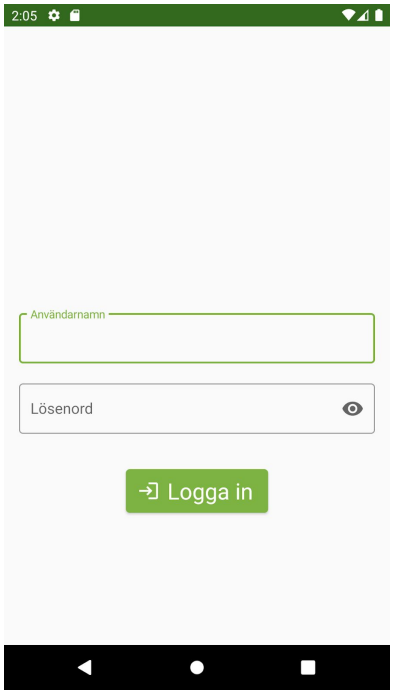
- Vid val av kategori ska användaren kunna välja svårighetsgrad.
- Valet av svårighetsgrad ska speglas i de frågor som används i quizet

2.3 Definition of Done

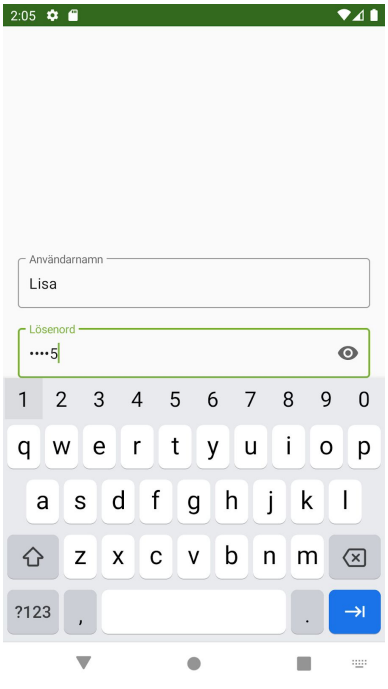
Följande punkter skall vara uppfyllda innan en användarberättelse kan räknas som genomförd.

- Personen som har fått användarberättelsen ska anse att den är klar.
- En annan gruppmedlem ska kolla igenom och testköra koden. Denna gruppmedlem ska också anse att användarberättelsen är klar.
- Det ska stå författare på nyskapade dokument.
- Koden ska vara lagom kommenterad. Publika klasser bör vara kommenterade.
- Modellen ska vara testad till hög grad.

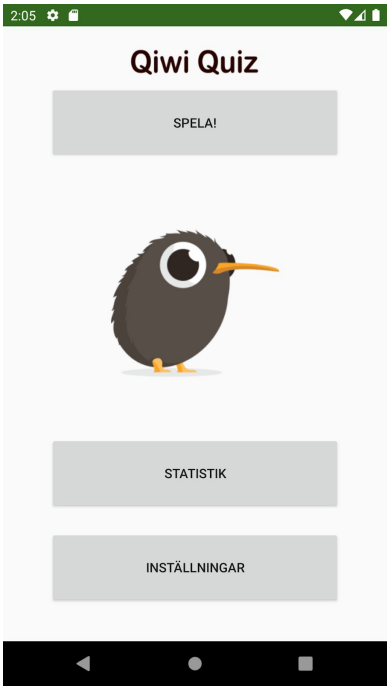
2.4 Användargränssnitt



Figur 1



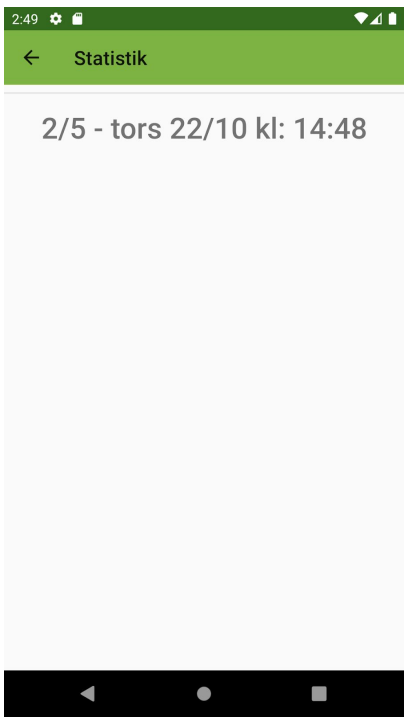
Figur 2



Figur 3

Applikationen öppnas på denna inloggningssida (Fig 1). Användaren tvingas att skriva in ett användarnamn och ett lösenord för att gå vidare. Detta görs för att kunna koppla en användare till varje spelomgång. I det nuvarande stadiet kan man skriva in valbart namn och lösenord utan att förlora åtkomst till sidan—men tanken är att denna sida representerar en korrekt utförd och användbar login-sida, med den framtida möjligheten att skapa user och logga in för att nå specifika användarsidor och funktioner. Det finns ett sparat konto, som är nåbart från inloggningssidan: användarnamn ‘Lisa’, och Lösenord ‘12345’. Genom detta konto får man tillgång till ett eget skapat quiz och tidigare historik. Om man loggar in utan Lisas konto så är ens historik och användar-quiz tomma. Inmatning för loggin följer (Fig 2) och är ett klassiskt inmatningsschema utan överraskningar, vilket borde bara tydligt, förståeligt, och användarvänligt.

När man har loggat in så kommer man till den tänkta startsidan (Fig 3). På denna sida syns applikationens titel “Qiwi Quiz” överst. Under titlen finns “Spela!” knappen—denna knapp leder till själva spelmekaniken. Under “Spela!” kommer spelets logga, en Kiwifågel som är en referens till applikationens titel. Dom två sista knapparna på startsidan är “Statistik” och “Inställningar”. Den förstnämnda knappen leder till en statistiksida, där spelhistorik visas i en upprepande horisontell lista, se (Fig 4-5). Denna historik följer formen (poäng/möjliga-poäng - veckodag dag/månad kl: tt:mm (* hint symbol)). Sorteringen av listan är så att den visar den senaste statistiken överst.

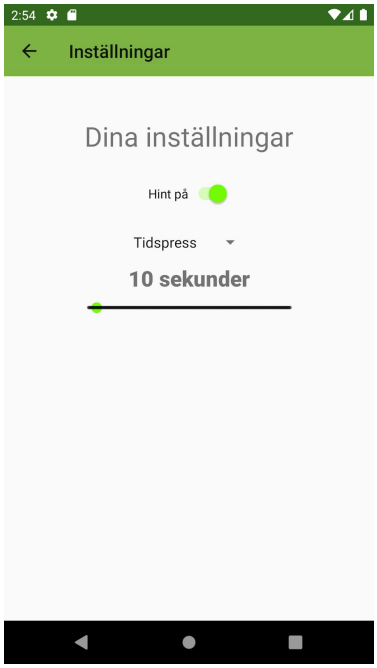


Figur 4

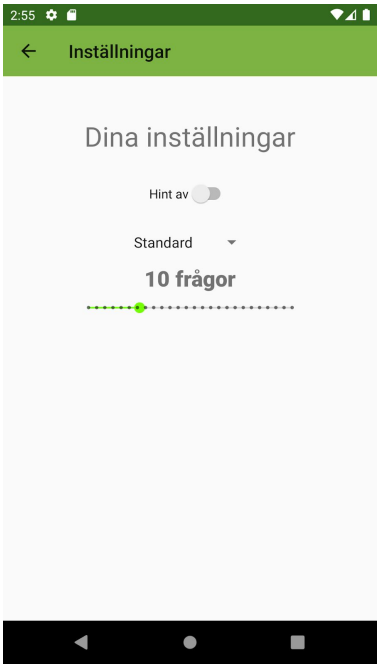
Figur 5

Figur 6

När det är en (*) symbol (se Fig 6), i slutet av ett historik resultat, så betyder det att den spelomgången använde sig av hints. Spelresultaten som använder sig av hints, och dom som inte, separeras för att användaren ska kunna se om sin möjliga förbättring inte är baserad på användandet av hints.



Figur 7



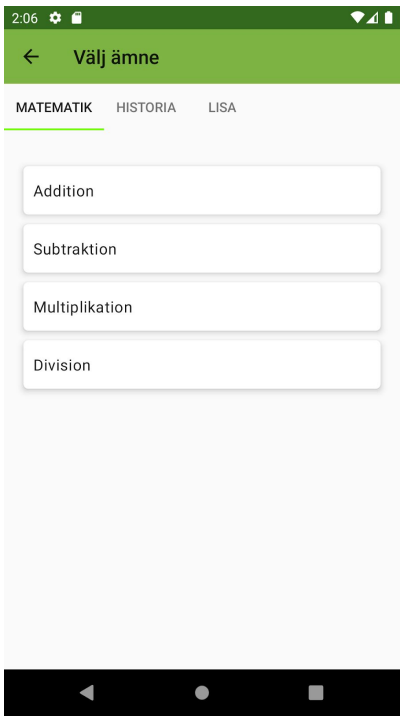
Figur 8

Figur 9

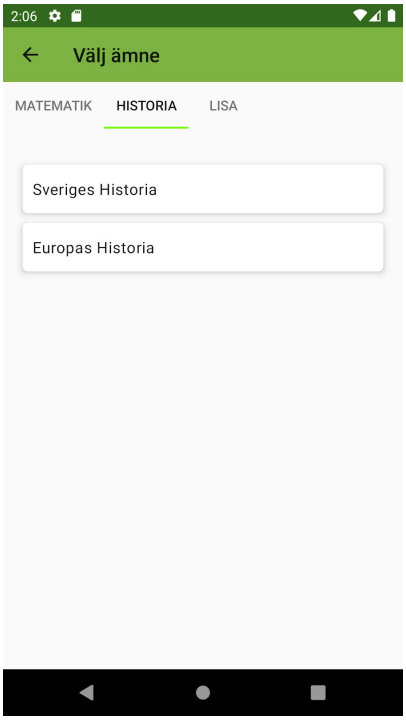
Om användaren, från startsidan (se Fig 3), klickar på “Inställningar” så öppnas sidan som syns i dom övre figurerna (Fig 7-9). Under inställningssidan kan användaren stänga av och på hints med en toggle knapp, byta mellan tre olika spellägen med en kombinationsruta, och—när tillämpbart—ställa in specifika inställningar för spellägenerna. Spellägen är olika konfigurationer för hur quiz spelas och uppfattas. Bilder och ytterligare förklaring på spellägena finns på (s.15,17-18).

Dom tre spellägena som är implementerade är: “Tidspress” (se Fig 7) där användaren väljer hur lång tid användaren har på sig att genomföra sitt valda quiz, “Standard” (se Fig 8) där hen väljer hur många frågor som kommer presenteras under quizet, och “Max tre fel” (se Fig 9) där användaren har tre liv på sig innan quizet avslutas.

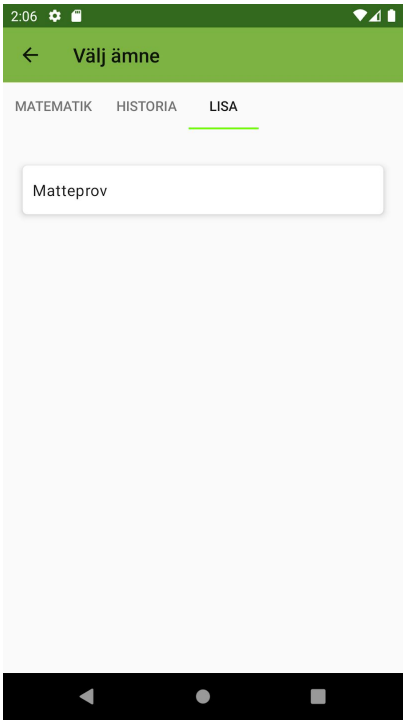
Hint är en funktion som tillåter användaren att tydligt inaktivera upptill två svarsalternativ i ett pågående quiz. Hint sätts på och av med en beskrivande och tydlig toggle knapp. Den är på i (Fig 7), av i (Fig 8-9). När hint togglen är av så syns ingen hint knapp i den nedre delen av ett pågående quiz (Fig 16). Men när hint togglen är på så finns det en synlig klickbar hint knapp (Fig 13).



Figur 10

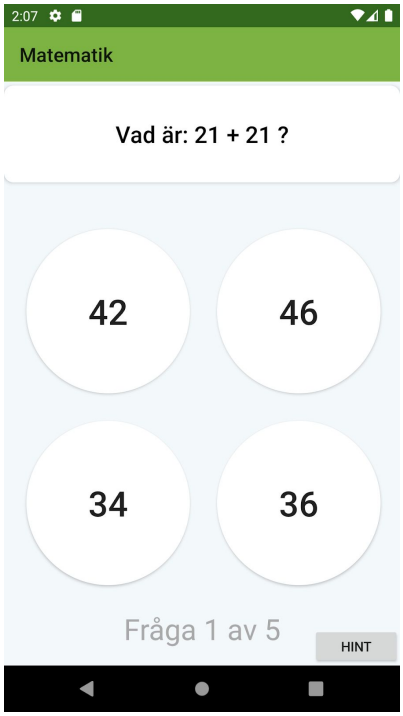


Figur 11

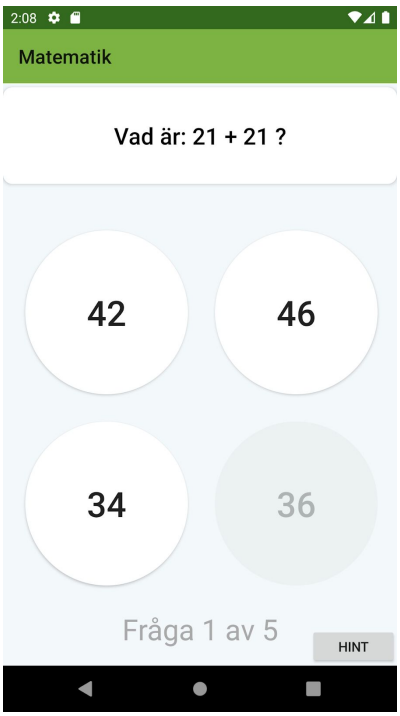


Figur 12

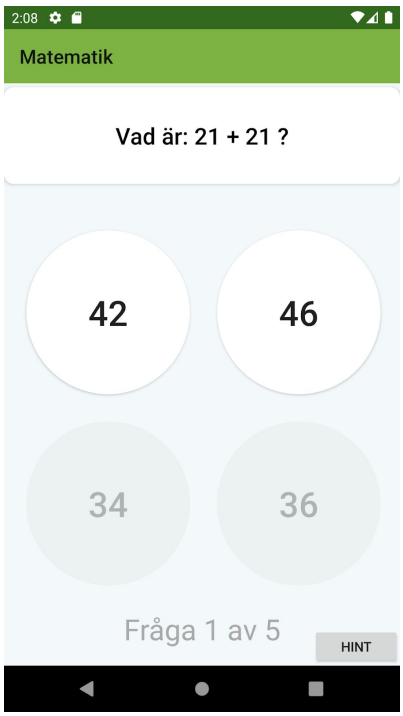
Om användaren klickar på “Spela!” så leds dom till “välj ämne” sidan (se Fig 10-12). Här väljs quizets ämne, så som Matematik, Historia och User-quiz (som i detta fallet heter Lisa efter användaren). Detta övergripande ämne väljs med en ‘tabb-meny’ där hen kan klicka eller svepa med fingret mellan dom olika undersidorna. Efter ett val av övergripande ämnen väljer användaren en av flera underkategorier. Om ‘Addition’ väljs så startar ett additions quiz med matematik frågor: till exempel “15+42?” (se Fig 13). Frågorna kopplade till detta quiz system kan antingen vara genererade eller sparade frågor.



Figur 13



Figur 14

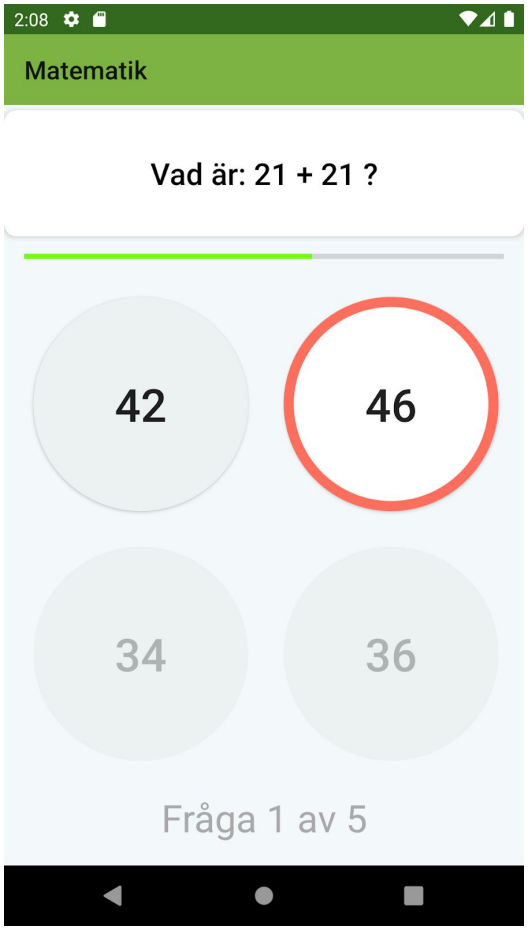


Figur 15

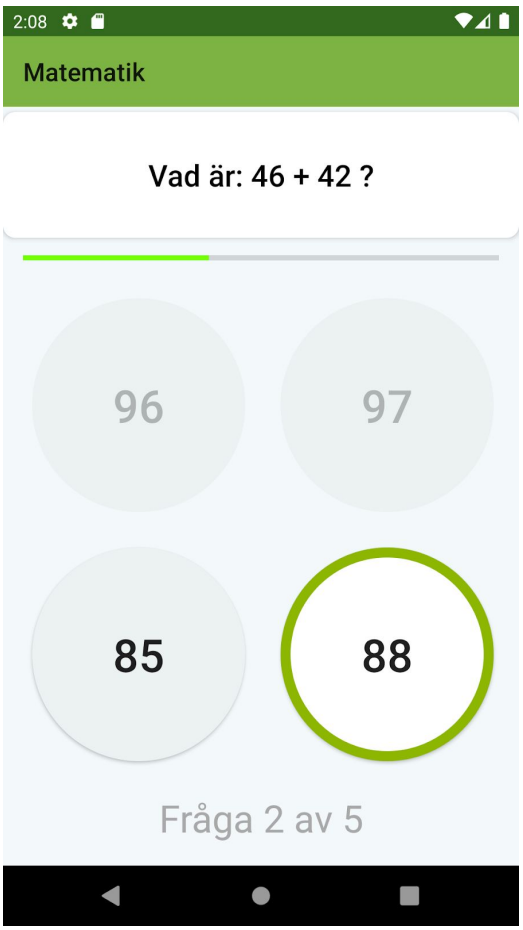
(Fig 13) är hur ett ‘standard’ quiz ser ut, med hint på, och ämnet ‘addition’. Om användaren klickar på hint knappen så blir ett felaktigt alternativ grått och oklickbart (se Fig 14). Detta kan göras två gånger per quiz fråga (se Fig 15), efter två alternativ är gråa så kan inte hint knappen längre klickas eller interageras med.

I ‘standard’ quiz spelläget är antalet frågor satta i förväg av användaren, eller lämnat på dess default. Antalet frågor och vilken fråga användaren är på syns längst ner på sidan (se Fig 13).

I alla quiz spellägen och ämnen så är frågan längst upp och fyra svarsalternativ under. Det som ändras med olika ämnen är titeltexten (gröna partier i Fig 13) och vilken typ av frågor/alternativ.

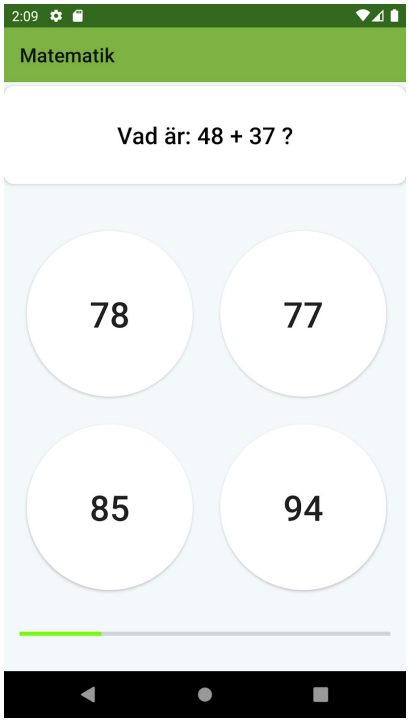


Figur 16

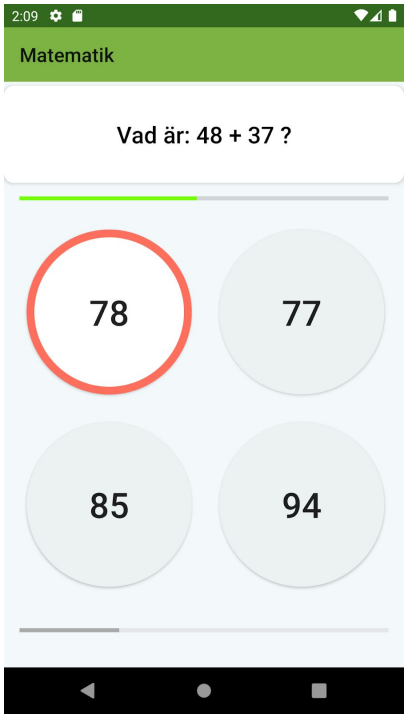


Figur 17

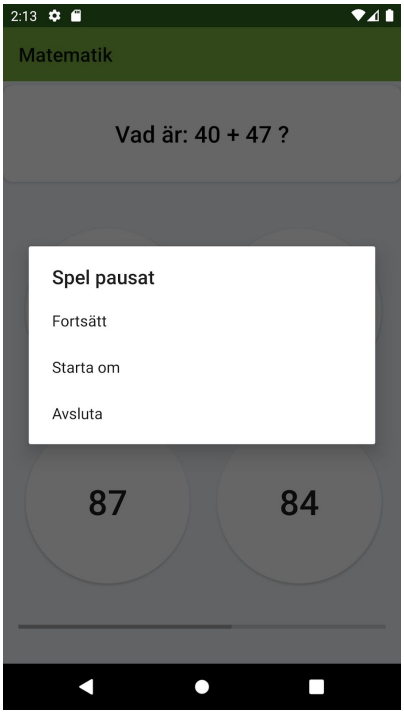
(Fig 16-17) Visar hur sidan reagerar när man klickar på felaktiga alternativ (Fig 16) eller det korrekta alternativet (Fig 17). Alla quiz, i vår nuvarande prototyp, har tre felaktiga alternativ och ett korrekt alternativ. En grön nedräknings linje syns under frågan som signalerar när nästa fråga dyker upp eller, vid ett avklarat quiz, när resultatsidan kommer visas.



Figur 18



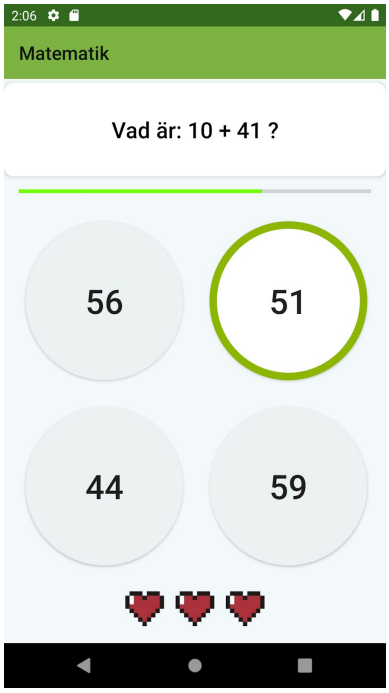
Figur 19



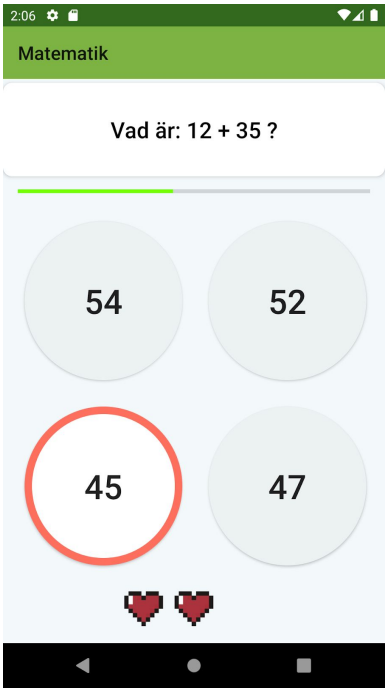
Figur 20

I (Fig 18-19) visas spelläget ‘Tidspress’. Detta spelläge begränsar användarens speltid, istället för att begränsa frågor eller liv/chanser. Nedräkningen syns av den nedre gröna linjen (se Fig 18) men den pausas när man har svarat och tills nästa fråga (se Fig 19). När den gröna linjen är full så avslutas spelomgången.

Om användaren, mitt i ett quiz, klickar på en bakåtknapp, till exempel pilen i nedre vänstra hörnet av appen, så dyker ett popup fönster upp (se Fig 20). Detta popup fönster tillåter användaren att navigera tillbaka till startsidan med “Avsluta”, börja om samma typ av quiz med “Starta om”, och klicka ner popup-fönstret med “Fortsätt” för att fortsätta med samma quiz som redan var påbörjat.



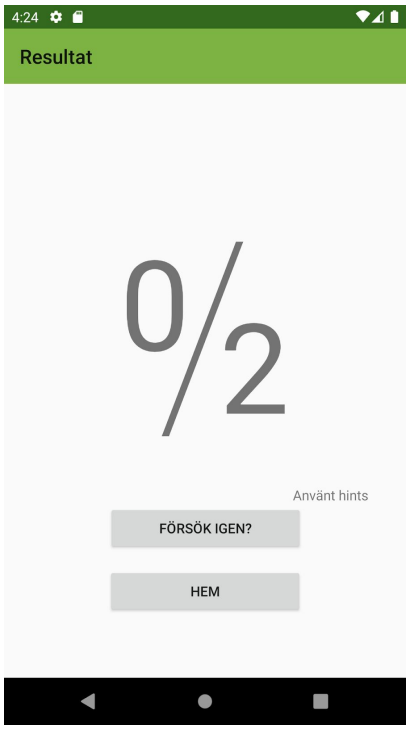
Figur 21



Figur 22

Figur 23

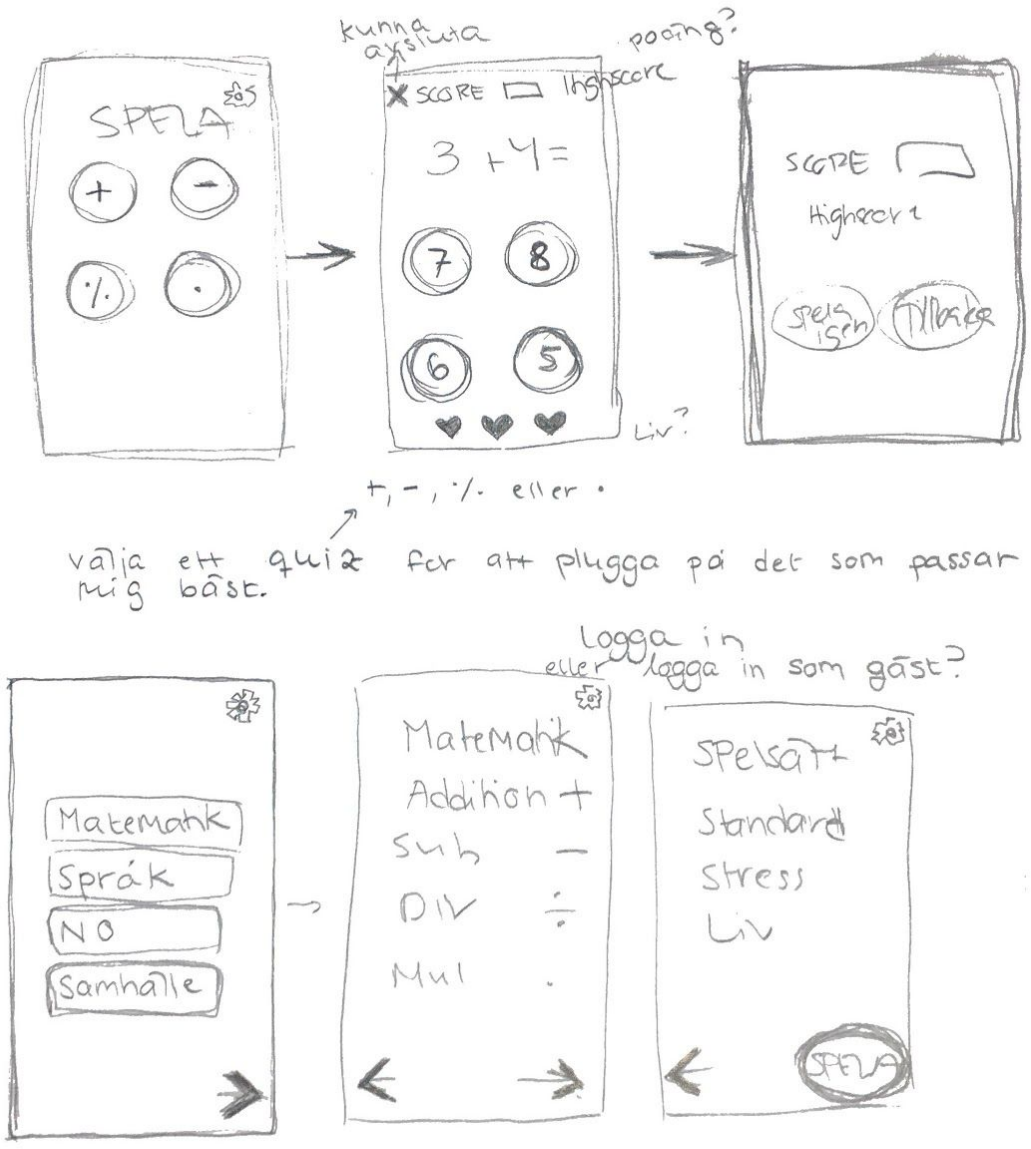
I “Max tre fel“ spelläget är användaren begränsad av liv. Spelläget ger tre chanser att ha fel, innan spelet avslutas och går vidare till resultatsidan. Liven illustreras av hjärtan (se Fig 21) som försvinner en och en om man svarar fel (se Fig 22).



Figur 24

Resultatsidan (se Fig 23) presenterar användaren med hur många rätt och fel hen fick i det tidigare quizet. Därifrån kan hen “försöka igen?” vilket kör samma typ av quiz igen, eller så kan dom klicka “Hem” vilket öppnar startsidan igen. Resultatsidan visar även, med en liten text-påminnelse, om användaren har använt hints i deras senaste spelomgång (se Fig 24).

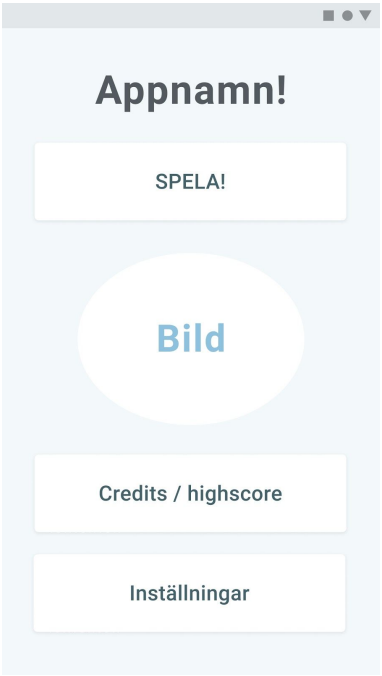
Prototyp Designen



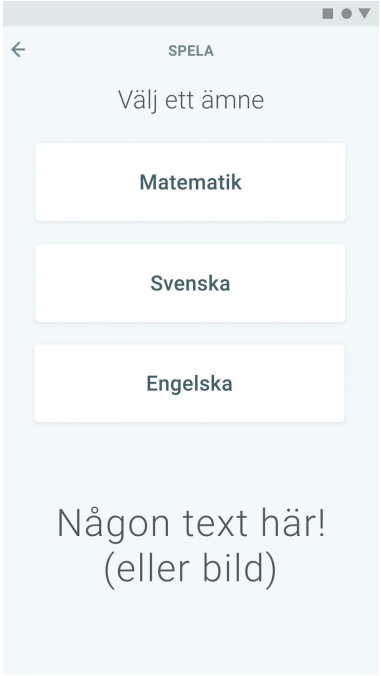
Figur 25

Den första prototypen, (se Fig 25), var en pappersskiss med många av samma funktioner och sidor som vi slutligen byggde i vår android applikation. Några likheter var spellägen, funktionen ‘val av ämne’ och quizets själva design och upplägg.

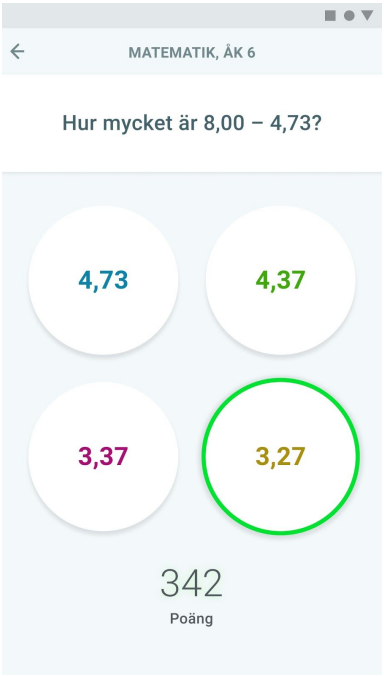
Nästa steg var en Figma design prototyp (se Fig 26-29) som någorlunda stod för designvalen och dom slutliga sidorna som applikationen använder sig av. Designen skiljer sig åt då, efter råd av handledare, designen låg-prioriterades.



Figur 26



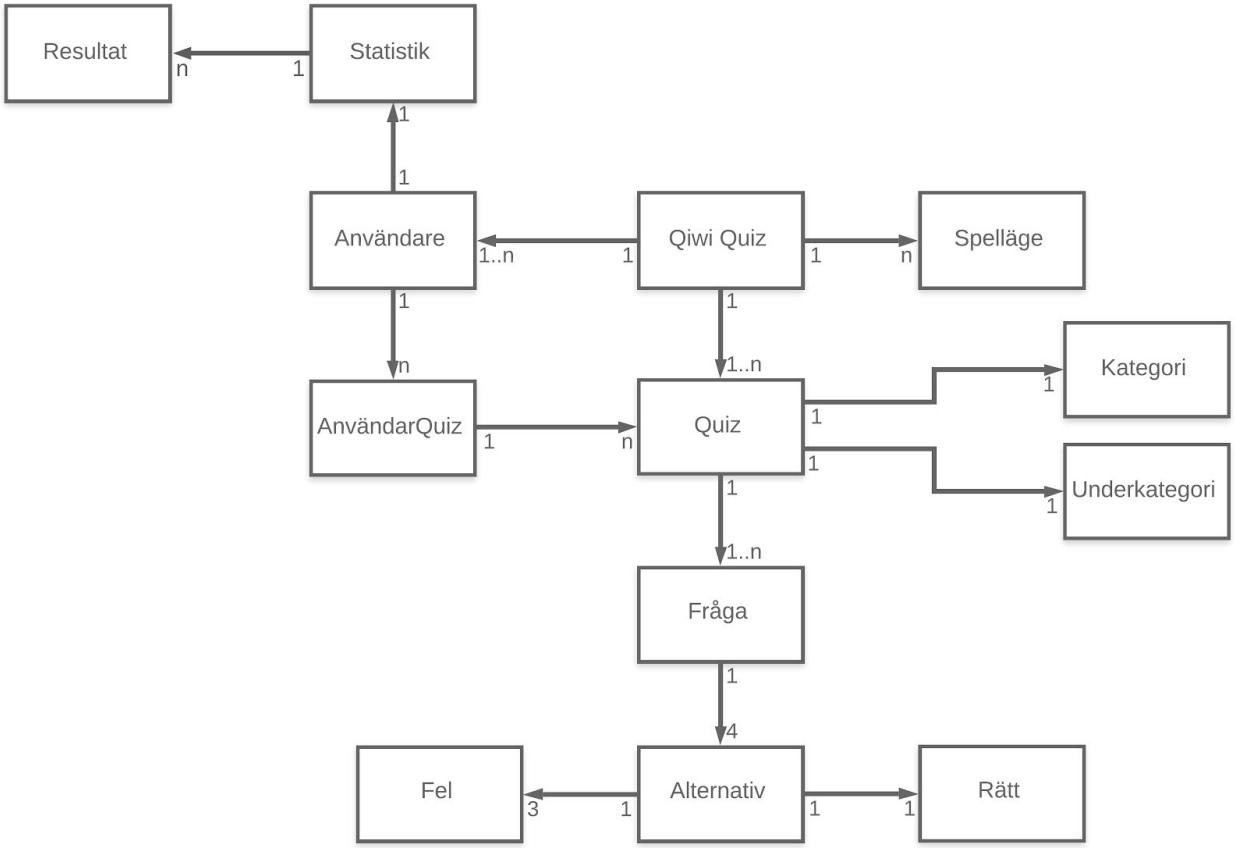
Figur 27



Figur 28

Figur 29

3. Domänmodell



Figur 30: Domänmodell för Qiwi Quiz.

3.1 Ansvarsområden

Nedan listas ansvarsområdena för varje del av domänmodellen (figur 30).

Qiwi Quiz:

- Denna del speglar kärnan av applikationen. Den håller ihop programmet och står för mycket av funktionaliteten i modellen.

Quiz:

- Huvudsyftet med applikationen är att kunna spela ett quiz. Varje quiz innehåller flera olika frågor varav varje fråga har en frågeställning och fyra svarsalternativ. Av dessa alternativ är ett rätt och resterande tre alternativ är felaktiga.

Kategori och underkategori:

- Varje enskilt quiz har en kategori och en underkategori för att kunna organiseras och lokaliseras. Ett exempel är ett quiz med kategorin “matematik” och underkategorin “addition”.

Spelläge:

- Denna del står för de olika spellägena som applikationen erbjuder. Deras huvudfokus är att göra lärandet roligare. Exempel på ett spelläge är “tre liv”, vilket är ett överlevnadsläge där användaren endast får svara fel tre gånger.

Användare:

- Användaren har tillgång till en personlig statistik som i sin tur är en lista av tidigare quizresultat.

AnvändarQuiz:

- Varje användare har även tillgång till personliga quiz som de lägger in på egen hand. Dessa användarquiz har samma skelett som de redan erbjudna quizen, med undantag att kategori, subkategori och frågorna i sig är personliga.

4. Referenser

[1] Ulf Eriksson, Christer Mattson. (2009-10-19). Från användarberättelse till färdigtestat system, [Hemsida], Länk: https://konsultbolag1.se/bloggen/fraan-anvaendarberaettelser-till-faerdigtestat-system (Besökt 2020-10-21).

[2] Lars Danielsson. (2015-11-01). Vad är det folk pratar om när folk pratar om Scrum?. [Hemsida], Länk: https://techworld.idg.se/2.2524/1.640036/scrum-norm-for-systemutveckling (Besökt 2020-10-22)

System Design Document for Qiwi Quiz

Carl Bergman, Erik Blomberg, Henrik Johansson,
Sara Persson, Louise Tranborg

October 23, 2020
Version 2

1 Introduction

Qiwi Quiz is a quiz application for Android phones. It is directed primarily towards Swedish children enrolled in elementary school. The questions are collected from the different subjects that they study, for example: mathematics and history. All questions use multiple alternatives and there is always one, and only one, choice that is correct.

While developing the application the most important goal was to plan ahead to ease development and make sure it is "future proof". Implementing additional features took a back seat to ensuring that the application remained extensible, maintainable and reusable, in other words "future proof". As long as the application follows these criteria, there will be no trouble to implement more than a thousand features and it will be possible to develop and maintain the application for all eternity.

This document describes the functionality and displays relations between different parts of the application. Furthermore, it reports how the quality of the code was ensured, and also how the application can be improved in the future.

1.1 Definitions, acronyms, and abbreviations

- **Activity** — Our application is built upon Activities, which is a hybrid between a View and a Controller. Every new section of the application is a new Activity, it presents a designated task for the user.
- **Android** — An operating system developed by Google for mobiles.
- **Design Pattern** — Solutions for common programming design problems.
- **GUI** — Graphical User Interface.
- **MVC** — Divide the program into three separate parts, Model, View and Controller. Used to avoid potentially dangerous dependencies.
- **Service** — General term for a modular package which is responsible for reading and storing data, works without the rest of the model knowing how data is retrieved/stored.
- **UML** — Unified Modeling Language.

2 System Architecture

2.1 Overview

Subjects are used to categorize the questions, they are used as a primary key to request questions from a service. Questions in the application are created by a uniform Question Interface, which integrates the ability to get a question text and a possibly unlimited number of alternatives. Questions store their alternatives in a static form, thus patterns can easily be found in the question alternatives. To combat this, the order that the questions appear, and their respective alternatives, are randomized before they are presented on the screen. The game logic, and how the questions are presented, differ depending on the chosen game mode. By contrast, the base gameplay—choosing the correct alternative in response to a question—does not depend on the current game mode.

2.2 Program Flow

The program launches into a title screen where the user can choose between three options, where two of them are: *High Score* and *Settings*. The *High Score* menu is used to view previous quiz results.The *Settings* menu is used to customize certain features to the players liking. The third option is *Play*. Upon selecting this option, the application will advance to another screen presenting a list of categories with subcategories, such as the category “Maths” with the subcategories “Addition” and “Subtraction”. Selecting a subcategory will result in the game starting.

During the game, the user is presented with a question and four alternatives to choose from. The player must now choose the correct alternative, which can be several or just one. After an alternative has been chosen, the alternatives will reveal whether they are correct or wrong. Depending on game mode, the player might, for example, be awarded points upon choosing the correct answer or loose time when selecting an incorrect answer. The process is now repeated until all questions have been answered, a result view is then displayed, where the user can view a summary of the amount of correct answers given. The results are stored and the player can now choose to play again or return to the title screen.

3 System Design

3.1 Overview

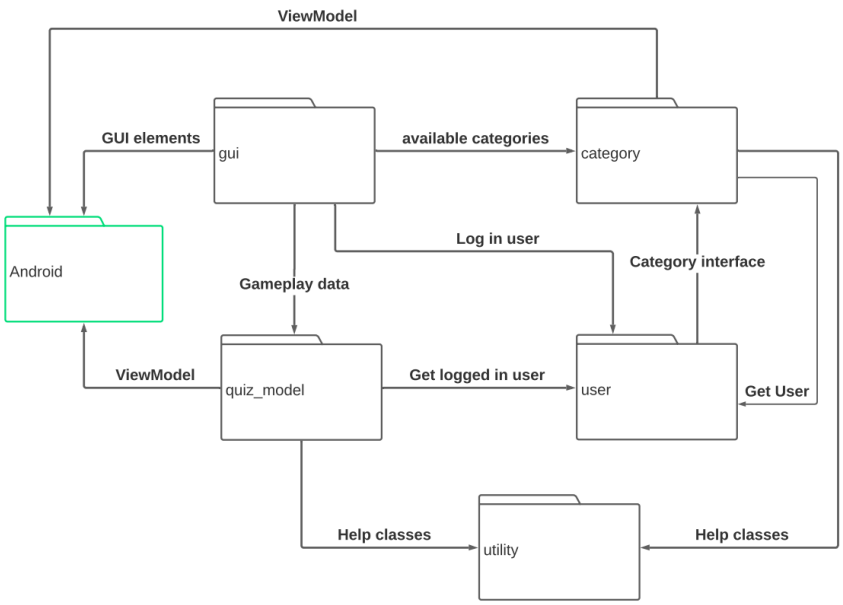


Figure 1: The top-level diagram, identifying the core packages

3.2 MVC Architecture

The program utilizes Android's recommended architecture, which is similar to standard MVC[1]. In Android, the *View* and *Controller* part is handled by `Activity` classes. These classes bind an existing XML layout and can then respond accordingly to user interactions. The *Model* part is handled by so called `ViewModel` classes, which can be used by `Activity` classes. The benefit for using this approach is for example when the user rotates the screen (ultimately destroying the `Activity`) all data is still intact, because `ViewModels` survive configuration changes. In this case the **gui**-package has all visual components such as `Activity` classes, **quiz_model**-package and **category**-package provide `ViewModel` classes.

3.3 Packages

Below are all packages shown in more detail, except for `gui` and `Android`. Subpackages and classes colored green implement components from the `Android` namespace. For a more comprehensive look, view the following diagrams: GitHub

3.3.1 Category

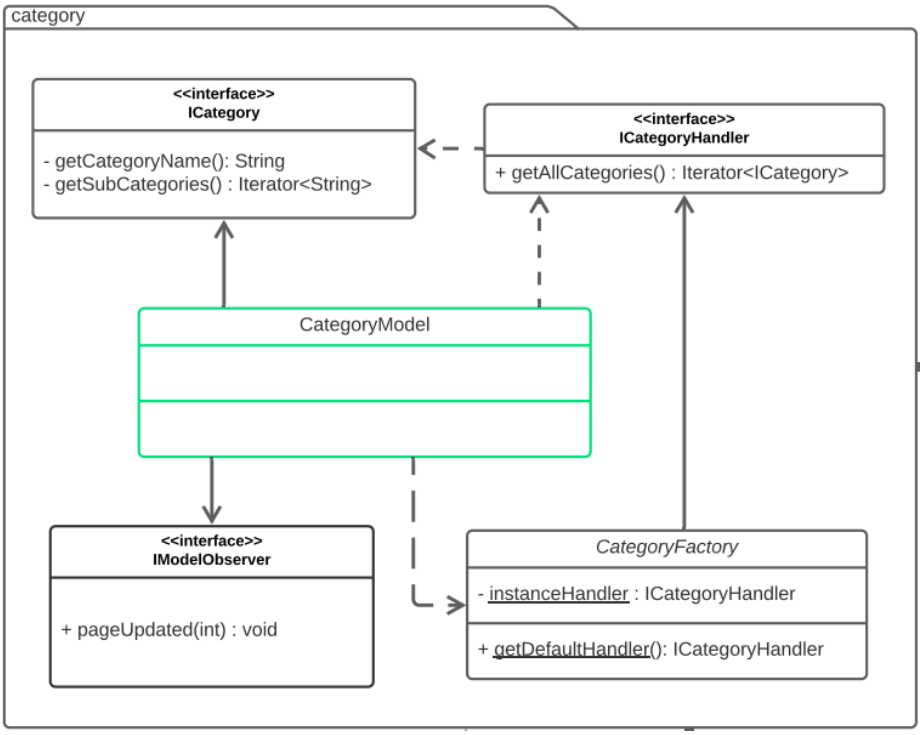


Figure 2: UML diagram showcasing the category package

The essential interface is `ICategory`, which represents a category and it's respetive subcategories. By combining this with the `CategoryModel`, which inherits the `ViewModel` class, the GUI can now showcase categories from multiple sources in a standardized way. The `CategoryHandler` interface acts as a uniform way to access a collection of categories no matter how they are gathered or stored.

3.3.2 User

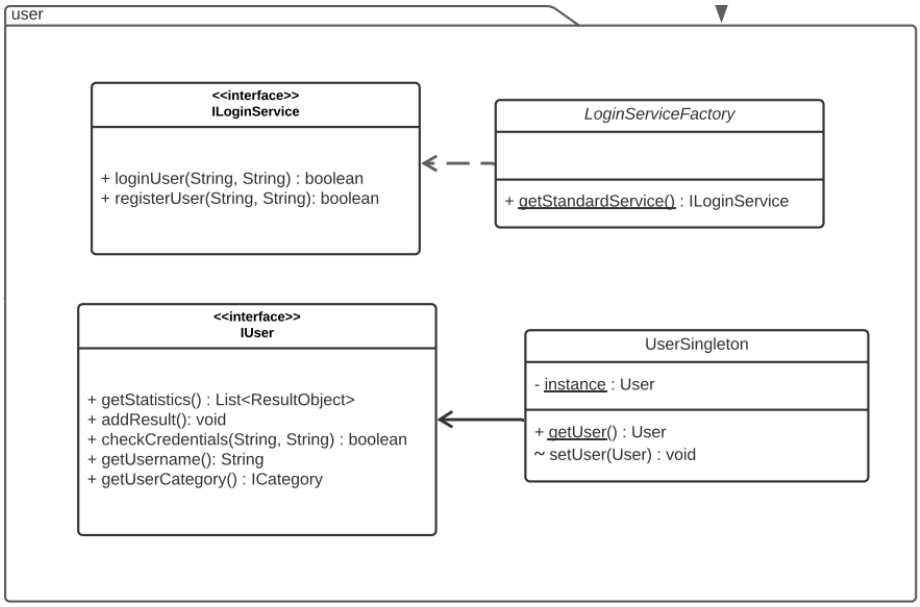


Figure 3: UML diagram showcasing the user package

The `ILoginService` interface is used to log in or register user, how the actually authentication process is conducted is hidden. After a successful authentication process the user is represented as an `IUser`, which enables statistics or custom categories as `ICategory` (see **Category**) to be stored. To make sure only one user is logged in and an easy access point for the rest of the application, an `UserSingleton` is implemented.

3.3.3 Utility

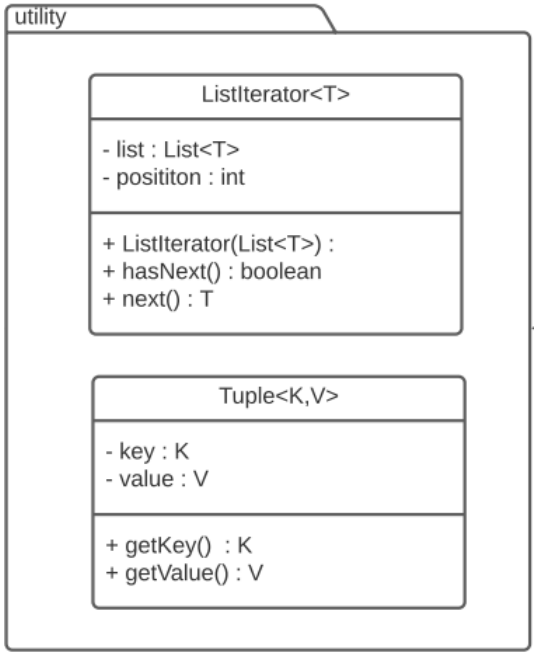


Figure 4: UML diagram showcasing the utility package

Only two classes in this package, however they act as general solution used throughout the model. The `Tuple` class is for example used to store question alternatives, a text component and Boolean to determine it's validity. The `ListIterator` class inherits the `Iterator` interface[2] and is similar to the iterators supplied by the `List` interface[3], however it has one crucial difference. The underlying list can change while another thread iterates over the collection without causing an run-time error[4].

3.3.4 Quiz Model

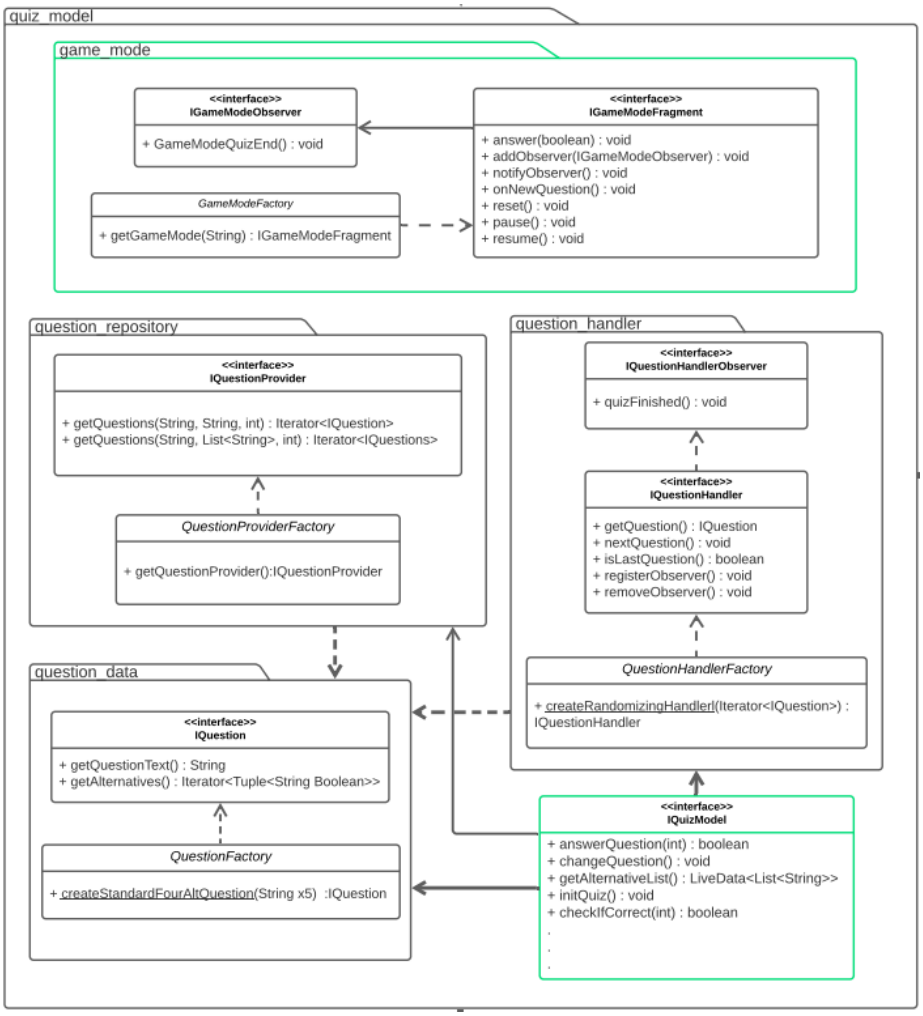


Figure 5: UML diagram showcasing the quiz model package

The core is the `IQuizModel` interface, which ties together all components. Subtypes of `IQuizModel` implements the `ViewModel` class, making it accessible by the GUI. All questions are defined by the `IQuestion` interface. The `IQuestionProvider` interface is utilized to fetch questions for the `IQuizModel`, by providing a category, subcategory(s) and the number of questions. `IQuestionProvider` was purposely designed to hide the internal details of how questions are stored and retrieved, to enable future expansions. Lastly, the `IQuestionHandler` interface is responsible to store and keep track of the questions during the game, handing out for example questions with

randomized alternatives.

3.4 Design Patterns

- **Iterator Pattern** — Used to future proof the application, if a data-structure changes, then the rest of the application is not affected. Can be seen in the file `ICategoryHandler` in **Category** and `IQuestionProvider` in **Quiz Model**.
- **Factory Pattern** — This pattern was used to promote loose coupled design, hiding the instantiating part of concrete classes. Can be seen in the **User** package and the **Category** package, specifically `LoginServiceFactory` and `CategoryFactory` respectively.
- **Facade Pattern** — Used to simplify and hide internal complexity of a module. Used together with factory pattern, to create services which can be seen in **User**, `ILoginService`, but also in **Quiz Model**, `IQuestionProvider`.
- **Observer Pattern** — A pattern utilized to notify 'listeners' about changes. Used with `IModelObserver` in **Category** package, `IQuestionHandlerObserver` in **Quiz Model** package, among other occasions in the application.
- **Singleton Pattern** — Utilized to make sure that only one instance can be created. Used in **User** package and only with the file `UserSingleton`.

3.5 Domain and design model

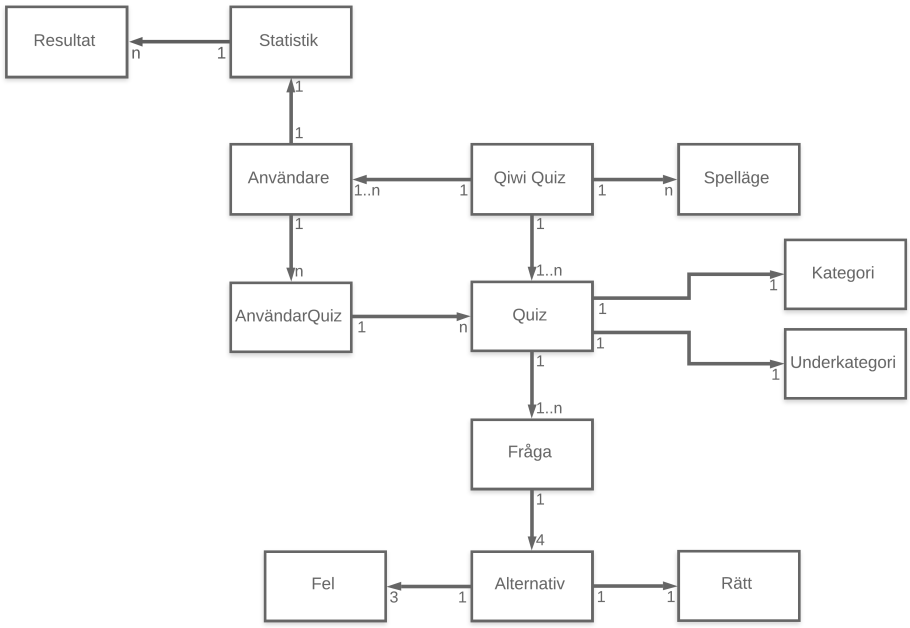


Figure 6: Domain model of the application

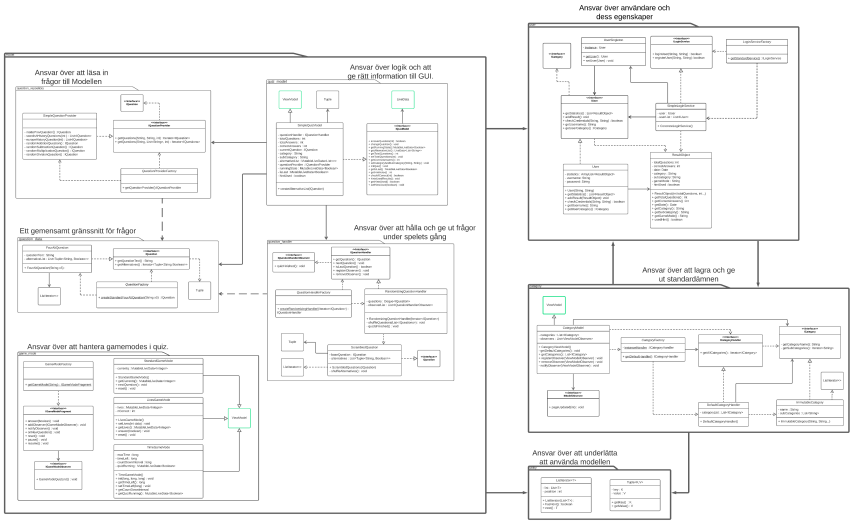


Figure 7: Design model of the application

The relation between the domain model and the design model of the application is clearly visible by looking at "Qiwi Quiz" in the domain model and the `SimpleQuizModel` class in the design model. We can see that "Qiwi Quiz" has at least one Quiz. These quizzes are represented by `RandomizingQuestionHandler` in the design model, which has a Deque with questions. These questions, in turn, contain a list with four alternatives, where one is correct and the rest are incorrect — directly corresponding to the domain model.

Additionally we can see that "Quiz" in the domain model has a "Category" and a "Subcategory", these are represented by 2 strings within the `SimpleQuizModel` class. "Qiwi Quiz" also has a user, represented by a User object accessible via a service. These users has statistics, consisting of a list of Result objects. In the domain model there are "n" amount of Results, which matches with the list that can be empty or filled with n-amount of results. The "User" also has "UserQuiz"s, these quizzes work the same way as the regular quizzes explained above.

Lastly, in the domain model there is also GameMode, represented by the `game_mode` package in the design model. Although not directly working with `SimpleQuizModel`, it is used by the Activities that present the quiz to the user.

4 Persistent data management

The application makes some use of persistent data, there are however occasions where data is not saved in a way that makes it remain on application restart. This is not a conscious choice but it is due to time constraints while developing the app. As there was not enough time to implement persistent data saving completely, the service-pattern is used whenever necessary to accommodate for databases and other ways to save relevant data.

In the current version of the application there is no way to save user-profiles. To keep the same user while the application is active, the active user instance is kept in a class called `UserSingleton`. This makes it possible to reach the user wherever necessary in the application. Note that it is *not* the actual user that is the singleton, as it should be easy to swap out one user for another.

For the images and strings that are used throughout the application, they are stored in the native android directory `src/main/res`. In this directory the android application can reach saved strings in `strings.xml`, images in the `drawable` subdirectory, and many more types of data in other xml files and subdirectories.

To save data throughout the different activities, there are two methods used primarily. Method one is the usage of viewmodels, the relationship between viewmodel and activity is further described above, but the most important part is that an android viewmodel is conscious of the activity lifecycle. That means that even if an activity is destroyed it will remain until the activity calls the built in method `finish()`[5]. The other method is used when there is a need to use data between activities, in that case the built in `SharedPreferences`[6] is used. This saves data with a key and is accessible wherever there is a context, i.e. Activities and Fragments.

5 Quality

When creating the application the aim was to have 100 % of the model tested, and the tests cover close to that. These unit tests can be found in the directory `src/test/java/com/down_to_earth_rats/quiz_game` from project root, and the easiest way to run the tests are by launching Android Studio, right clicking on `src/main/java/com/down_to_earth_rats/quiz_game/quiz_model`, and selecting the option to run the tests with coverage. This will run all the tests for the model and display how much of it that is covered by the tests.

STAN[7] was not able to run properly on any of the computers that it was tested on, and thus there cannot be any generated dependency graph included.

There are no found issues with the application during run-time, but there are things that could be improved or changed, further on this in 5.2 below.

5.1 Access control and security

There is a login-feature in the application and it is required for a user to be logged in for the application to function. At the moment it is just a temporary login and the only user that the application remembers is one that has its credentials written directly onto the code. There are therefor no differentiation between different users, and all logged in users have the same authority.

The login feature is very simple because of time constraints during development, it is however treated as a service, so it is easy to swap out the current login-implementation with something more complex.

As described in the section above, the user is kept within a `UserSingleton` that all relevant parts of the program can reach. There is only support for one user to be logged on at all times, the user logs in on startup, and is logged out upon application finish.

5.2 Improvements

Below are some improvements in the design and code of the application that would have been done if there were more time available:

- Move the `game_mode` package outside the `quiz` package. This because while it does handle parts of the quiz, it is not connected to anything else in that package.
- `UserSingleton` doesn't actually behave as a singleton. At the moment it has static functions and attributes and is never instantiated. The improvement here would be to implement it as an actual singleton, or rename it to something that better describes what it is.
- Removal of the `utility` package. This due to that the generic classes in it are only used in specialised cases so there is no need for them to be generic. As well as that the created `ListIterator` can be replaced with a built in iterator.
- The complexity of the viewmodel for `QuizActivity` is quite high. The viewmodel should be derived into a model part as well as a thinner viewmodel part, where the model handles more logic.
- Removal of the functionality that displays whether a new question or the result will be showed next during a quiz answer. This is already removed from the UI because the game modes make it clear if there are questions left. The removal of this functionality from the code would reduce complexity in many parts of the quiz model and it is currently unnecessary.
- Reducing the amount of attributes in `QuizActivity`. Changing attributes to local variables and moving some to the viewmodel would reduce complexity and make data persist even if Android decides to shut down the activity. Some of the previous improvements would already reduce them somewhat but there are several that are not covered by previous examples.

- Towards the end of development there was a circle dependency found between the `user` and the `category` package. Additionally, the problem `CategoryHandler` is trying to solve, a uniform way to access a collection of categories, is virtually non-existent and a refactoring is much needed in this part. This should be corrected.
- The use of Observer patterns is questionable in many cases. Polling techniques could be used instead, like in `QuestionHandlerObserver`. Another issue is that each Subject implement their own observer version, a more general interface would satisfy both DIP and OCP. This also should be corrected.

References

[1] Google Developers. (2020). Guide to app architecture, [Online]. Available: https://developer.android.com/jetpack/guide#overview (visited on 2020-10-21).

[2] Oracle. (2020). Iterator interface, [Online]. Available: https://docs.oracle.com/javase/10/docs/api/java/util/Iterator.html (visited on 2020-10-21).

[3] Oracle. (2020). List interface, [Online]. Available: https://docs.oracle.com/javase/10/docs/api/java/util/List.html#iterator() (visited on 2020-10-21).

[4] Oracle. (2020). Concurrentmodificationexception, [Online]. Available: https://docs.oracle.com/javase/10/docs/api/java/util/ConcurrentModificationException.html (visited on 2020-10-21).

[5] Google Developers. (Aug. 2020). Viewmodel overview, [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel (visited on 2020-10-21).

[6] Google Developers. (Sep. 2020). Documentation: Sharedpreferences, [Online]. Available: https://developer.android.com/reference/android/content/SharedPreferences (visited on 2020-10-21).

[7] Bugan IT Consulting UG. (2017). STAN — Structure Analysis for Java, [Online]. Available: http://stan4j.com (visited on 2020-10-21).

StudyBuddy, reviewed by Grupp 14 - Down to Earth Rats

We all agree that the application is very well made and were impressed by the code and solutions. Well done!

1. **Do the design and implementation follow design principles?**

   1.1. **Does the project use a consistent coding style?**

   Yes! We believe that it uses a consistent coding style.

   1.2. **Is the code reusable?**

   Yes, the code utilizes interfaces. But more abstractions can be added, like the concrete Broadcast class maybe can use an abstraction.

   1.3. **Is it easy to maintain?**

   Factories are used which greatly increases the maintainability, the factory encapsulates all future changes. Nice! There are comments to help other programmers to understand the code, but it might be a little uneven throughout the project, too little at some places (Broadcast) and maybe too specific at some (AddBroadcastFragment).

   1.4. **Can we easily add/remove functionality?**

   We feel that it seems to be easy to add or remove functionality. The application is structured into well sized packages which makes it modular and thus helps the possibility of adding or removing functionality.

   1.5. **Are design patterns used?**

   The singleton pattern is used in combination with database instances, nice stuff! Also the Observer Pattern is used, for example in BroadcastRepository with the class LiveData.

2. **Is the code documented?**

   There are areas that are not commented and other areas that are. We believe that the areas that are lacking need comments. There are also places with a lot of comments, maybe even too specific if the reader is a programmer.

3. **Are proper names used?**

   Proper names are used, they keep a consistent style throughout the project and are descriptive of the classes' responsibilities. As outsiders, we could easily understand each class's responsibilities.

4. **Is the design modular? Are there any unnecessary dependencies?**

   Refer to point 1.4

   Looking at the class diagram the dependencies seem reasonable and no circular dependencies.

5. **Does the code use proper abstractions?**

   Refer to point 1.2

6. **Is the code well tested?**

   We could not run the tests due to an error, but that might be on us. But we could see that you have written tests and they look okay. But it looks like you can add a “setUp-method” in “UserTest” so you only have to create a user and BroadcastObjects one time instead of doing it in many tests. Maybe reduce the number of asserts used in each test method.

7. **Are there any security problems, are there any performance issues?**

   The code exposes some concrete classes which can lead to unsafe dependencies and rigid code.

   You use FireBase which we think is good for security.

8. **Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?**

   We all agree that the code is easy enough to understand in its complexity. The model doesn’t use any GUI components, thus isolated from the rest.

9. **Can the design or code be improved? Are there better solutions?**

   The Course class currently only has a single String with getter/setter, something seems a little bit off. Why does it need to be a single class that only contains a String?

   Some methods also have a lot of code, so more functional decomposition would be nice.

   More abstractions could be added to improve the project modularity and to future proof additionally.

SDD uses really nice package diagrams!