Optimalt lärande Arkitekturdokument

PUM09

2024-04-12

Version 3.0

Status

Granskad		
Godkänd		

emiho191@student.liu.se

Projektidentitet

Grupp E-post: emiho191@student.liu.se

Hemsida: http://www.liu.se/grouppage

Beställare: Kristian Sandahl, Linköpings universitet
Tfn: 013-28 19 57
E-post: kristian.sandahl@liu.se

Kund: Jörgen Blomvall, Linköpings universitet
Tfn: 013-28 14 06
E-post: jorgen.blomvall@liu.se

Handledare: Lena Buffoni
Tfn: 013-28 40 46
E-post: lena.buffoni@liu.se

Kursansvarig: Kristian Sandahl, Linköpings universitet
Tfn: 013-28 19 57
E-post: kristian.sandahl@liu.se

Projektdeltagare

Namn	**Roll**	**E-post**
Mabest Amin (MA)	Arkitekt	mabam091@student.liu.se
Odin Dahlström (OD)	Utvecklingsledare	odida723@student.liu.se
Emil Holmstedt (EH)	Teamledare	emiho191@student.liu.se
Martin Hultgren (MH)	Kvalitetssamordnare, Databasansvarig	marhu242@student.liu.se
Casper Erik Nerf Kanefall (CK)	Testledare	casne582@student.liu.se
Eric Van Nunen (EVN)	Analysansvarig	eriva185@student.liu.se
Wiliam Puranen (WP)	Konfigurationsansvarig	wilpu732@student.liu.se
Yadgar Suleiman (YS)	Dokumentansvarig	yadsu309@student.liu.se

INNEHÅLL

emiho191@student.liu.se

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2024-02-19	Första utkastet.	MA, OD	EVN, EH
2.0	2024-03-08	Åtgärdade handledarens kommentarer på antaganden och beroenden, arkitektoniska vyer och val, begränsningar och motiveringar samt skapade avsnitt 5.1 och 5.2 för att förtydliga uppdelningen mellan den befintliga och uppdaterade arkitekturen.	MA, OD	OD
3.0	2024-04-12	Uppdaterade och lade till diagram (figur 3, 5 och 8) samt lade till beskrivande text gällande figur 8.	MA, OD	OD

1 INLEDNING

1.1 Syfte

Syftet med arkitekturdokumentet är att möjliggöra en översiktlig uppföljning av utvecklingsprocessen och arkitektoniska designbeslut från idé till implementation av mobilapplikationen Optimalt Lärande. Dokumentet summerar beslut, krav och teknisk beskrivning genom hela processen för att genomgå projektet Optimalt lärande. Detta arkitekturdokument är avsett för projektgruppen samt för externa intressenter för att de tydligt ska kunna följa utvecklingsprocessen och i framtiden kunna bygga på applikationen genom att bygga på en bra förståelse för arkitekturen. Syftet med dokumentet är att beskriva de val och motiveringar som har gjorts och ska göras i utvecklingsprocessen. Det är även viktigt att identifiera vilka vägval som har prioriterats och vilka som har avförts från projektet och det huvudsakliga målet är att alla gruppmedlemmar ska förstå det önskade resultatet.

1.2 Översikt

Arkitekturmål och filosofi definierar de främsta målen för produkten/applikationen och dess tekniska grundprinciper. Antaganden och beroenden går djupare in på de förutsättningar och externa faktorer som påverkar produkten.

Arkitektoniska signifikanta krav beskriver de kraven som projektet har fått av kunden för produkten. Val, begränsningar och motiveringar förklarar de designval som projektgruppen har gjort, i detta inkluderas motiveringar för varför vissa funktioner och dessutom vilka begränsningar som har beaktats samt fastställts.

Arkitektoniska mekanismer beskriver främst de metoder och strategier som bör användas för att skapa en väloptimerad arkitektur över hela projektet även på det som redan gjorts av tidigare PUM-grupper. Nyckelabstraktioner ger en mer grundläggande beskrivning av de funktioner som ska implementeras.

Arkitektoniska designmönster beskriver de strukturer och lösningar som projektgruppen har valt för att hantera specifika utmaningar med arkitekturen. Vyer som representerar systemet visuellt, inklusive användning av ramverk och systemskisser, för att tydliggöra implementeringen samt logiken bakom dessa implementeringar.

2 ARKITEKTURMÅL OCH FILOSOFI

Arkitekturen för produkten bygger på ett tydligt gränssnitt mellan modulerna. Genom att adressera dessa krav kan arkitekturen möta behoven hos både kunden samt användaren genom att erbjuda valmöjligheter.

Vidare är en viktig del av arkitekturen att säkerställa kommunikationen mellan mobilapplikationen och backendsystemet så att det sker effektivt, främst för att föra statistik och optimera inloggningsfunktionen. Genom att ha tydliga gränssnitt och klart definierade arkitekturmål samt krav kan arkitekturen säkerställa en smidig integration mellan de olika delsystemen.

De arkitekturmål och krav som har definierats inkluderar behovet av att hantera olika licenskrav, beroenden, säkerställa effektiv kommunikation mellan mobilapplikationen och backend-systemet samt att erbjuda en tydlig och säker systemarkitektur som kan hantera både nuvarande och framtida krav vilket är ett mål från kunden. Dessa mål formar arkitekturen och filosofin genom att fokusera på att möta specifika krav och behov som har identifierats i projektet från både kunden och användaren.

3 ANTAGANDEN OCH BEROENDEN

Projektet bygger på att återanvända arkitekturen från tidigare kandidatarbeten, vilket är en viktig faktor för de arkitektoniska besluten. Att bygga om hela systemet skulle vara både tidskrävande och ineffektivt, så återanvändning av befintlig arkitektur är fördelaktigt för projektet. Ramverk såsom Django och Flutter återanvänds från det tidigare projektet. Detta bibehålls främst av två större anledningar, att den existerande arkitekturen redan är bra nog men även att det skulle bli tidskrävande att byta ramverk. Det skulle vara möjligt att byta ut vissa ramverk, till exempel Django mot Flask, men det finns ingen större anledning att göra det. Flask är en lättviktigt mikroramverk som är idealiskt för enkla, utbyggbara webbapplikationer, medan Django är ett fullfjädrat ramverk som är bäst för skalbara, funktionsrika webbapplikationer. Det hade gått att byta ramverk till Flask men de funktioner som Flask hade gett projektet skulle inte vara värd den tid som sparas genom att använda föranvända ramverket Django. Detta gäller även ramverket Flutter, det hade varit möjligt att omstrukturera projektet och gå över till plattformsspecifik kod för Android respektive IOS. Däremot så skulle det behövas extra arbete för att implementera appen för både IOS samt Android och då Flutter redan används i den befintliga arkitekturen, så det är inte optimalt eller nödvändigt att byta från Flutter. De ramverk som används i den befintliga arkitekturen anses inte vara ett hinder för att uppfylla de arkitektoniskt signifikanta krav som är ställda i kravspecifikationen (se tabell 2).

Gruppmedlemmarna behöver inte ha några specifika kunskaper för att delta i projektet, eftersom utbildning inom relevanta områden är planerad, såsom kunskaper inom ramverket *Flutter*. Detta påverkar hur arbetsuppgifterna fördelas och vilka områden som prioriteras för inlärning inom gruppen.

Databasarkitekturen utvecklas främst av arkitekten och utvecklingsledaren, med fokus på att skapa en bra struktur som kunden är nöjd med, eftersom kunden har starka åsikter om hur databsen ska vara strukturerad för att kunna bedriva sin forskning.

Driftsättning av databasen och webbservern sker via Docker i Linux-miljö, på en produktionsserver som administreras av LiU-IT. Gruppen har utsett två personer till att vara databasansvariga, och dessa har fått tillgång till produktionsservern via SSH.

4 ARKITEKTONISKT SIGNIFIKANTA KRAV

4.1 Krav på tidigare PUM-projekt

Tabell 1: Tabell över arkitektoniskt signifikanta krav från tidigare PUM-projekt

Krav NR	Kravbeskrivning
1	Appen måste inkludera en funktion för visuell representation av addition.
2	Appen måste inkludera en funktion för visuell representation av subtraktion.
3	Appen måste inkludera en funktion för visuell representation av multiplikation.
4	Appen måste innehålla en funktion för visuell representation av division.
5	Appen ska erbjuda en funktion för visuell representation av användarens förbättring över tid.
6	Appen måste vara kompatibel med minst två olika modeller av iOS-mobiler.
7	Appen måste vara kompatibel med minst två olika modeller av Android-mobiler.
8	Appen måste vara kompatibel och fungera på minst en iPad-enhet.
9	Appen måste vara kompatibel och fungera på minst en Android-surfplatta.
10	Appen måste ha funktionalitet för att kommunicera med en databas.
11	Appen ska ha möjlighet att spara data lokalt på enheten under offlineanvändning // och sedan synkronisera och skicka den sparade datan till databasen när internetuppkopplingen återupprättas.
12	Appen ska spara versionen av applikationen i databasen för att möjliggöra jämförelse av statistik från olika versioner.
13	Information om skola och klass som skrivs in av lärare i appen ska sparas i databasen för att möjliggöra beräkning av användarstatistik.
14	Appen måste vara användbar både med och utan tillgång till internet.
15	Appen måste ha en svarstid på högst 150 millisekunder från det att ett svar matas in tills en ny fråga visas på en iPhone 5S.
16	Appen måste vara kompatibel och kunna köras på den fysiska iPhone 5S som tillhandahålls av kunden.
17	Appen ska inte innehålla någon text förutom namn, inmatning av klass och skola.
18	Eventuella arkitekturförändringar på databasen samt ändringar på den befintliga AI-algoritmen ska diskuteras med kunden innan de genomförs.

4.2 Krav på nuvarande PUM-projekt

Tabell 2: Tabell över arkitektoniskt signifikanta krav på nuvarande PUM-projekt [1]

Krav NR	Kravbeskrivning	Krav NR i kravspec
19	Applikationen ska synkronisera data mellan enheter så att användare kan använda applikationen på flera enheter. Ej samtidigt.	24
20	Applikationen ska stödja flera användare på samma enhet.	20
21	Applikationen ska stödja att flera användare registreras med samma e-mail.	21
22	Applikationen ska autentisera skapade konton via e-mail.	22
23	Applikationen ska lagra statistik i databasen för hur länge användare befinner sig på visualiseringssidan.	2
24	Applikationen ska autentisera inloggning från en ny enhet via e-mail.	23
25	Applikationen ska kunna hämta ut uppgifter från optimeringsalgoritmen.	1
26	Applikationen ska spara data lokalt om internetuppkoppling inte finns tillgängligt. Denna data ska sedan skickas till databasen så fort internetuppkoppling är tillgänglig.	7
27	All funktionalitet i applikationen ska fungera på Android 5 (API Level 21) eller senare.	25
28	All funktionalitet i applikationen ska fungera på iOS 12 eller senare.	26
29	Det ska gå att använda applikationen för att räkna matematik utan tillgång till internet.	30
30	Användaren ska kunna skapa en profil i applikationen utan tillgång till internet.	31
31	Applikationen ska utvecklas i Flutter.	32
32	Backend-systemet ska ha ett REST-API via HTTP som kan användas för att hämta ut användare och räknade uppgifter, med ändpunkter och svarsformat enligt den specifikation som tagits fram och godkänts av kunden.	33

5 VAL, BEGRÄNSNINGAR OCH MOTIVERINGAR

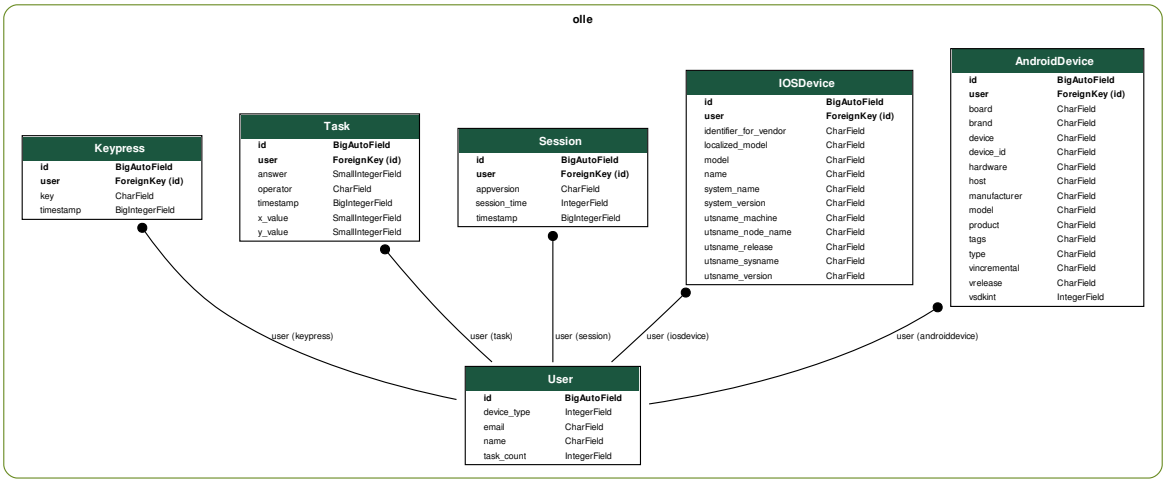
I detta avsnitt diskuteras de begränsningar som sätts på arkitekturen, särskilt med tanke på den befintliga arkitekturen, och de val och motiveringar som har gjorts i den uppdaterade arkitekturen.

5.1 Befintlig arkitektur

Den befintliga arkitekturen av mobilapplikationen Optimalt Lärande utvecklades av två tidigare projektgrupper inom kursen TDDD96 och innehåller funktionalitet som tillåter användaren att öva på fyra olika räknesätt (addition, subtraktion, multiplikation och division) genom att lösa givna räkneuppgifter. En optimeringsbaserad algoritm skriven i C++, som kunden har utvecklat, används för att välja ut uppgifter i en optimal ordning.

Den befintliga arkitekturen inkluderar främst en mobilapplikation (frontend) och ett backend-system som innehåller en webbserver och en databas. Mobilapplikationen skickar data till backend-systemet i syfte att spara statistik för kundens forskning, men har ingen funktionalitet för att synkronisera användarens progression.

Databasdiagrammet i figur 1 beskriver databasstrukturen som de tidigare PUM-grupperna skapade. Databasens struktur definieras av modellklasser i ramverket Django, och omvandlas till tabeller i en PostgreSQL-datbas med hjälp av ORM-systemet i Django. Strukturen för databasen är något som kunden inte vill att projektgruppen ska ändra utan denna är designad på ett optimalt sätt för den befintliga mobilapplikationen, och större omstruktureringar anses därför vara onödiga. Det projektgruppen ska göra är att lägga till nya och utöka befintliga modellklasser (och därmed tabeller i databasen), till exempel för att implementera det kontosystem som beskrivs i de arkitektoniskt signifikanta kraven (se tabell 2).



Figur 1: Databasdiagram för den befintliga databasen (genererat från Django-modeller)

5.2 Uppdaterad arkitektur

I den uppdaterade arkitekturen kommer användargränssnittet att utökas för att utveckla animerade och interaktiva visualiseringar av de fyra räknesätten. Dessutom kommer progressionsskärmen vidareutvecklas för att visa användarens framsteg enligt den beräknade nivån.

Det finns flera begränsningar, några av de identifierade begränsningarna inkluderar att den befintliga applikationen kan ha en påverkan på utvecklingen, speciellt med tanke på tidigare beslut som redan har fattats. Applikationen är också avsedd att vara tillgänglig för olika smartphones, både nya och äldre modeller, vilket innebär att den inte får bli för krävande för prestanda. Detta kommer dessutom att begränsa vilken typ av Android-version eller rättare sagt vilken Android API-nivå som kommer behövas ta hänsyn till för att få med så gamla mobiler som möjligt (detsamma gäller för IOS).

Den uppdaterade arkitekturen ska inte kräva en kontinuerlig internetanslutning mellan backend och frontend eftersom de funktioner som kräver internet är valbara och inte påverkar huvudfunktionerna (att räkna entalsaritmetik), och backend-webbservern kan anropas när uppkopplingen återställs igen. Applikationen ska ha möjlighet att synkronisera lösta uppgifter och progression för en och samma användare mellan olika mobilenheter, genom ett konto som länkar samman de profiler som användaren har skapat på olika enehter. Dessutom ska det även vara möjligt att skapa en profil och spela spelet utan en e-postadress.

Beslut för den minimala produkten innefattar att behålla den övergripande arkitekturen från de tidigare versionerna. Fokus kommer att ligga på att utöka visualiseringar för de fyra räknesätten, inklusive animeringar och interaktivitet, samt att lägga till en skärm som visar progressionen. Databasen kommer att utökas för att implementera det nya kontosystemet, och webbservern kommer att utökas för att implementera det nya REST-gränssnittet.

Framtida utökningar som diskuterades men inte ingår i den minimala produkten inkluderar en språkförändring, vilket ska göra det möjligt för användaren att välja språk när hen skapar en profil. Detta har låg prioritet då målet är att skapa ett spel som använder sig så lite som möjligt av ord och mer av siffror samt symboler.

6 ARKITEKTONISKA MEKANISMER

- Kommunikationsmönster: Beskriver hur olika delar av systemet kommunicerar med varandra, inklusive klient-server-kommunikation och gränssnitt hantering. För närvarande används en enkel klient-server-modell för kommunikation mellan front-end och back-end. Gränssnittet är statiskt och behöver utvecklas för att bli mer interaktivt.
- Datalagringsmekanism, beskriver huvudsakligen hur data lagras och hanteras i systemet, detta omfattar även databasstruktur samt datalagring principer. För närvarande används en databas för att lagra konton samt profiler och övningsdata. Databas strukturen är enkel men behöver utökas för att kunna stödja nya krav från kunden.
- Användargränssnittshantering, specificerar hur användargränssnittet interagerar med användaren och presenterar den visuella informationen på ett användarvänligt sätt. I projektets fall kommer användaren oftast vara barn. För närvarande begränsas användargränssnittet till att visa enkla visualiseringar och räkneuppgifter, något kunden vill att projektgruppen ska utveckla. Utökningen ska inkludera interaktiva och visuella element för att förbättra användarupplevelsen.
- Beteendemönster, definierar generellt hur systemet beter sig i olika situationer och hur systemet hanterar olika händelser och processer. För närvarande finns det endast grundläggande beteendemönster för hantering av användarinteraktion och dataflöde. Detta ska utvecklas och förbättras för att hantera mer komplexa användarscenarier och systemhändelser.

7 NYCKELABSTRAKTIONER

- Profil
  - Syftet är att spara optimeringsdata, statistik och progression i appen.
  - Det ska vara möjligt att spara profilen lokalt på enheten och ska kunna användas online samt offline.
  - Det ska vara helt anonymt (profilnamnet behöver inte skickas till databasen), med tanke på GDPR.
- Konto
  - Syftet med kontot är att synkronisera profiler mellan enheter utan att behöva skapa en profil med en mejladress.
  - All data om kontot ska sparas i databasen och kan bara användas online.

8 ARKITEKTONISKA DESIGNMÖNSTER

Tabell 3: Tabell över arkitektoniska designmönster

Designmönster	Delsystem	beskrivning
Model-View Controller (MVC)	Backend	Designmönstret består av tre komponenter som samarbetar för att skapa en fungerande applikation. - Modell: Hanterar de grundläggande datastrukturerna som används i databasen. Dessa inkluderar modeller för användare, enheter (en för Android och en för iOS), användarsessioner, användaruppgifter och knapptryck. - Vy: Ansvarig för att leverera svar till mobilapplikationen. Det finns olika vyer för att skapa användare, spara träningsessioner (statistik) och visa antalet användaruppgifter. - Styrenhet (Controller): Ansvarig för att uppdatera både vy och modell. Den styrs genom REST API:et för att hantera kommunikationen mellan applikationen och dess bakre del.
REST API	Backend	Applikationsgränssnittet (API) fungerar som en intermediär som styr kommunikationen mellan webbservern och mobilapplikationen. Gränssnittet är utformat enligt REST-designmönstret, detta innebär att en mobilapplikation kan interagera med gränssnittet genom att göra enkla requests istället för att behöva underhålla en pågående session.
Vyhierarki	Frontend	Användarens visuella upplevelse är baserad på en stack struktur där varje vy representerar en sida i applikationen. Den översta vyn i stacken är den som för närvarande visas för användaren. När användaren navigerar till en ny sida läggs en ny vy till i stacken, och när användaren backar tas den aktuella vyn bort från stacken för att visa den tidigare vyn som då hamnar överst i stacken.
Offline-first	Frontend Backend	Offline-first är en arkitektonisk designprincip som kan tillämpas på både backend och frontend av en applikation. Det handlar om att prioritera funktionalitet som gör att användare kan interagera med applikationen även när de inte har tillgång till internet. Således kan offline-first-principen implementeras både i frontend för att hantera gränssnittet och i backend för att hantera dataåtkomst och synkronisering med en server. Det är alltså inte specifikt en backend- eller frontend-teknik, utan snarare en övergripande strategi för att utforma applikationer med offline-funktionalitet i åtanke.

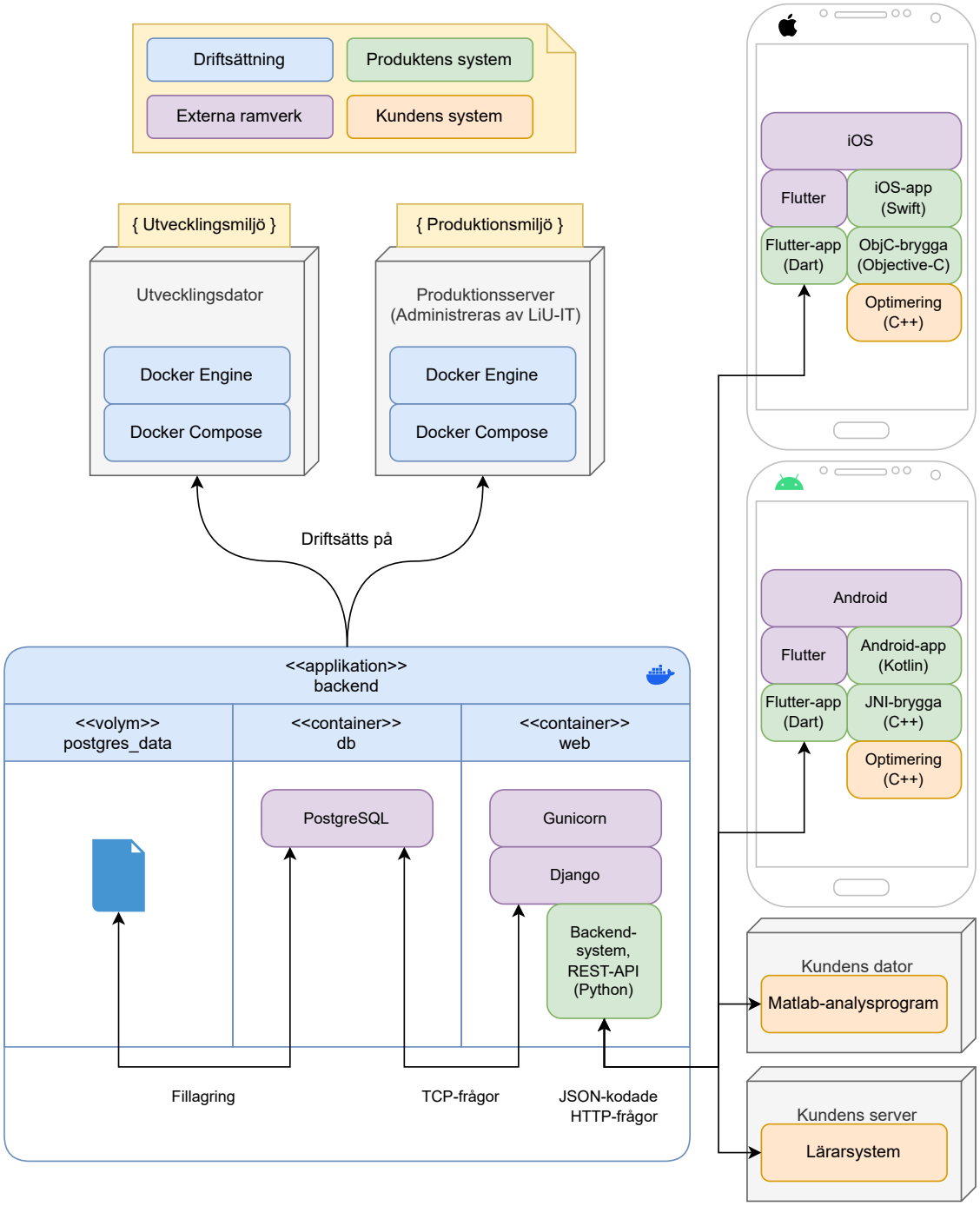
9 ARKITEKTONISKA VYER

Figur 2 visar sambandet och länkningen mellan mobilen där appen kommer finnas och självaste backend och systemet. Denna läkning sker via både en IOS och en Android där färgerna visar vilket lager varje modul är på i både mobil mässigt men även i systemet. Figur 3 är ett use case diagram som tydligt visar vilka aktörer som kommer att göra dessa use cases. Backend-servern som innehåller databasen kommer hantera flera use cases såsom att synkronisera profiler men det är viktigt att urskilja backend-servern med enheten då data kommer att behövas sparas lokalt för att kunna skapa konton offline osv. Aktören lärare är ett krav från kunden som ska ha tillgång till att skapa klasser av elever som hen sedan kan se över via statistik, topplistor, med mera. Figur 4, representerar ett flödesschema över länkningen mellan profiler och konton. Ett konto kan ha flera profiler, men en profil kan inte ha flera konton. Det är viktigt att urskilja dessa två begrepp då tanken är att man ska kunna synkronisera sitt konto från olika enheter där varje enhet har sin specifika profil. För att skapa en helhetsbild av systemet har en systemanatomi tagits fram, se figur 5.

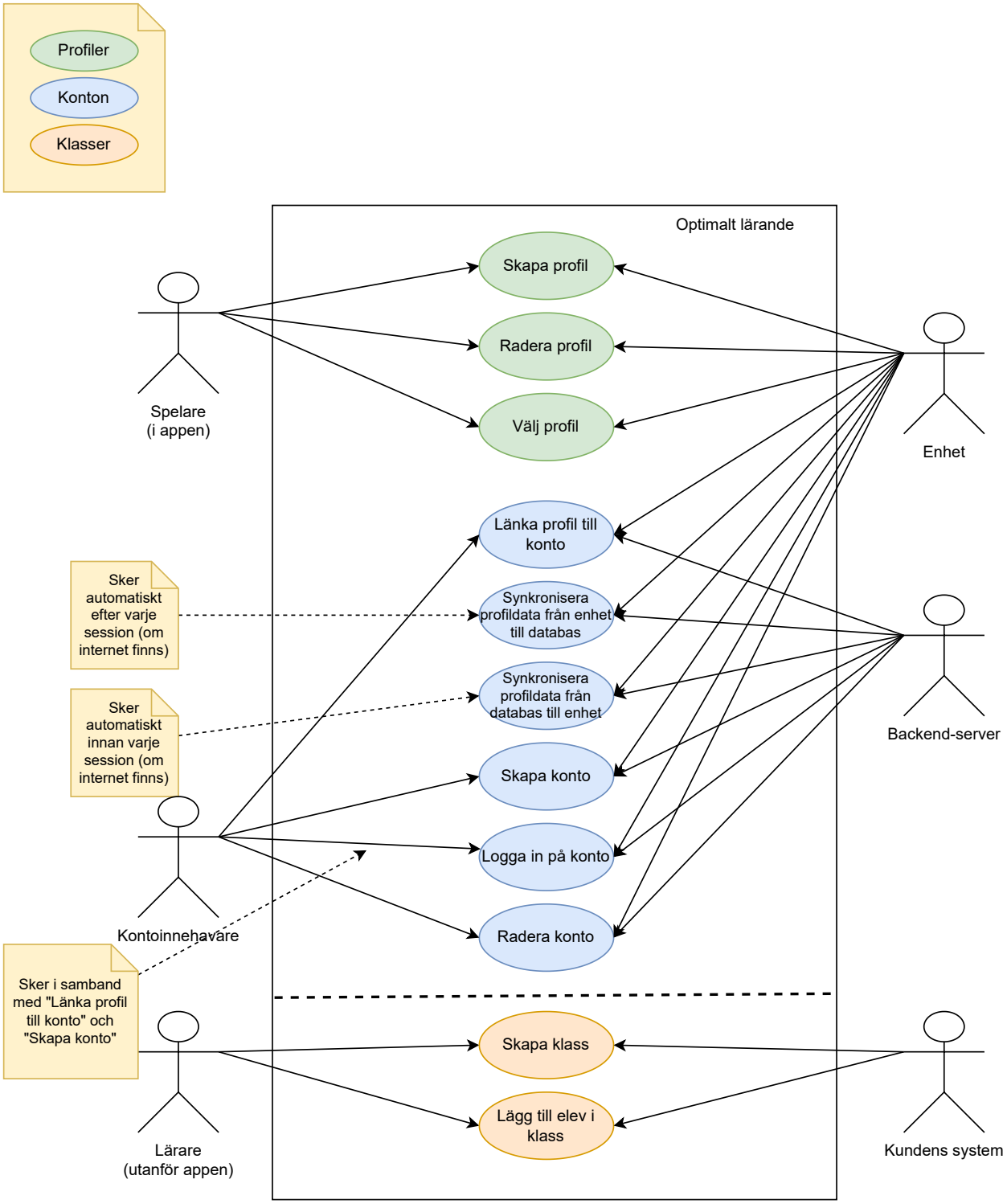
I figur 6 visas de kodmoduler som ingår i systemet, vilket beskriver hur koden är strukturerad i både backend, frontend men även kundens optimeringskod som kompileras in i appen. Detta kan m.h.a visualiseringen från figur 2 ge en bra förståelse för hur modulerna är uppdelade samt deras syfte. Översiktligt filträd representerar mapparna och undermappar, detta delas därefter upp till de centrala filerna i varje mapp. Modulen `olle_app` är en mobilapplikation för iOS och Android som innehåller det grafiska användargränssnittet. Modulen `nativelib` definierar en uppsättning JNI (Java Native Interface)-funktioner och andra hjälpfunktioner som agerar brygga för anrop till funktioner i kundens optimeringsalgoritm. Kundens optimeringsalgoritm som finns i filen `optQuestions.h` utgör ett ramverk för att hantera frågor och svar i algoritmen. Den definierar klasser för att skapa och hantera frågor av olika typer (addition, subtraktion, multiplikation, division) samt spåra statistik och historik för varje frågesession. Koden inkluderar också funktioner för att ladda och spara datafiler samt beräkna spelningstid. Mappen `backend` består främst av kodmodulerna `django_backend` samt `olle`. Modulen `django_backend` är en Python-mdodul och utgör webbserverns huvudmodul, som bland annat innehåller Django-inställningar. Modulen `olle` är också en Python-modul, som är en Django-applikation, och innehåller projektgruppens REST-API och databasgränssnitt. Denna modul innehåller huvuddelen av backend-systemets kod.

Figur 7 representerar ett flödesdiagram för logiken mellan profil/konto, främst när en användare ska skapa respektive. Målet med denna lösning är att det ska vara lätt för användaren att synkronisera sina profiler m.h.a ett konto samtidigt som användaren inte ska lägga ner för mycket tid och ansträning på att skapa profiler och konton utan fokusera på att spela spelet.

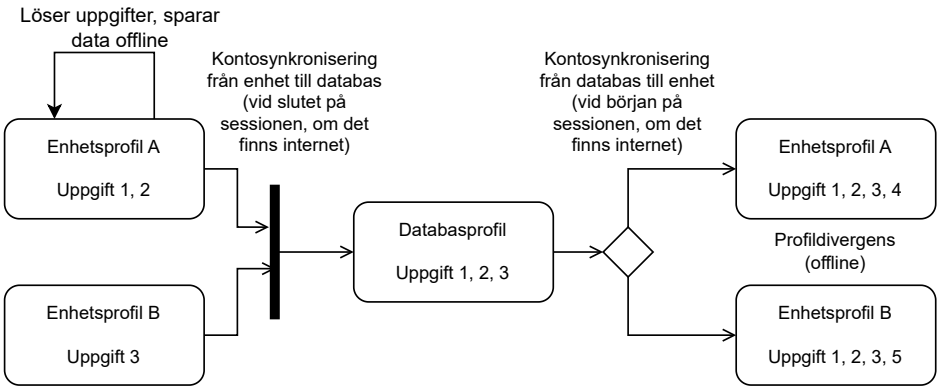
Databasstrukturen för den nya arkitekturen visas med hjälp av EER-diagrammet i figur 8. De nya tabellerna och fälten som har tillkommit är markerade med orange färg, men i övrigt är EER-diagrammet baserat på och har samma innehåll och struktur som det befintliga databasschemat som visas i figur 1. Vissa fält har utelämnats för att förenkla diagrammet, då syns oanslutna streck som ska tolkas representera alla fält som redan har definierats i figur 1.



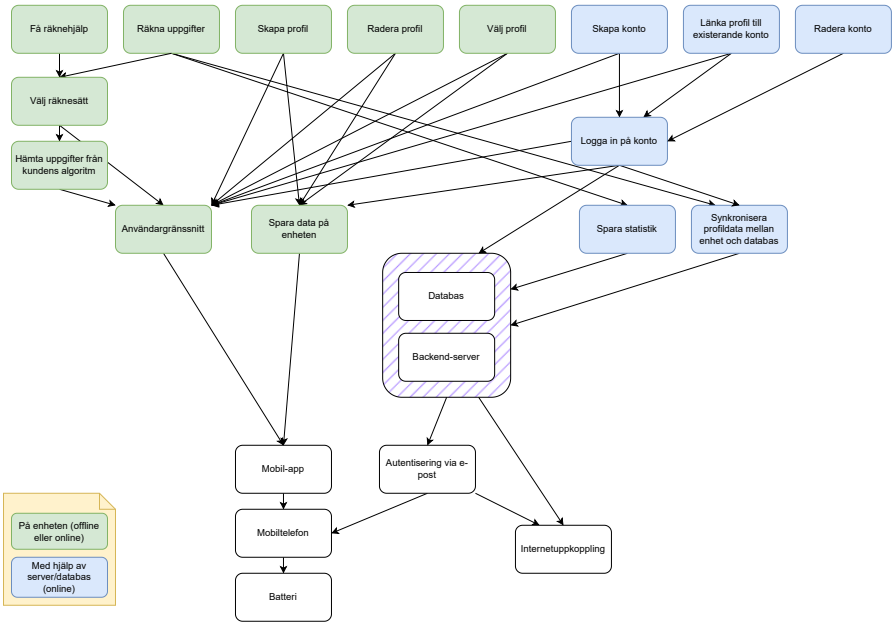
Figur 2: Arkitekturdiagram som visar mobiltelefoner, backend-system och kundens system



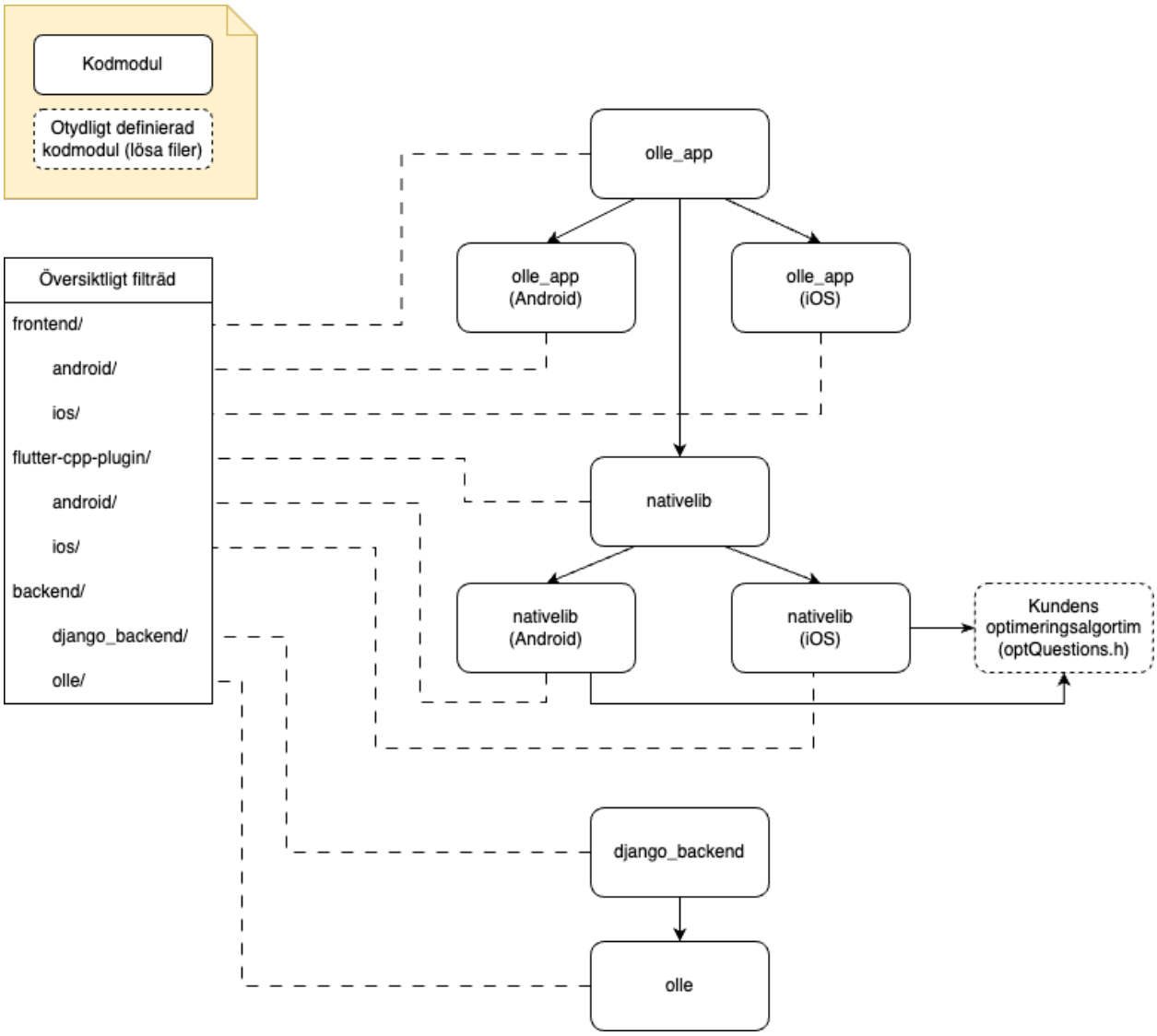
Figur 3: Use case diagram



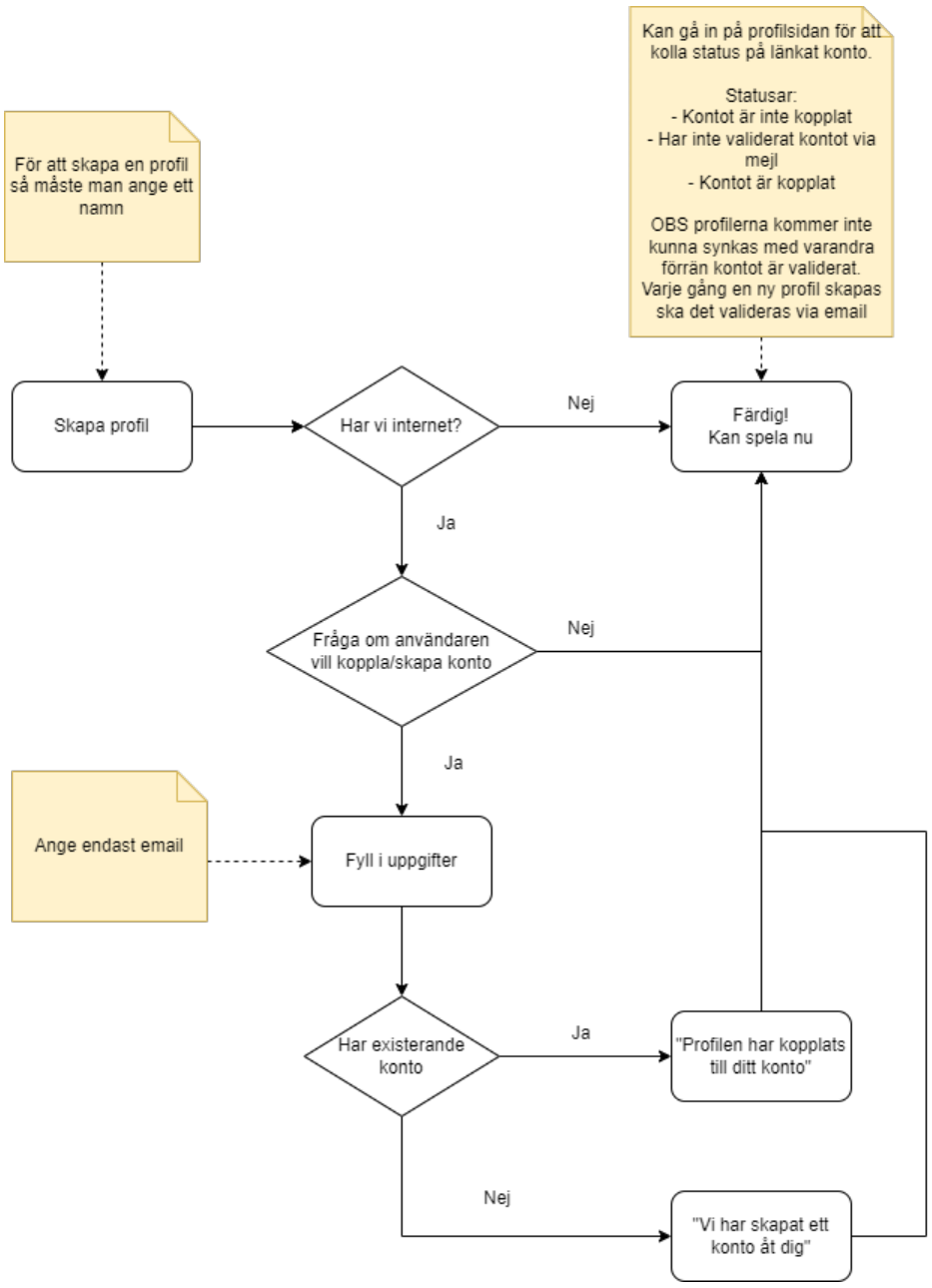
Figur 4: Flödesschema för synkronisering av profil och konto



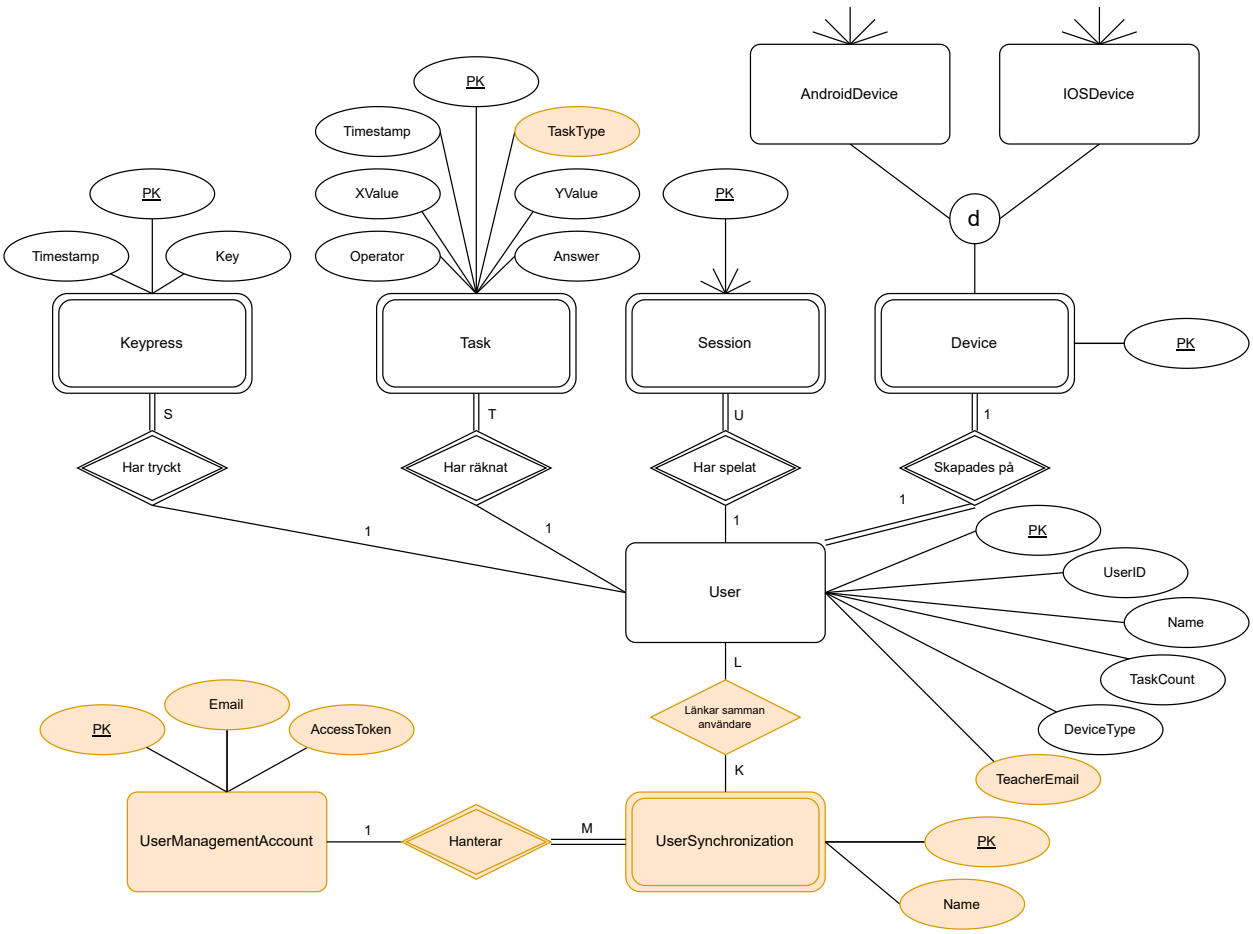
Figur 5: Systemanatomi



Figur 6: Arkitektur över kodmoduler



Figur 7: Konto-profil flödesschema



Figur 8: EER-diagram för den nya arkitekturen (nya tabeller och fält markeras med orange färg)

REFERENSER

[1] PUM09, “Kravspecifikation optimalt lärande,” 2024.