CHALMERS | GÖTEBORGS UNIVERSITET

Design och implementering av ett ansiktsigenkänningssystem för digital autentisering

Examensarbete inom högskoleprogrammet Datateknik

Max Nilsson, Fredrik Ström

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2026
www.chalmers.se

Examensarbete 2026

Design och implementering av ett ansiktsigenkänningssystem för digital autentisering

MAX NILSSON
FREDRIK STRÖM

GÖTEBORGS
UNIVERSITET

CHALMERS

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg 2026

Design och implementering av ett ansiktsigenkänningssystem för digital autentisering
MAX NILSSON, FREDRIK STRÖM

Handledare: Magnus Östgren, Institutionen för Data- och Informationsteknik
Examinator: Johannes Åman Pohjola, Institutionen för Data- och Informationsteknik

Examensarbete 2026
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Illustration av digitalt ansikte skapad med AI verktyget ChatGPT.

Skriven i LaTeX
Göteborg 2026

Design och implementering av ett ansiktsigenkänningssystem för digital autentisering

MAX NILSSON, FREDRIK STRÖM
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet

Sammanfattning

Detta examensarbete syftar till att utforma och implementera en konkret lösning för biometrisk autentisering med hjälp av ett ansikte. Målsättningen för projektet är att skapa en applikation, i vilken en registrerad användare ska kunna verifiera sin identitet med hjälp av en kamera. Ansiktsbiometri erbjuder en intuitiv och säker metod för autentisering, som kan ersätta behovet av lösenord och fysiska nycklar. I detta examensarbete har två olika metoder för både detektering och igenkänning av ett ansikte implementerats, vilket möjliggör en utvärdering av dessa med avseende på systemmässig prestanda och önskat utfall vid en framtida vidareutveckling. Det utvecklade systemet använder sig av utvalda metoder för att generera ett unikt värde i form av en embedding, vilket är en slags vektor som representerar en individs unika ansiktsdrag. Vektorn kan därefter användas för jämförelse med andra ansikten i ett embeddingrum och därigenom påvisa identitet eller olikhet.
Resultatet pekar på att den implementerade applikationen för biometrisk ansiktsautentisering tycks genomföra huvudmålet för projektet på ett tillförlitligt sätt och samtidigt indikerar ett stabilt system utifrån projektets förutsättningar.
Avslutningsvis diskuteras framtida vidareutveckling av applikationen, vilket exempelvis innefattar användningen av alternativa biometriska lösningar såsom fingeravtryck.

Nyckelord: ansiktsigenkänning, ansiktsdetektering, euklidiskt avstånd, biometri, embedding.

Förord

Det här examensarbetet har utförts i samarbete med Broccoli Engineering AB i Göteborg. Stort tack skall riktas till VD Björn Bergholm och konsultansvarig Ola Ljunggren för handledning, stöd och idéer under projektets gång, tillhandahållande av lokal samt GitHub-utrymme för projektet. Dessutom riktas även ett stort tack till Magnus Östgren, doktorand vid Institutionen för Data- och Informationsteknik vid Chalmers Tekniska Högskola, för handledning under projektets gång samt stöd vid rapportskrivning.

Max Nilsson & Fredrik Ström, Göteborg, Januari 2026

Akronymer

Nedan är listan av de akronymer som har använts i examensarbetet ordnade i alfabetisk ordning:

AES	Advanced Encryption Standard
BBox	Bounding box
CNN	Convolutional Neural Network
CRUD	Create, Read, Update, Delete
DI	Dependency Injection
FA	False Accept
FAR	False Acceptance Rate
FCN	Fully Convolutional Network
GCM	Galois/Counter Mode
GUI	Graphical User Interface
IDE	Integrated Development Environment
JPEG	Joint Photographic Experts Group
KNN	K-Nearest Neighbour
MTCNN	Multi-Task Cascaded Convolutional Neural Networks
MVC	Model-View-Controller
NMS	Non-Maximum Suppression
PIL	Python Imaging Library
RGB	Red, Green, Blue
SoC	Separation of Concerns
SQL	Structured Query Language
TA	True Accept
UI	User Interface
VAL	Validation Rate

Innehåll

Figurer

Tabeller

1

Inledning

Idag använder vi oss fortfarande i stor utsträckning av traditionella autentiseringsmetoder för att få åtkomst till olika enheter och system eller för att få tillträde till fysiska platser. Dessa inkluderar kunskapsbaserade metoder (lösenord och PIN-kod) och tokenbaserade metoder (nycklar och passerkort). Många väljer lösenord som är lätta att minnas, där till exempel namn, födelsedagar, favoritfilmer eller husdjursnamn ofta används för att skapa ett lösenord i siffror eller bokstäver. Sådana lösenord är därmed enkla att knäcka genom att helt enkelt gissa sig fram eller använda sig av brute-force dictionary-attacker [1]. Att välja starkare lösenord som är längre och innehåller en kombination av bokstäver, siffror och tecken är därför mer fördelaktigt. Det är dessutom lämpligt att aldrig återanvända samma lösenord för olika applikationer samt regelbundet byta lösenord. Majoriteten följer dock inte dessa råd, vilket leder till att säkerheten för en individs uppkopplade system och applikationer riskeras om ett enda lösenord röjs. Slutligen kan inte ett system veta med säkerhet vem den verklige användaren är om ett lösenord delas mellan exempelvis kollegor. Ett enda svagt lösenord riskerar att säkerheten för varje system som användaren har tillgång till blir utsatt, vilket innebär att den övergripande säkerheten kan bero på hur bra det svagaste lösenordet är. När det kommer till ägandebaserad identifiering kan nycklar bli stulna eller tappas bort, vilket leder till att inkräktare enkelt skulle kunna ta sig in i fysiska eller virtuella utrymmen. En lösning till att göra autentiseringsprocessen säkrare är att använda biometrisk data som baserar sig på individers fysiologiska eller beteendemässiga egenskaper, exempelvis ansiktsdrag, fingeravtryck, röstprofil eller irisstruktur, då biometri är avsevärt mycket svårare att förfalska. Biometrisk autentisering kan därmed öka säkerheten för alla användare i systemet. Biometri tillåter att användaren inte behöver komma ihåg långa, svåra lösenord eller bära på en fysisk nyckel, samtidigt som lösningen kan upprätthålla en hög grad av säkerhet för systemet [2].

Ansiktet är en av de vanligaste och mest använda datan för biometriska identifieringsmetoder. Ansiktsigenkänning har i dagens samhälle fått stor spridning och är numera en välutvecklad teknik för identifiering av användare. Till skillnad från kunskaps- och ägandebaserade metoder, erbjuder ansiktsigenkänning möjligheten att autentisera en användare på ett smidigare och kontaktlöst sätt utan att användaren behöver bära på fysiska nycklar eller minnas ett lösenord. Ansiktsigenkänning utnyttjar istället en persons biometriska egenskaper, som inte kan glömmas, stjälas eller kopieras lika enkelt som de traditionella metoderna och kan därmed potentiellt öka både säkerhet och tillgänglighet.

Ansiktsigenkänning, liksom alla dagens biometribaserade system, har dock begränsningar som kan ha negativa konsekvenser på ett systems säkerhet. Dessa inkluderar:

1. Noise in sensed data (Brus i insamlad data): Mätdatan kan innehålla brus eller vara desorienterad. Bruset kan bero på defekta eller felaktiga sensorer, den omkringliggande miljön eller variationer i biometriska drag såsom ärr.

2. Intra-class variations: Den biometriska datan som genereras från en användare under autentisering kan skilja sig markant från den data som användes under registrering, vilket kan påverka matchningsprocessen.

3. Distinctiveness: Urskiljningsförmågan för individers biometriska drag kan ibland påverkas negativt vid mycket stora likheter mellan valda drag, vilket kan försvåra förmågan att särskilja olika personer åt.

4. Nonuniversality: Alla användare har inte de biometriska egenskaper som förväntas. En del användare har inte möjlighet att extrahera unika identifierare ur ett fingeravtryck medan andra saknar vissa av de karaktäristiska drag i ansiktet som krävs för att kunna generera en embedding.

5. Spoof attacks (Förfalskningsattacker): Angripare försöker lura ett biometrisk system genom att presentera falsk eller modifierad data för att få åtkomst till systemet. Exempelvis skulle bilder, videor eller 3D-masker kunna användas för att lura ett ansiktsigenkänningssystem [2][3].

Vid utveckling och implementering av system för biometrisk autentisering bör även aktuella integritetsaspekter och etiska frågor behandlas, då slutprodukten ofta speglar en kompromiss mellan säkerhet och personlig integritet. Den enskilde användaren bör göras införstådd i vilken data som lagras i systemet, hur det säkerställs att denna lagras på ett tryggt och pålitligt sätt samt hur den kan raderas. Ur ett etiskt perspektiv bör transparens och samtycke vara centrala aspekter vid utveckling och implementering av autentiseringsteknik, då slutanvändaren bör ha full insikt i det kontrakt som denne ingår i med systemägare. Biometrisk autentisering finns idag redan implementerad i delar av världen som en brottsbekämpningsmetod. Att massövervaka en befolkning, oavsett grund, kan ses som ett etiskt övertramp som kränker den personliga integriteten, vilket ger upphov till frågeställningar kring hur långt teknikens fördelar kan försvaras i sådana avseenden.

Det neurala nätverket som utgör själva kärnan i systemet tränas på dataset för att kunna skapa korrekta vektorrepresentationer av ansikten. Dessa dataset kan utgöra en källa till snedfördelning (bias) om de har en alltför snäv representation, har en överrepresentation av en folkgrupp eller helt saknar relevant demografisk representation. Djupinlärningsmodellen kan då få svårt att korrekt identifiera de ansiktsmarkörer som den tränats på, om träningsdatan är obalanserad med låg mångfald av exempelvis olika nationaliteter, kön, åldrar och variationer i hudton. Om träningsdatan saknar viss representation, exempelvis personer som bär slöja, kan systemet få svårt att både urskilja dessa personer och skapa en korrekt vektorrepresentation av dem.
Ett känt problem med diversifiering i dataset finns beskrivet i [4], där modeller

tränade på testdata med låg variation av miljö undersöks. Undersökningen konstaterar att modellerna uppvisar hög prestanda vid identifiering av djur i kända miljöer som återfinns i testdatan, medan de har svårare att korrekt identifiera djuren i nya miljöer.

Ett citat ur VGGFace2: A Dataset for Recognising Faces across Pose and Age [5]:

"The VGGFace2 dataset contains 3.31 million images from 9131 celebrities spanning a wide range of ethnicities, e.g. it includes more Chinese and Indian faces than VGGFace (though, the ethnic balance is still limited by the distribution of celebrities and public figures), and professions (e.g. politicians and athletes). The Images were downloaded from Google Image Search and show large variations in pose, age, lighting and background. The dataset is approximately gender-balanced, with 59.3% males, varying between 80 and 843 images for each identity, with 362.6 images on average."

Sjukdomsfall eller olycka kan leda till vanställda ansiktsdrag och ärrbildning, vilket ställer krav på att systemet bör vara utformat så att användare kan begära att få en ny bild tagen i syfte att skapa en ny embedding. I dessa fall kan den biometriska träningsdatan bidra till diskriminering enligt punkt 4 ovan, då otillräcklig representation kan försvåra systemets förmåga att korrekt detektera ett ansikte som möjligen helt eller delvis saknar utvalda ansiktsmarkörer.

I välutvecklade biometriska autentiseringssystem bör ett tröskelvärde användas för att finjustera resultatet av identifieringsprocessen. Tröskelvärdet är centralt för hur strikt ett system kan avgöra likhet mellan två ansikten. Det bör vara strikt nog för att andelen felaktiga positiva gensvar, FA (False Accept), hålls så låg som möjligt men samtidigt tillräckligt generöst för att tillåta att användare exempelvis har en ny frisyr eller glasögon, vilket annars kan generera felaktigt avvisande utfall (False Negative). Utöver det bör systemet vara tränat på data som inte diskriminerar beroende på variationer i ljussättning, bakgrundsmiljö eller liknande.

Slutligen bör teknik för ansiktsigenkänning utformas för att vara motståndskraftig mot spoofing-attacker, ett känt problem som nämns under punkt 5 ovan. Då biometrisk autentisering kan användas som verifieringsmetod för åtkomst till allt från databaser, sekretessbelagda personuppgifter, privata bostäder och konfidentiell företagsinformation, bör obehöriga aktörer som försöker lura systemet avvisas.

1.1 Om Broccoli

Detta kandidatarbete har genomförts i samarbete med Broccoli Engineering AB (hädanefter Broccoli) i Göteborg. Företaget bedriver konsultverksamhet för ingenjörer med en stor kundbas i Göteborg med omnejd. Deras anställda ingenjörer besitter kunskaper inom tekniska fält såsom data, mjukvara, elektronik, mekatronik och teknisk fysik. Flertalet av dessa arbetar som konsulter ute i industrin.

Företagets intresse för detta projekt grundar sig i vad de uppfattar som ett generellt växande behov av säker autentisering i olika former, där biometrisk ansiktsigenkänning ingår. De upplever en ökad efterfrågan på ingenjörer med kunskaper inom området, då framtiden kräver allt kraftfullare skydd mot dataintrång.

Broccoli uttryckte tidigt under samarbetet att de huvudsakligen ville låta författarna av denna kandidatuppsats utforma och genomföra merparten av detta examensarbete på egen hand. Utöver ett växande behov av ingenjörer med kunskaper inom området för autentisering, undersöker Broccoli även möjligheten att implementera ett system för biometrisk autentisering för internt bruk för in- och utpassering i företagets lokaler. Resultatet av detta examensprojekt har som målsättning att underlätta för företaget att förverkliga den idén.

1.2 Syfte

Detta projekts syfte är att utveckla och implementera ett inloggningssystem med ansiktsigenkänning för att på ett smidigt och säkert sätt kunna autentisera en användare genom att verifiera dennes identitet med hjälp av biometrisk ansiktsdata. Systemet ska kunna avgöra om en användare som försöker logga in är registrerad sedan tidigare. Dessutom ska applikationen vara framtidssäkrad och konstruerad på ett enkelt sätt med hög sammanhållning och låg koppling för att främja framtida utvecklingsmöjligheter.

1.3 Mål

Det övergripande målet är att utveckla ett användarvänligt och säkert inloggningssystem som använder sig av ansiktsigenkänning för att verifiera ett ansikte. Viktiga delmål är:

- Implementera säkerhetslösningar för att förhindra felaktigt eller otillåtet bruk.
- Skapa ett modulärt program med möjlighet till framtida vidareutveckling.

1.4 Avgränsningar

Tidsramar, resurser och tekniska förutsättningar under projektet ligger till grund för följande avgränsningar:

- Inga egna modeller ska utvecklas eller tränas, utan analys och implementation baseras på redan existerande lösningar. Syftet är inte att ta fram nya algoritmer, utan att genom en litteraturstudie undersöka befintliga metoder inom området och välja en lämplig lösning anpassad till projektets omfattning samt implementera denna.
- Projektet syftar inte till att jämföra flera olika modeller eller lösningar, utan grundar sig i att välja och implementera en konkret lösning som kan demonstrera hur biometrisk autentisering med hjälp av ansiktsigenkänning kan tillämpas i en prototyp för inloggning i ett system.
- Systemet innehåller inga avancerade metoder för att skydda mot förfalskningsattacker genom användning av en bild, video eller 3D mask för att förfalska

någons identitet och därmed få obehörig åtkomst. Detta lämnas åt framtida vidareutveckling.

- Systemet är utvecklat som en prototyp och har inte implementerats på externa applikationer, system eller databaser.
- Val och användning av bibliotek, däribland Python och OpenCV, innebär att projektets slutresultat är beroende av valda teknologiers prestanda och begränsningar.

2

Teknisk Bakgrund

2.1 Embedding

Ett embeddingrum är ett linjärt d-dimensionellt vektorrum, i vilket vektorrepresentationer av objekt innehållandes exempelvis data, ord eller bilder kan finnas projicerade. En sådan vektorrepresentation kallas för en embedding och kan användas för avståndsmätningar till andra embeddings för likhetsjämförelser mellan de objekt som vektorerna representerar.

Idén kring utnyttjandet av embeddings (vektorer av biometrisk ansiktsdata) som ett verktyg inom ansiktsigenkänning presenterades först av Schroff et al [6] som en lösning på de utmaningar som ansiktsautentisering stod inför. En embedding kan då liknas vid en form av signaturvektor för identitet med en bestämd längd. Konceptet består i att mappa till en punkt i ett kompakt euklidiskt rum (en d-dimensionell vektor) från en ansiktsbild. De geometriska avstånden i detta vektorrum kan sedan användas för att mäta likheter i identitet mellan olika embeddings.

Vid jämförelse av två embeddings, pekar ett kort avstånd på likhet medan ett stort avstånd tyder på olikhet. Då ansikten som är lika mappas till närliggande punkter i det euklidiska vektorrummet, används tröskelvärde (threshold) och KNN (*K-Nearest-Neighbour*) över vektorerna för verifiering, igenkänning och clustering (klustring).

FaceNet ses idag som föregångaren till flertalet senare implementationer av tekniken. Grundprincipen är densamma för alla.

2.2 Face Detection

Ansiktsdetektering är det första steget inom ansiktsigenkänning, där målet är att detektera och urskilja mänskliga ansikten i bilder samt returnera deras position via en eller flera BBox:ar (*Bounding box*). En klassisk algoritm för ansiktsdetektering är Haar Cascades som utvecklades av Viola och Jones 2001 [7]. Algoritmen visade tidigt en lovande prestanda i form av hastighet och noggrannhet. Sedan dess har tekniken varit under ständig utveckling från manuellt kalibrerade funktioner såsom Haar-liknande drag, till djupinlärningsbaserade metoder som kan förbättra extraheringsprocessen av ansiktsdrag. I de tidiga stadierna för djupinlärning inom ansiktsdetektering spelade flerstegsdetektorer (Multi-stage detectors), såsom

MTCNN (*Multi-Task Cascaded Convolutional Neural Networks*), en betydande roll under utvecklingen [8]. Flerstegsdetektorer genererar först kandidatboxar som vid senare tidpunkt kan omdefinieras för att identifiera ansiktsdrag och hitta ansikten. Flerstegsdetektorerna gav generellt positiva resultat men led av låg effektivitet. Enstegsdetektorer (Single-stage detectors), som exempelvis RefineFace [9], syftar till att förbättra effektiviteten och kan i ett enda steg genomföra både klassificering och returnering av BBox:en. Prestandan ökas genom att enstegsdetektorer helt utelämnar steget att generera kandidatboxar. Enstegsdetektorer har lägre precision än flerstegsdetektorer och är designade för att kunna köras på edge-datorer där beräkningskapaciteten är låg [10].

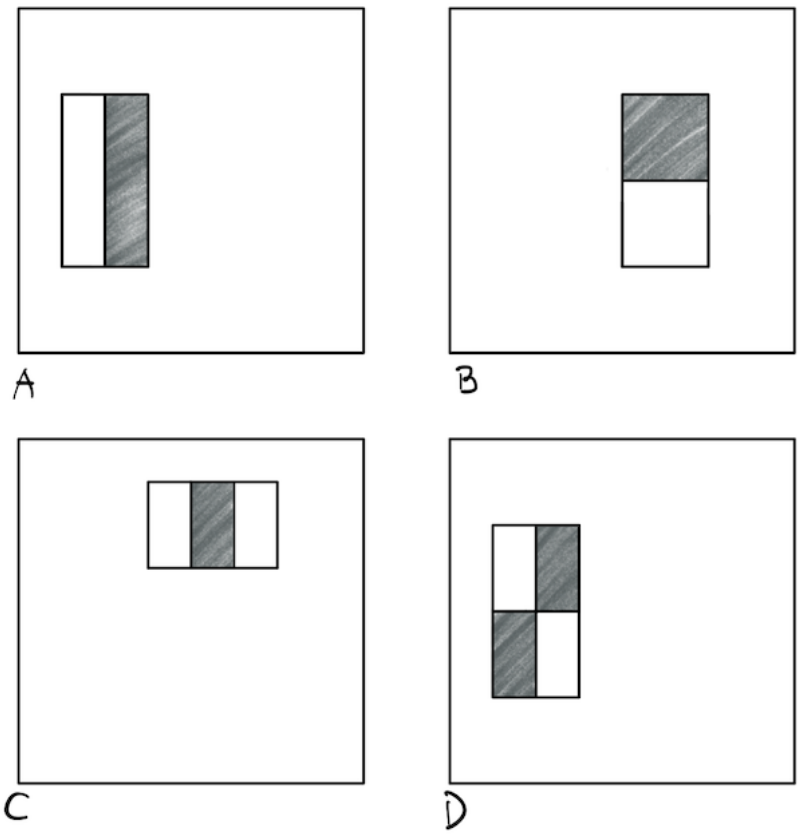
Efter ansiktsdetektering följer ansiktsjustering, som kalibrerar hur ett ansikte är positionerat och förenklar ansiktsigenkänningsprocessen. En av de vanligaste metoderna för detta ändamål är att använda ansiktslandmärken, som exempelvis kan illustreras som fem givna punkter: en för mitten av vardera öga, en för toppen av näsan och en för vardera kant av munnen. Dessa fem punkter används sedan för att justera in ansiktet i affin transformation, vilket är en geometrisk omvandling som bevarar linjer och parallellitet och möjliggör en standardisering av ett ansiktes orientering bland datamängden. Justeringen kan underlätta en senare jämförelse och igenkänning, eftersom den gör det enklare för ett neuralt nätverk att känna igen ansiktet. Kvalitén på ansiktsdetekteringen är därmed ofta beroende av kvalitén på detekteringen av ansiktslandmärken. Kvalitativa ansiktslandmärken kan förbättra detekteringsprocessen och identifiering av dessa punkter kan även avgöra om processen har lyckats eller inte.
Detekteringen av ansiktslandmärken kan delas in i tre kategorier: regression-baserade, heatmap-baserade och modellanpassningsmetoder i 3D. Regression-baserade detekteringsmodeller syftar till att hitta koordinater som direkt kartlägger olika ansiktslandmärken [11]. Heatmap-baserade detekteringsmodeller använder sig av en värmekarta. Intensitetsvärdet för ett ansiktsdrag på värmekartan indikerar sannolikheten att ett landmärke för ansiktet befinner sig på denna position [10][12]. Till skillnad från den koordinatbaserade modellen är heatmap-baserade metoder mer anpassade för att detektera exakta landmärken. Beräkningskostnaden för modellen är dock dyrare och mer känslig för extremvärden, medan den koordinatbaserade modellen däremot anses vara både snabb och robust [13][14]. Under de senaste åren har användningen av ansiktsmodeller i 3D blivit allt mer populära, då användningen av 3D-landmärken förbättrar uppskattningen av ett ansiktes position och orientering (pose-uppskattning). Modellen ger också ytterligare geometrisk information om ansiktets djupdata, vilket kan underlätta att ta fram en mer exakt representation av ansiktet [10][15].

2.2.1 Haar Cascades

Haar Cascades är en maskininlärningsteknik som används för detektering av objekt och utvecklades av Viola och Jones under tidigt 2000-tal [7]. Tekniken bygger på Haar-liknande drag (Haar-like features), som kan beskrivas som simpla rektangulära mönster som syftar till att jämföra skillnader i ljusstyrka mellan olika regioner i en bild. Dessa mönster består av svarta och vita rektanglar och kan vara två-rektangel-,

tre-rektangel- eller fyr-rektangelmönster.

Genom att använda sig av integralbilder (integral images), som för varje punkt (x,y) i en bildrepresentation innehåller summan av alla pixelvärden ovan och till vänster om punkten inklusive punktens egna pixelvärde, kan summan för varje enskild rektangel i bilden beräknas. Denna kan i sin tur användas för att beräkna och extrahera simpla Haar-liknande drag baserat på differensen av summorna mellan närliggande rektanglar.



Figur 2.1: Exempel på rektangulära mönster som utgör Haar-liknande drag, relativt till det omgärdande detektionsfönstret. Den summa som pixlarna innanför de vita rektanglarna utgör subtraheras från summan av pixlarna i de gråa rektanglarna. (A) och (B) visar två-rektangelmönster, (C) visar tre-rektangelmönster och (D) visar fyr-rektangelmönster.

Flera svaga klassificerare (weak classifiers) som vardera är baserade på ett simpelt Haar-liknande drag, kan med hjälp av maskininlärningsalgoritmen AdaBoost (Adaptive Boosting) kombineras till en stark klassificerare. Denna kan i sin tur ordnas i en kaskad (sekvens). Varje stark klassificerare avgör därefter om regionen innehåller ett ansikte. Om en klassificerare returnerar ett negativt svar, gallras regionen omedelbart bort och inga ytterligare kontroller genomförs. Ett positivt svar skickar vidare regionen till nästa starka klassificerare.

AdaBoost är en iterativ algoritm som appliceras mellan varje nytt steg i kaskaden,

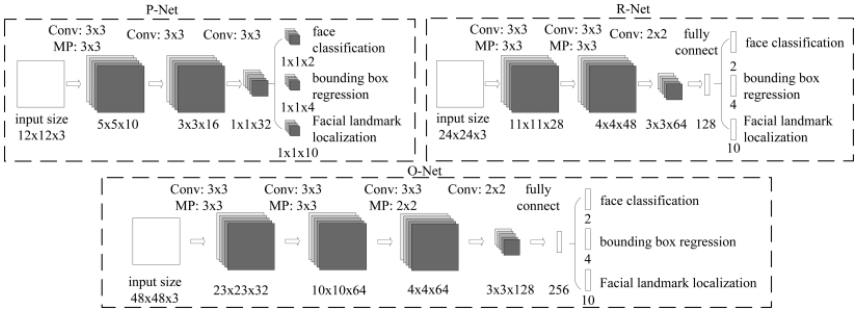
vilket medför att varje steg hanterar allt svårare regioner som passerat tidigare steg. Då flertalet mindre områden i en bild inte innehåller ansikten, möjliggör kaskaden att dessa snabbt och tidigt kan kasseras. AdaBoost kan istället rikta inlärningen till de svårare områdena och därmed användas till att bygga en pålitlig ansiktsdetekterare.

2.2.2 Multi-Task Cascaded Convolutional Networks

Multi-Task Cascades Convolutional Networks är ett ramverk för ansiktsdetektering och ansiktsjustering (identifiering av ansiktets geometriska struktur i en bild) som utvecklades Zhang, Kaipeng et al [8]. De presenterade ett djupt kaskerat multi-task ramverk (deep cascaded multi-task framework) som kan öka prestandan genom att utnyttja sambandet mellan detektering och justering. Ramverket använder sig av en kaskadstruktur med tre steg av djupa konvolutionella nätverk (deep convolutional networks).

Innan en bild går igenom ramverket, skalas den om till olika storlekar. Dessa bygger sedan upp en slags bildpyramid som därefter används som indata för ramverket.

I det första steget tillämpas ett FCN (*Fully Convolutional Network*) kallat Proposal network (P-Net). Här är syftet att ta fram kandidatfönster och extrahera deras BBox-regressionsvektorer. Därefter används ett NMS (*Non-Maximum Suppression*) för att foga samman överlappande kandidater. I det andra steget tillämpas ett CNN (*Convolutional Neural Network*) kallat Refine Network (R-Net), i vilket kandidatfönsterna förfinas genom att en stor mängd falska kandidater avvisas. NMS används därefter ytterligare en gång. I det sista steget går kandidatfönsterna igenom ett Output Network (O-Net) som levererar BBox:en och fem landmärken för ansiktet [8].



Figur 2.2: Strukturen av P-Net, R-Net och O-Net

2.3 Face Recognition

Efter att en bild på ett ansikte har erhållits går den igenom det som kallas funktionsextraktion, för att avgöra vem ansiktet på bilden tillhör. Processen går ut på att omvandla ansiktsbilden till en mer informativ representation av ett ansikte i form av ett embeddingrum, där varje vektor i rummet representerar olika ansiktsdrag. Det är av stor vikt att varje embedding av ett ansikte är utmärkande, så att systemet kan

särskilja olika personer åt vid verifiering och identifiering i större skala. Ansikten av samma person ska ha ett kortare avstånd till varandra i det euklidiska rummet, medan ansikten av olika personer ska ha ett märkbart längre avstånd till varandra. När en embedding har skapats kan ansiktsverifieringen jämföra två ansikten och avgöra om de representerar samma person eller inte med hjälp av ett tröskelvärde. Om avståndet överstiger tröskelvärdet pekar det på olika identitet. Identifiering av en person, vilket kan ses som ett klassificeringsproblem, drar ofta nytta av tekniker som KNN [10].

Historisk sett använde sig tidiga system av statistiska metoder som Eigenface och FisherFace. Utvecklingen av djupinlärningsmetoder har möjliggjort att teknologin för ansiktsigenkänning har gått framåt. Konvolutionella neurala nätverk, CNN, används bland annat för att förbättra dessa system [16].

2.3.1 FaceNet

FaceNet är ett ansiktsigenkänningssystem utvecklat av en forskargrupp ansluten till Google. Forskarna tog fram ett enhetligt system för ansiktsverifikation (”Är detta samma person?”), ansiktserkännande (”Vem är denna person?”) och klustring (”Hitta liknande personer bland dessa ansikten”). Systemet använder deep-learning CNN, som är tränad på att kartlägga ett ansikte utifrån en ansiktsbild till ett kompakt multidimensionellt euklidisk rum där avstånden mellan embeddings motsvarar ett mått på likhet i ansiktsdrag [6]. FaceNet är därmed en implementation av den teknik som diskuterats i tidigare stycken.

Verifiering av ett ansikte sker genom att först konvertera två ansiktsbilder till embeddings. För att avgöra likheten mellan de två ansiktsbilderna, det vill säga om bilderna föreställer samma person eller inte, används det euklidiska avståndet mellan embeddingarna. För att kunna avgöra om ansiktena är tillräckligt lika för att påvisa identitet eller om de tillhör olika personer, behövs ett fördefinierat tröskelvärde.

Till skillnad från verifiering, är identifiering ett klassificeringsproblem som handlar om att kunna förutsäga vem personen är baserat på indata. Detta kan exempelvis uppnås genom att använda en KNN-algoritm, som hittar de embeddings som ligger närmst den givna indatan i det euklidiska rummet.

2.3.2 Triplet Loss

Neurala nätverk implementerade i exempelvis system för ansiktsigenkänning, kan tränas på att känna igen liknande embeddings eller skilja olika personer åt genom förlustfunktionen Triplet Loss. I korthet syftar metoden till att minska avståndet mellan liknande ansikten (embeddings av samma person) och öka avståndet mellan olika ansikten (embeddings av olika personer) i det d-dimensionella vektorrummet. Med hjälp av förlustfunktionen kan en modell för ansiktsigenkänning därmed tränas på att lägga liknande ansikten närmare varandra i vektorrummet.

Detta projekt avser inte att implementera Triplet Loss-funktionen eller träna en modell. För att nå projektets målsättning om att utveckla en fungerande proto-

typ för ansiktsigenkänning, har istället en förtränad modell av FaceNet som redan optimerats med Triplet Loss-funktionen implementerats. En kondenserad men mer detaljerad teoretisk bakgrund kring Triplet Loss återfinns i Appendix 1 under avsnitt A.1.

2.3.3 Tröskelvärde

I detta projekt har ett fördefinierat värde på en tröskelvariabel använts. Tröskelvärdet symboliserar den högsta tillåtna differensen av avstånden mellan två identifierade ansikten i det d-dimensionella embeddingrummet, för att korrekt avgöra om ansiktena tillhör samma person. En differens under eller lika med tröskelvärdet indikerar identitet, medan en differens som överskrider tröskelvärdet indikerar avsaknad av identitet.

Metoder som VAL (Validation Rate) och FAR (Falce Acceptance Rate) används ofta för att kalibrera ett system för ansiktsigenkänning. VAL mäter andelen användare som korrekt släpps igenom, medan FAR visar andelen obehöriga användare som felaktigt godkänns. Genom att modifiera tröskelvärdet och utvärdera VAL och FAR kan önskad precision uppnås. Detta projekt syftar inte till att analysera tröskelvärdet för det utvecklade systemet utefter dessa metoder, utan projektets målsättning om att implementera en funktionell lösning för biometrisk autentisering med hjälp av ansiktsigenkänning har istället legat i fokus. Det valda tröskelvärdet på 0,6 ger tillräckligt god noggrannhet för att möta projektets mål och förutsättningar och baseras på rekommendationer från litteraturen.

En mer djupgående teoretisk bakgrund kring tröskelvärde samt VAL och FAR återfinns i Appendix 1 under avsnitt A.2.

2.4 Säkerhet

Ett system för ansiktsigenkänning kräver flera lager av säkerhet. Då potentiellt integritetskänslig data behandlas i och skickas mellan flera moduler innan lagring i databasen, är korrekt hantering av informationen fundamental för att undvika eventuella dataläckor. En databas bör använda hemliga inloggningsuppgifter, så att direkt åtkomst och manipulering av dess data kan hållas under strikta former. Enskilda användare bör endast ges möjlighet att ansluta till databasen via applikationen samt hämta och lagra information genom specifika metodanrop. Känslig data bör krypteras för att göra den oanvändbar vid eventuella dataintrång och läckor. Systemnycklar bör inte hårdkodas, för att säkerställa att dessa hålls utanför repository och Git under utveckling.

2.4.1 Hashning

Bcrypt är ett väletablerat bibliotek som tillhandahåller tekniska lösningar för hashning av exempelvis lösenord, vilket kan ses som en envägslösning för kryptering av data [17]. Efter hashning är datan både oläslig och omöjlig att återskapa, vilket gör tekniken ideal för lösenordshantering. Hash-funktionen tar en input, exempelvis ett

lösenord, behandlar denna och returnerar en sträng av bestämd längd som består av tillsynes slumpmässiga tecken. Bcrypt skapar ett slumpmässigt 16-byte salt (ett värde) vid varje initial hashning, som i kombination med exempelvis ett givet lösenord används för att göra varje resultat unikt oavsett input. Utöver det används en kostnadsfaktor k som upprepar hashningsprocessen 2^k gånger, vilket gör det svårt för utomstående att återskapa lösenordet genom en så kallad brute-force-attack. Kostnadsfaktorn avgör den beräkningsmässiga kostnaden för hashningen och syftar till att göra en eventuell attack mycket resurskrävande. Saltet och antal krypteringsiterationer sparas i den resulterande hash-strängen. Vid lösenordsvalidering under exempelvis inloggning hämtas den sparade hashen, för att extrahera det initiala saltet och antal krypteringsiterationer. Informationen används därefter för att hasha det tillhandahållna lösenordet. Om detta är korrekt, är resultatet av hashningen av det tillhandahållna lösenordet identiskt med det sparade.

2.4.2 Kryptering

AES (*Advanced Encryption Standard*) är en standardiserad krypteringsalgoritm som använder sig av ett symmetriskt blockchiffer [18]. Vid kryptering produceras en chiffertext (cipher text), som i sin tur används vid dekryptering som konverterar datan tillbaka till läsbar text. Algoritmen kan använda sig av kryptografiska nycklar som är 128, 192 eller 256 bitar långa. En längre nyckel innebär fler krypteringsiterationer och möjliga kombinationer. Då en embedding representerar biometrisk data som skulle kunna användas i andra autentiseringssyften för att få obehörig åtkomst till känslig data eller utrymmen utanför det avsedda systemet, är en 256-bitar lång nyckel lämplig för denna applikation. Det innebär 2^{256} möjliga kombinationer, vilket gör en brute-force attack i praktiken omöjlig att genomföra. AES-nyckeln kan sparas i en .env-fil och därmed hållas dold för utomstående.

GCM, eller Galois/Counter Mode, är ett läge som kan tillämpas på AES [19]. GCM är en välutvecklad algoritm med hög prestanda som kan arbeta med parallella datablock, vilket gör den lämplig för den utvecklade applikationen. GCM producerar en autentiseringstag vid krypteringen som kan avgöra om datan har modifierats i efterhand, vilket kan peka på dataintrång. Under krypteringen använder sig algoritmen av ett nonce (number used once) och skapar vid varje separat krypteringsoperation ett nytt slumpmässigt nonce. Tillsammans med en nyckel krypteras inputen med noncen, vilket garanterar att samma input producerar olika resultat. Slutresultaten ger den ursprungliga noncen, autentiseringstagen samt chiffertexten.

2.4.3 Databasanrop & parametriserade SQL-queries

I applikationer som använder en databas för att spara potentiellt känslig information, är det viktigt ur säkerhetsaspekt att integrera modulerna på ett sådant sätt att potentiella dataläckor inte röjer sekretessbelagda uppgifter. Utöver kryptering, kan den data som delas mellan databasen och applikationen hållas inom strikta ramar genom en välvald utformning av specifika metodanrop. Anropen bör garantera att enbart nödvändig information hämtas från databasen, för att exponera så lite data som möjligt i användarupplevelsen. Utöver fördefinierade queries (frågor)

till databasen, kan attacker genom SQL-injektion effektivt förhindras genom parametriserade queries. En parametriserad query har platshållare för variabler, vilket separerar inmatad data från SQL-kod när den tas emot och behandlas av databasen. Inmatad användardata tolkas därmed som en sträng, vilket effektivt motverkar försök till skadliga attacker då den aldrig kan tolkas som ren SQL-kod.

3

Metod

Under projektets inledande fas gjordes en litteraturstudie på intressanta områden och utvecklade tekniker med anslutning till projektets mål och syfte. Dessa innefattade tekniker för ansiktsdetektering, ansiktsautentisering, kryptering och databashantering. Därefter gjordes ett urval baserat på komplexitet i implementering, hur pass väletablerad och dokumenterad tekniken är samt lämplighet till projektet.

Viktiga delmål för den färdiga produkten:

- Registrera användare med inloggningsuppgifter.
- Scanna av ett ansikte med hjälp av kamera/sensorer.
- Läsa av identifieringsmarkörer.
- Avgöra om dessa finns i en befintlig krypterad databas, det vill säga kunna avgöra om personen är behörig.
- Ge administratörer åtkomst till CRUD-metoder på registrerade användare i databasen.
- Låsa databasen och förhindra otillåten åtkomst.
- Kryptera känslig information.
- Anamma en genomtänkt kodstruktur.

Implementationen av projektets huvudmål, en applikation för ansiktsautentisering, utvecklades i programspråket Python.

Databastypen valdes till PostgreSQL genom biblioteket *psycopg2*, som uppfyller de krav på säkerhet, pålitlighet och modularitet som ställdes för systemet. PostgreSQL-databasen är ett lämpligt val till ett system med ett växande antal användare, vilket framtidssäkrar systemet. För att separera datan något, planeras en tabell för ren användardata samt en för ansiktsvektorer. Dessa kan sedan kopplas samman via ett gemensamt id som nyckelreferens. En användare kan därmed eventuellt ha fler än en embedding vid behov.

För kryptering valdes AES-256-GCM genom biblioteket *PyCryptodome*, då denna lösning erbjuder en robust krypteringsteknik som uppfyller kravet på hög säkerhet. Lösningen tillhandahåller både krypterings- och dekrypteringsmetoder av data, vilket gör den lämplig för att spara känslig information såsom biometrisk data i

databasen för senare rekonstruktion. En användares embedding lagras krypterad i databasen och dekrypteras vid autentiseringstillfället för jämförelse.

För lösenordhantering valdes biblioteket *bcrypt*, som tillhandahåller väletablerade metoder för säker hashning av data. Med hjälp av ett salt hashas en lösenordssträng ett förinställt antal gånger, vilket genererar en sträng med oläslig data. Lösenordet går därefter inte att återskapa. Ett lösenord sparas därmed aldrig i klartext i databasen, vilket gör applikationen motståndskraftig mot eventuella attacker.

Den biometriska ansiktsdetektionen implementeras med Haar Cascades genom OpenCV och MTCNN. En unik identitet som konstrueras av biometrisk ansiktsdata, det vill säga en embedding, skapas med hjälp av det neurala nätverket FaceNet (InceptionResnetV1). Denna förtränas på ett av två möjliga dataset; VGGFace2 [5] eller CASIA-Webface [20].

3.1 Implementering

Ett intuitivt grafiskt användargränssnitt (GUI) bör vara en central del vid implementering av systemet, för att ge användare en väl utformad grafisk representation av applikationen. GUI:t bör dessutom vara lättöverskådligt och enkelt att använda. Tkinter är en lämplig kandidat för applikationens visuella representation, som är ett standardiserat grafiskt användargränssnitt i Python och täcker applikationens behov.

Registreringsprocessen ska kunna registrera en användare med ett unikt användarnamn och lösenord. Den huvudsakliga identifieringsmarkören för användaren ska dock vara den embedding som skapas utefter ett foto på användaren som tas vid registreringstillfället. Som utgångspunkt ska embeddingen sedan användas för identifiering av användaren, för att ge eller neka denne access till ett fiktivt utrymme. Ett delmål är att registreringsprocessen ska kunna avgöra om ett ansikte är dolt vid fototagningen samt ge användaren instruktioner om så är fallet.

En administratör ska skapas och lagras i databasen vid första uppstart av applikationen. Vid varje efterföljande uppstart bör systemet kontrollera att ursprungsadministratören finns kvar i databasen alternativt återskapa denna, så att det kan garanteras att databasen alltid innehåller som minst en användare med administratörsbehörigheter. Inloggningsuppgifter för denne ska vara dolda för vanliga användare, för att förhindra obehörigt intrång i databasen. En administratör ska kunna få tillgång till utvald information i databasen samt kunna manipulera denna. Enbart en administratör ska ha möjlighet att ge andra användare administratörsbehörigheter.

Känslig data i databasen krypteras med hjälp av utvalda metoder för hashning och kryptering för ökad säkerhet. Skulle utomstående trots säkerhetsåtgärder få tillgång till databasen, är känslig information såsom lösenord, foto och embedding oläslig.

När utvecklingen av autentiseringsapplikationen har nått sin slutfas, förväntas tester i begränsad omfattning kunna utföras. Testerna bör kunna avgöra om implementationen kan ge tillräckligt goda resultat för att unikt identifiera en person samt skilja denne från en annan.

3.2 Arkitektur & Designprinciper

För denna applikation är designkonceptet MVC en lämplig kandidat för dess arkitektur, då ramverket ger kodbasen en tydlig indelning och struktur och främjar framtida underhåll. Sammanhörande kodstycken placeras i moduler som kategoriseras för att skilja på användargränssnitt, datahantering och ren logik. Modulariteten gör applikationen framtidssäkrad och gynnar vidareutveckling av enskilda delar.

För att undvika onödiga beroenden i applikationen, bör interfaces och beroendeinjektion tillämpas i stort. Syftet med dessa implementationsdetaljer är att generellt generera så låg koppling som möjligt i applikationen, skapa tydliga implementationskontrakt och ha en distinkt ansvarsfördelning.

Hemligheter såsom inloggningsuppgifter till databasen, administratörsuppgifter och krypteringsnycklar bör sparas i en .env-fil som hålls utanför repositoryt och Git. Vanliga användare hindras därmed från att komma åt dessa systemnycklar.

Applikationen bör kommunicera med databasen genom specifika metodanrop och förutbestämda SQL-queries. Obehörig åtkomst till databasen ska förhindras med hemliga inloggningsuppgifter. Känslig information i databasen ska dessutom krypteras för att ytterligare öka systemsäkerheten.

Programflödet för en vanlig användare planeras se ut enligt följande:

Start → Registrera användare → Välja inloggningsuppgifter → Starta kamera → Detektera ansikte → Ta ansiktsbild → Bearbeta bild → Hasha lösenord, skapa och kryptera embedding → Spara ny användare och embedding i databas → Inloggning → Fylla i inloggningsuppgifter → Validera inloggningsuppgifter → Starta kamera → Detektera ansikte → Ta ny ansiktsbild → Bearbeta ny bild → Skapa ny embedding → Hämta sparad embedding → Jämföra avståndet mellan sparad och ny embedding → Visa utfall av autentisering

3.3 Broccoli & samarbetets upplägg

Initialt genomfördes ett brainstorming-möte med författarna av denna kandidatuppsats och handledare på Broccoli Engineering AB, för att bolla projektidéer, rikta projektet mot en intressant frågeställning och forma en tydlig ram för arbetet.

Den praktiska delen av arbetet har till större delen bedrivits i Broccolis lokaler på Lindholmen i Göteborg. Företagets in-house team har under projektets gång funnits till förfogande för eventuella frågor, stöd och hjälp angående projektet. Det tillhandahölls även ett Github-repository för den kod som utgör projektets programmeringstekniska del.

Arbetet har huvudsakligen genomförts självständigt av författarna av denna kandidatuppsats, utan större styrning eller inblandning av handledarna på Broccoli. Företaget har under projektets gång visat stort intresse för författarnas egna förmåga att problematisera och skapa lösningar på de frågeställningar som diskuterades

efter att projektidén landade i autentisering och ansiktsigenkänning. Författarna gavs därmed fria tyglar att själva utforma projektets mål och metod och har i stor utsträckning sökt svar på de problem och frågeställningar som uppstått under projektets gång på egen hand. Broccolis roll var därmed främst stödjande, där de fanns tillgängliga som bollplank och för teknisk rådgivning samt gav författarna möjlighet att bedriva arbetet i en professionell utvecklingsmiljö.

Då merparten av produktutvecklingen genomfördes i Broccolis lokaler, kunde företagets handledare regelbundet hållas uppdaterade genom korta möten. Regelbundna möten hölls även med projektets akademiska handledare som bidrog med stöd och råd kring projektets övergripande mål samt handledning gällande uppsatsdelen av kandidatarbetet.

4

Genomförande

Utvecklingen av ansiktsigenkänningsapplikationen har skett både på en Windows-dator med Windows 11 och en Mac-dator med MacOS 15.5. Det huvudsakliga kodspråket är Python 3.12. Koden utvecklades med hjälp av IDE:n Visual Studio Code med tilläggen Python och Pylance, som båda är officiella Microsoft-skapade tillägg.

Utvecklingen av applikationen har inte styrts eller påverkats av en sedan tidigare färdigutvecklad projektidé från Broccoli. De krav som ställdes på den färdiga produkten är de som arbetades fram efter projektets start under planeringsfasen.

En enkel Kanban-board har använts för att samla idéer kring implementationsdetaljer under utvecklingen, förenkla arbetsgången samt följa upp de mål som sattes för projektet. Utvecklingen har skett iterativt och applikationen har under utvecklingen omarbetats i faser, för att uppnå en tydlig ansvarsfördelning genom SoC (*Separation of Concerns*) och ett enhetligt flöde. Implementerade moduler och programflöden har genomgått manuella tester och viss enhetstestning innan de integrerats med resten av systemet, för att säkerställa att korrekt bearbetning av information genomförs och rätt logiska beslut tas under informationsflödet i applikationen.

GitHub användes för att tillhandahålla en central plats för att spara och hämta färdig kod samt för säker versionshantering under projektets gång.

Under utvecklingen av autentiseringslogiken planerades och genomfördes tre större steg. Det första för att implementera logik för att detektera ett ansikte, det andra för att skapa en vektor (embedding) av ansiktsdata och det tredje för att validera och jämföra en vektor med en annan.

Det beslutades att hålla det grafiska användargränssnittet, GUI:t, relativt simpelt, användarvänligt och enkelt att navigera. Beslutet fattades då det huvudsakliga fokuset låg i implementering av logiken för ansiktsigenkänning och inte ett avancerat GUI.

Under projektets initiala utvecklingsfas skapades manuellt en PostgreSQL-databas med två tabeller, users och embeddings, med hjälp av standardapplikationen för PostgreSQL 17, Terminal i MacOS samt specifika psql-kommandon. Därefter hanterades databasanslutningen och operationer mot databasen i applikationen med hjälp av biblioteket psycopg2. Ett migrationsskript skapades för att hålla tabellerna uppdaterade med nya kolumner som tillkom under utvecklingen av applikationen. Under utvecklingen fylldes migrationsskriptet på med ytterligare information och funktio-

nalitet riktad mot databasen, för att kontinuerligt hålla databasen uppdaterad, täcka de behov som uppstod under projektets gång samt spegla den implementation som genomfördes i applikationen.

De externa biblioteken har installerats i en virtuell miljö, för att säkerställa att de hålls isolerade från andra projekt och minska risken för eventuella versionskonflikter. En virtuell miljö för externa bibliotek behåller specifika biblioteksversioner och underlättar dessutom för andra att ladda ned och använda koden på egen dator, då projektet hamnar i en identisk utvecklingsmiljö och enkelt kan vidareutvecklas på andra enheter.

För att säkerställa att känsliga uppgifter kan lagras på ett tryggt sätt samt främja en möjlig framtida produktion och distribuering av en färdig applikation, skapades en .env-fil för att hålla på systemets miljövariabler. Dessa miljövariabler motsvarar systemets nycklar, det vill säga användaruppgifter till den skapade databasen, inloggningsuppgifter till det administratörskonto som skapas vid uppstart samt en AES-nyckel för kryptering. Miljövariablerna möjliggör att ingen känslig information finns hårdkodad i applikationskoden, eftersom de kan hållas utanför systemet och aldrig pushas (laddas upp) till repositoryt.

4.1 Design

Applikationen är designad med MVC-konceptet (*Model-View-Controller*) i åtanke, för att dels ha tydliga avgränsningar gällande ansvar genom SoC samt underlätta vid framtida vidareutveckling och testning.
MVC-konceptet bygger på att moduler utvecklas på ett genomtänkt sätt för att främja interaktion mellan dem samtidigt som de har så lite kännedom om varandra som möjligt. Varje modul kan sägas tillhöra ett lager, `Model`, `View` eller `Controller`. I detta projekt benämns vissa moduler tillhöra ett fjärde lager, `Service`.

`Model` ska hålla på datarepresentation och datalogik men har inte insyn i hur data presenteras. Model sägs därmed vara ”smart”.
`View` ska i generella drag göra så lite som möjligt och innehålla så lite logik som möjligt; den ska vara ”tunn”. View ska presentera data, ta emot input och dirigera flödet vidare genom Controller-lagret.
`Controller` ska enligt MVC-strukturen hållas ”dum”. Det innebär att den ska ta emot input från View-lagret och anropa rätt modell eller service samt dirigera om till rätt vy. Här finns affärslogik men ingen eller endast begränsad datalogik.
`Service` kan sägas representera ett lager mellan Model och Controller. Lagret kan användas för att hålla på komplex logik som annars skulle göra Model eller Controller onödigt tunga. Tyngre processer för behandling av data är lämpliga att kategorisera till detta lager, så att Model kan fokusera på datarepresentation och Controller kan delegera större arbetsflöden till Service istället för att själv behandla tung data innan Model kan anropas.

Informationsflödet rör sig typiskt mellan lagren enligt följande:
`Input -> View -> Controller -> Service -> Model -> Service -> Controller -> View`

Denna ansvarsfördelning i applikationen gör dels att moduler har tydliga avgränsningar utan alltför stort ansvar, dels ökar testbarheten för applikationen då enskilda moduler kan testas utan att onödig logik behöver involveras. Uppdelningen främjar även framtida påbyggnad av nya funktioner och gör applikationen skalbar. Moduler ska enkelt kunna bytas ut eller byggas på, utan att förändringar påverkar hela resten av applikationen. Dessutom främjar MVC-konceptet att moduler blir återanvändbara. Både ökad testbarhet och utbytbarhet uppnås även genom att flertalet moduler använder interfaces (Python Protocol).
Applikationen är i stor utsträckning byggd kring DI (*Dependency Injection*), vilket i kombination med interfaces skapar lös koppling mellan applikationens olika delar.

Visuellt är applikationen simpelt byggd, vilket gör det enkelt för användare att interagera med applikationen och förstå de val som finns tillgängliga.

4.1.1 Model

4.1.1.1 user

Datastrukturen representerar en användare och de egenskaper som systemet behöver känna till för att presentera de registerade användarna i `AdminView`. Datastrukturen `user` används av `UserRepository` och `AdminController` via deras respektive interfaces för effektiv dataöverföring.

4.1.2 View

De moduler som tillhör View-lagret har ansvar för presentation av data och hanterar främst UI och indirekt navigering. Gemensamt för nästan alla vyer är att de har ett direkt beroende (dependency) till `tkinter` för att visa fönster samt `tkinter.messagebox` för popup-meddelanden. De har dessutom ett typningsberoende (type-checking dependency) till `from controllers.navigator import Navigator`. Denna import exekveras inte vid körning vilket undviker risk för cirkulär import, men ger IDE:n en idé om parameterobjektet navigator och mer specifikt dess typ. Själva `Navigator`-objektet skapas inte i vyerna, utan fås genom DI efter initiering i `App` i controllern `app_view`. I samtliga vyer ger `Navigator`-objektet, som fungerar som en Controller, tillgång till navigering mellan vyerna med hjälp av `show_view`-metoden i `Navigator`-klassen.

4.1.2.1 admin_login_view

I denna vy har användaren möjlighet att logga in som administratör. Med hjälp av DI av ett `AdminController`-objekt kan den delegera kontroll av användaruppgifter till `admin_service` genom det publika interfacet `IAdminService`. Navigering kan ske bakåt till `LoginView` eller framåt till `AdminView`.
För att öka säkerheten, göms det inskrivna lösenordet bakom stjärntecken.

4.1.2.2 admin_view

I denna vy har en administratör möjlighet att se de registrerade användarna, manipulera behörighetsstatus för access eller admin samt ta bort enstaka användare ur systemet eller tömma databasen helt på användare. Denna modul har även ett beroende till `ttk`, som ger den möjlighet att rendera användarna i en Treeview med en rullningslist.
Genom DI av ett `AdminController`-objekt, har modulen tillgång till alla administratörsåtgärder via interfacet `IAdminService`. Navigering kan ske bakåt till `StartView`.

4.1.2.3 login_view

Här har användaren möjlighet att välja att logga in som vanlig användare, ”User”, eller administratör, ”Admin”. Genom DI av ett `Navigator`-objekt, kan användare navigera vidare `AdminLoginView`, `UserLoginView` eller bakåt till `StartView`.

4.1.2.4 register_view

Vyn ger användaren möjlighet att registrera sig som ny användare. Den har fält för önskat användarnamn och lösenord samt en knapp för att starta kameran och ta ett foto av användarens ansikte. Efter att fotot har tagits får användaren besked om registreringen lyckades eller ej.
Genom DI-controllern `AuthController` kan modulen kommunicera med `auth_service`-modulen via det publika interfacet `IAuthService`, för att starta registreringen eller avbryta kameraprocessen. Navigering kan ske bakåt till `StartView`. Interna metoder ser till att fälten valideras, instruktioner visas samt att merparten av UI:n är låst under tiden som kameraprocessen är igång. Genom callback får modulen ett resultat (success eller error) av registreringen, som då genererar ett popup-fönster och återställer UI:n.

4.1.2.5 start_view

Detta är startvyn för applikationen. Användaren har här möjlighet att navigera till `LoginView` eller `RegisterView`, vilket sker genom DI-controllern `Navigator`:s `show_view`-metod, eller stänga ner applikationen med `Exit`-knappen. Modulen innehåller ingen affärslogik, utan hanterar enbart användarinteraktion. `StartView` initieras med en callback-metod, `on_exit()`, som används för att hantera nedstängningen av applikationen.

4.1.2.6 user_login_view

Modulen ger vanliga användare möjlighet att logga in via autentiseringsprocessen. Här ombeds användaren skriva in sina användaruppgifter, för att starta den biometriska autentiseringen med kameran. Modulen kommunicerar med `auth_service`-modulen genom DI av ett `AuthController`-objekt via `IAuthService`-interfacet, för att validera inloggningsuppgifterna och ta processen vidare. Liksom i de andra vyerna kan användaren navigera, här tillbaka till `LoginView`. Vyn hanterar tillståndet

för UI:n under autentiseringen och hanterar resultatet från kameraprocessen genom en callback.

4.1.3 Controller

4.1.3.1 admin_controller

`AdminController`-modulen ansvarar för att hämta och hantera användardata och ger administratörer tillgång till metoder för att manipulera databasens innehåll. Via interfacet `IAdminService` kan `AdminView` kommunicera förändringar till `AdminController` baserat på användarinteraktion, som `AdminController` i sin tur kommunicerar vidare till ett injicerat `UserRepository`-objekt via dess interface `IUserRepository` för åtkomst till databasen.

`AdminController` tillhandahåller metoder för att skapa en default-administratör, hämta alla användare, ändra status för access och admin, ta bort enskilda användare ur systemet eller radera hela innehållet i databasen samt validera användaruppgifter för en administratör vid inloggning. Valideringen sker till ett injicerat `CryptoService`-objekt via dess interface `ICryptoService`.

Modulen har ett beroende till `User`, för att kunna skapa `User`-objekt och fylla listan med användare.

4.1.3.2 app_view

I `app_view`-modulen finns klassen `App` som har en mycket central roll i applikationen. Klassen ansvarar för att skapa och koppla samman andra moduler som tillhör Model-, View- och Controllerlagren. Den innehåller Tk-root-fönstret som den ärver från `tk.Tk`, vilket gör modulen till hela applikationens rot för GUI:t. Modulen innehåller dock ingen logik kopplad till användarinteraktion eller programflöde och har heller ingen egen visuell representation. Den kommunicerar information mellan vyerna och till andra moduler tillhörande Controller- och Servicelagren i applikationen.

Vid instansiering av App-klassen i `main()` injiceras nödvändiga tjänster och controllers via DI genom respektive interface (`AdminController`, `ProcessManager`, `UserRepository`, `CryptoService` och `FaceEmbedder`). I `app_view`-modulen kan därefter andra tjänster och controllers skapas, varav vissa konstrueras genom att injicera de mottagna objekten genom DI (`FaceVerifier`, `Navigator`, `LifecycleManager`, `FaceCaptureService` och `AuthController`). `AuthController`-objektet ger `RegisterView` tillgång till controllerns metoder via dess interface. Modulen håller ett `AdminController`-objekt, som anropas via dess interface av `AdminView` och ger tillgång till administratörsåtgärder.

Genom instansiering av `LifecycleManager` ansvarar `App` för korrekt hantering och nedstängning av bakgrundsprocesser, framförallt vid nedstängning av applikationen. `App`:s `Navigator`-objekt är vyernas knutpunkt och ansvarar för navigering mellan samt visning av dem.

4.1.3.3 auth_controller

I `auth_controller` finns metoder för registrering och inloggning av användare, vilket gör modulen till en viktig del i applikationens programflöde. Modulen samordnar affärslogik med användarinteraktioner relaterade till registrering i `RegisterView` och autentisering i `UserLoginView`. `AuthController`-objektet skapas i `app_view`-modulens `App`-klass och injiceras med objekt av `UserRepository`, `CryptoService`, `FaceCaptureService`, `FaceEmbedder` och `FaceVerifier` som alla definieras genom sina respektive interfaces. Det gör att `AuthController` inte känner till objektens specifika metodimplementationer utan enbart har ett beroende till interfacen.

`AuthController` kan genomföra registrerings- och inloggningsprocesserna genom strategiska metodanrop till andra moduler och ett organiserat arbetsflöde. De callback-baserade metodanropen är icke-blockerande, vilket innebär att resten av GUI:t lämnas responsivt. `AuthController` hanterar även callbacks med resultat (True/False samt meddelande) från metodanrop till andra moduler.

4.1.3.4 navigator

`Navigator`-klassen fungerar som en slags controller för View-lagret i interaktionen mellan vyerna och `App`. Den injiceras med ett `AdminController`- och `AuthController`-objekt, som den därefter kan skicka vidare genom DI till berörda vyer vid initiering av dessa. Den injiceras även med en callback-metod som används vid nedstängning av applikationen, som den skickar vidare specifikt till `StartView`. Modulen hanterar logiken för att skapa, lagra och växla vy. För att hålla systemet responsivt skapas alla vyer när `Navigator`-objektet instansieras i `App`. Därefter sparas de i en `dictionary` av typen nyckel-värde. Vybyte sker via anrop från vyerna till `show_view()`-metoden via DI av ett `Navigator`-objekt, som med ett giltigt namn som inputsträng exponerar motsvarande vy. För att hålla listan med användare aktuell, anropas `refresh_user_tree()`-metoden i `AdminView` innan denna vy visas.

4.1.4 Service

4.1.4.1 crypto_service

Modulen innehåller klassen `CryptoService`, som tillhandahåller metoder för att kryptera och dekryptera känslig information såsom vektorer och foton samt hashning av lösenord. För att öka säkerheten i anslutning till datahanteringen i applikationen, har lösenord och nycklar inte hårdkodats i applikationen och skrivs heller aldrig ut i klartext vid bearbetning av miljövariablerna. Skulle utomstående få tillgång till applikationens källkod, kan de därmed inte få tillgång till några känsliga uppgifter.

För kryptering används AES-256-GCM genom modulens beroende till biblioteket `pycryptodome` (närmre bestämt `Crypto.Cipher.AES`) och för hashning används biblioteket `bcrypt`. Modulen innehåller en metod för att ladda en AES-nyckel från en miljövariabel i en .env-fil, som kallas på i `main`-modulen vid uppstart. Det är även i `main()` som en `CryptoService`-instans skapas tillsammans med den inlästa AES-nyckeln, som i sin tur injiceras till `AuthController` via `App` och `AdminController`

via `main()`.

Klassen implementerar interfacet `ICryptoService`. Klassens metoder anropas via interfacet, vilket leder till att anropande moduler enbart känner till metodsignaturer och inga implementationsdetaljer.

4.1.4.2 face_capture_service

`FaceCaptureService`-klassen hanterar arbetsflödet mellan de delar i applikationen som styr bakgrundsprocessen med kamera och detektering av ansikte, samt kommunikationen mellan bakgrundsprocessen och huvudprocessen.

Klassen ansvarar för att skapa ett `Process`-objekt och ett `Queue`-objekt som kan hålla processens resultat och därefter starta en kameraprocess med `run()`-metoden i `camera_worker`-modulen. Den ansvarar även för att hämta resultat från kameraprocessen med hjälp av en polling-metod och städar därefter upp och frigör resurser med hjälp av `cleanup()`-metoden. Genom DI av en `ProcessManager`-instans, säkerställer modulen att en startad process kan hanteras centralt i ett `ProcessManager`-objekt via dess interface.

En instans av `FaceCaptureService` skapas i `App` och skickas därefter vidare till `AuthController` genom DI. `AuthController` kommunicerar sedan med `FaceCaptureService`-objektet via dess interface och känner därmed bara till metodsignaturer. Genom interfacet kan `AuthController` anropa `FaceCaptureService`-objektets metoder vid registrering och inloggning.

4.1.4.3 face_capture

Klassen instansieras i `camera_worker`-modulen och injiceras då med ett `FaceProcessor`-objekt. `FaceCapture`-klassens enda metod, `capture_face()`, ansvarar för all direkt interaktion med kameran (för att ta en bild) via OpenCV-biblioteket samt att returnera resultatet. Klassen samarbetar med `FaceProcessor`-klassen för att kunna ge visuell feedback till användare genom att rama in ett detekterat ansikte i en grön rektangel, som skapas utefter de koordinater som fås genom metodanropet till `FaceProcessor`-objektets `detect_first_bbox()`. Koordinaterna motsvarar rektangelns övre vänstra samt nedre högra hörn. `FaceCapture`-klassen anropar även `FaceProcessor`-objektets `encode_face()`-metod för att beskära och omvandla en bild till JPEG-format.

Klassen innehåller simpel logik för att kunna tolka användarinput från tangentbordet, för att antingen ta en bild (Enter) eller avsluta processen (Esc).

FaceCapture-klassen har beroenden till `cv2` (OpenCV), `numpy` (för hantering av bilddata), `time` (för att implementera en timeout för kameran vid inaktivitet) samt `FaceProcessor`.

4.1.4.4 face_detector

`FaceDetector`-klassen instansieras också i `camera_worker`-modulen och tar därefter emot anrop från `FaceProcessing`-klassen via sitt interface. I `face_detector`-

modulen finns logiken för själva bildbehandlingen och ansiktsdetektionen. Vid instansieringen av klassen tar den emot en typparameter som indikerar vilken detekteringsmodell som ska användas (förvalda MTCNN eller Haar).
`detect_faces()`-metoden returnerar en BBox med koordinater till det övre vänstra samt nedre högra hörnet för den rektangel som ramar in ett upptäckt ansikte. Metoden upprepar detta för varje detekterat ansikte.

`FaceDetector`-klassen implementerar ett eget interface, `IFaceDetector`, som möjliggör utbytbarhet och testning av `FaceDetector`-klassen. Utöver `IFaceDetector`-interfacet har klassen även beroende till `cv2` (för Haar Cascade-modellen och bildbehandling), `numpy` (för att kunna ta in en numpy-array och behandla denna för ansiktsdetektering), `DetectorModelType` (för val av detektionsmodell) och `BBox` (för definition av BBox) samt `mtcnn` (som enbart importeras om detta är modellvalet genom så kallad *lazy load*).

4.1.4.5 face_embedding

`FaceEmbedding`-klassen instansieras i `main`-modulen. Klassens metoder anropas därefter via dess interface. I `FaceEmbedding`-klassen finns den databehandlingslogik som krävs för att omvandla en ansiktsbild till en embedding. Embeddingen används sedan i autentiseringsfasen, vilket gör logiken i `FaceEmbedding`-klassen särskilt kritisk för applikationens funktionalitet. Vid initiering av klassen tar den dels in den förtränade deep-learning modellen FaceNet (InceptionResnetV1) och dels ett dataset som modellen har tränats på (VGGFace2 eller CASIA-Webface). Valet av dataset genomförs i tidigare anrop till `create_model()` metoden i `main`-modulen.

Skapandet av ansiktsvektorer sker i `embed()`-metoden, som efter genomförd beräkning returnerar en embedding.

Två metoder ligger utanför klassen; `create_model()` och `_to_rgb_pil()`.
`create_model()` tar en dataset-parameter och returnerar därefter motsvarande förtränade modell.
`_to_rgb_pil()` tar in en bild som en numpy-array, konverterar den till RGB, skapar ett PIL-Image-objekt (Python Imaging Library), storleksomvandlar till 160x160 pixlar samt returnerar den behandlade bilden. Metoden förbereder en bild för att i senare skede kunna skapa en ansiktsvektor.

Modulen har beroende till `torch` (djupinlärningsramverk för modellen), `torchvision` (för bildtransformeringar), `facenet_pytorch` (laddar en förtränad modell för ansiktsigenkänning), `cv2` och `PIL` (bildomvandling och bildhantering) samt `numpy` (returtyp för färdig embedding). Den har även beroende till `EmbeddingModelType` (för val av dataset) och interfacet `IFaceEmbedder`.

4.1.4.6 face_processing

`FaceProcessing`-klassen instansieras i `camera_worker`-modulen och injiceras då med ett `FaceDetector`-objekt med hjälp av dess interface. `FaceProcessing`-klassens metoder anropas därefter av `face_capture`-modulen. Ett `FaceProcessing`-objekt kan ta emot bilder och returnera en BBox av det första detekterade ansiktet. Klassen

kan även beskära bilder samt lägga till viss marginal (padding) så att ett detekterat ansikte fångas i sin helhet. Slutligen kan klassen konvertera en bild från en numpy-array till JPEG-bytes. `FaceProcessing`-klassens publika metoder innehåller begränsad logik och anropar främst metoder i andra moduler för vidare bearbetning.

`FaceProcessing`-klassen har beroenden till `cv2` (för bildbehandling), `numpy` (för dataomvandling och beskärning), `BBox` från `detector_types` (för definiering av BBox) samt `IFaceDetector`.

4.1.4.7 face_verification

Klassen `FaceVerifier` jämför två embeddings med varandra för att avgöra om de tillhör samma person. Klassen instansieras i `App`-klassen och implementerar ett interface för att dölja implementationsdetaljer från anropande moduler. `AuthController` anropar klassens `verify()`-metod genom interfacet för verifiering av identitet. Ett `FaceVerifier`-objekt kan eventuellt instansieras med ett val av avståndsmått samt ett tröskelvärde.

Avståndsmåttet, det vill säga metric, kan väljas till euklidiskt (euclidean) eller cosinus (cosine). Standardmåttet är euklidiskt. Tröskelvärdet används för att anpassa och finjustera applikationens säkerhet, för att på så vis minska antalet falska positiva svar som accepteras och samtidigt även minska antalet falska negativa svar som avvisas av systemet. Som standard är tröskelvärdet satt till 0,6.

Metoden `verify()` tar emot två embeddings och anropar därefter den privata metoden `_compute_distance()`. Där beräknas avståndet mellan de båda vektorerna med det angivna avståndsmåttet och resultatet jämförs sedan med det givna tröskelvärdet. Metoden returnerar därefter ett positivt svar om avståndet är mindre än eller lika med tröskelvärdet. Ett positivt svar tolkas som att båda vektorerna representerar samma person.

`FaceVerifier` har beroende till `numpy` och `sklearn.metrics.pairwise` (för matrisberäkningar utefter valt avståndsmått), `IFaceVerifier` samt `Metric` (enum som definierar tillgängliga avståndsmått).

Interfacet möjliggör att modulen med enkelhet kan bytas ut utan att resten av applikationen påverkas i stort.

4.1.5 Övriga moduler

4.1.5.1 camera_worker

Modulen ansvarar för delar av kameraprocessen som sker i bakgrunden vid registrering och inloggning. Modulen körs som en separat process via `multiprocessing` och kan kommunicera ett resultat tillbaka till huvudprocessen genom en `Queue`. Kameraprocessen kan därmed hållas isolerad från applikationens huvudprocess, vilket håller applikationen responsiv om kamerarelaterade störningar uppstår. Modulen tar emot den valda detektorn som en sträng och instansierar därefter objekt av klasserna `FaceDetector`, `FaceProcessor` och `FaceCapture`.

4.1.5.2 database

Modulen tillhör ett eget lager som kan benämnas Repository och sköter all kommunikation mellan applikationen och PostgreSQL-databasen. Modulen implementerar interfacet `IDatabase`, vilket främjar möjligheten till byte av databastyp.

Modulen ärver från `psycopg2` för att möjliggöra kommunikation mellan databasen och applikationen, samt `os` för att hämta miljövariabler till databasen från en .env-fil.

`Database`-klassen kapslar in de viktigaste och mest grundläggande metoderna gentemot databasen, vilket inkluderar att skapa tabeller, exekvera queries, hämta och spara data samt stänga anslutningen mellan applikation och databas. `Database`-klassen har stöd för att användas som en så kallad kontextmanager, vilket möjliggör att utnyttja den i ett with-block vid instansiering i `main()`. Det garanterar att kopplingen mellan applikation och databas stängs även om det inträffar körningsfel.

Alla anrop till databasen sker med parametriserade queries, vilket förhindrar SQL-injektion.

4.1.5.3 interfaces

Ett antal interfaces har skapats för att ge tydliga riktlinjer om vilka metodsignaturer en modul som implementerar ett interface måste erbjuda. Interfacen säkerställer att moduler som anropar metoder i andra moduler via ett interface inte känner till implementationsdetaljer kring metoderna utan bara metodsignaturerna.

- `admin_controller_interface: IAdminService` - Administrativa metoder.
- `auth_controller_interface: IAuthService` - Autentiserings-, valideringsoch registreringsmetoder.
- `crypto_service_interface: ICryptoService` - Metoder för kryptering, dekryptering och lösenordshantering.
- `database_interface: IDatabase` - Databasmetoder.
- `face_capture_service_interface: IFaceCaptureService` - Metoder för att ta bild med kamera.
- `face_detector_interface: IFaceDetector` - Metoder för ansiktsdetektering.
- `face_embedding_interface: IFaceEmbedder` - Metoder för hur skapandet av embeddings ska gå till.
- `face_verification_interface: IFaceVerifier` - Metoder för ansiktsverifiering.
- `process_manager_interface: IProcessManager` - Metoder för hur hanteringen av bakgrundsprocesser ska gå till.
- `user_repository_interface: IUserRepository` - Metoder för hur användares data lagras, hämtas och uppdateras i databasen.

4.1.5.4 lifecycle_manager

Klassen `LifecycleManager` fungerar som en form av hjälp- och servicemodul och har ett övergripande ansvar över applikationens resurser och processer. Den tar, via dess interface, emot ett `ProcessManager`-objekt genom DI och har därmed ett beroende till `IProcessManager`. `LifecycleManager` registrerar processer och städfunktioner. Vid nedstängning av applikationen körs städfunktionerna, samtidigt som modulen även ser till att terminera både `ProcessManager`-objektet och eventuella andra aktiva processer.

Vid vidareutveckling av applikationen, med flera resurser och processer som ligger utanför `ProcessManagers` nuvarande ansvarsområde, kan `LifecycleManager` användas för att hantera applikationens generella resurshantering vid framförallt nedstängning.

4.1.5.5 main

`main`-modulen ingår inte i MVC-strukturen, utan fungerar som en startpunkt för applikationen. Här laddas miljövariabler (användarnamn och lösenord) från en .env-fil, för att via metodanrop till `seed_admin()` i `AdminController` skapa en administratörsanvändare. Här skapas även en instans av `CryptoService` med hjälp av en AES-nyckel som även den hämtas från en miljövariabel.

Ett `Database`- och `ProcessManager`-objekt instansieras och används som kontextmanagers i ett with-block, för att garantera att de stängs ner även vid programfel. Nödvändiga SQL-tabeller i databasen skapas genom anrop till `Database`-objektets `create_schema()`-metod.

`main`-modulen bär huvudansvaret för att skapa klassobjekt som kopplas via interfaces samt instansiera beroenden med DI. `FaceEmbedder` skapas med en förtränad modell (genom metodanrop till `create_model` i `face_embedding`-modulen), `UserRepository` med `Database`-objektet, `AdminController` med `UserRepository`- och `CryptoService`-objekten och slutligen `App` med `AdminController`-, `ProcessManager`-, `UserRepository`-, `CryptoService`- och `FaceEmbedder`-objekten.

`main`-modulen ansvarar slutligen för att starta applikationen med `App`-objektets Tkinter-mainloop samt stänga ner applikationen vid avslut.

4.1.5.6 process_manager

`ProcessManager`-klassen instansieras i `main`-modulen. `ProcessManager`-objektet används därefter för att instansiera `App`-, `LifecycleManager`- och `FaceCaptureService`-klasserna via DI. `ProcessManager` har ett eget interface, `IProcessManager`, som `LifecycleManager` och `FaceCaptureService` använder för att anropa `ProcessManager`-klassens metoder.

Genom sina metoder `add_process()` och `remove_process()` ansvarar `ProcessManager`-klassen för registrerade bakgrundsprocesser i anslutning till kameraprocessen samt resurshantering av dessa. Genom en intern lista kan klassen övervaka så att de skapade bakgrundsprocesserna stängs ner på ett korrekt sätt. `shutdown()`-metoden

eliminerar eventuellt kvarvarande kameraprocesser vid nedstängning av applikationen, för att undvika att de fortsätter ”leva” även efter att applikationen stängts ner.

`ProcessManager` initieras i ett with-block som en kontextmanager i `App`, vilket medför att skapade `ProcessManager`-objekt och registrerade bakgrundsprocesser kan avslutas säkert även vid oväntade fel.

Klassen har beroende till interfacet `IProcessManager` samt `multiprocessing.Process`-biblioteket för hantering av Process-objekt.

4.1.5.7 shared_types

Tre moduler ansvarar för typdefinitioner i applikationen genom olika enum, samt tillhandahåller en `from_str()`-metod som omvandlar en sträng till en faktisk typ. `detector_types` samlar de implementerade lösningarna för ansiktsdetektering och `embedding_types` samlar de tillgängliga dataseten som modellen tränas på för att skapa embeddings. Slutligen samlar `verification_types` de möjliga distansmåtten vid ansiktsverifiering. Med hjälp av fördefinierade enumvärden kan typfel lättare upptäckas och undvikas.

4.1.5.8 user_repository

`UserRepository`-klassen tillhör, liksom `Database`-klassen, Repository-lagret som ligger utanför MVC-strukturen. I denna modul ligger all applikationsspecifik SQL-logik samlad, vilket gör den till en central punkt för anrop till databasen. Syftet med modulen är att separera själva `Database`-klassen från konkreta SQL-queries. Alla anrop till databasen går genom någon av de fördefinierade metoderna i `UserRepository`-klassen, som i sin tur skickas vidare till det injicerade `Database`-objektet genom dess interface. Bland metoderna i klassen ingår CRUD-metoder för manipulering av användardata, att skapa `User`-objekt från hämtad användardata, uppdatering av access- och adminstatus samt hantering av autentisering och validering med embeddings. Modulen ansvarar främst för att skicka ny data till och returnera begärd data från databasen. Andra moduler anropar `UserRepository`-klassens metoder genom interfacet `IUserRepository`, vilket främjar låg koppling mellan modulerna. Alla SQL-queries är, liksom `Database`-klassens queries, parametriserade för ökad säkerhet.

5

Resultat

Projektet mynnade ut i en biometrisk autentiseringsapplikation som uppfyller de mål som sattes upp i planeringsfasen. En väl avvägd designstruktur tillsammans med noggrant utvalda bibliotek har bidragit till att uppnå dessa mål.

Applikationen har ett simpelt och överskådligt grafiskt användargränssnitt byggt med `Tkinter`, som är lätt att navigera. Ansiktsdetekteringen genomförs med ett av två möjliga val av detektor, Haar Cascades eller MTCNN. Applikationen skapar embeddings genom implementeringen av en förtränad djupinlärningsmodell för ansiktsigenkänning, FaceNet/InceptionResnetV1. Modellen har tränats på ett av två utvalda dataset, VGGFace2 eller CASIA-Webface.

Applikationen förutsätter att en databas har skapats innan uppstart. Med hänsyn till flexibilitet och säkerhet, har inga känsliga uppgifter gällande databasen hårdkodats i applikationen utan lämnas till systemägare att implementera. Autentiseringsuppgifter för åtkomst till databasen ska finnas specificerade i miljövariabler i en .env-fil, som hålls utanför repositoryt. Dessa ska överensstämma med de uppgifter som används vid skapandet av databasen. Vid uppstart etablerar applikationen en anslutning till databasen med hjälp av miljövariablerna och skapar därefter både tabeller och administratörsanvändare vid behov.

Från startsidan har användaren möjlighet att registrera sig eller logga in i systemet. En administratör kan även logga in via en separat inloggningssida, vilket ger tillgång till administratörsåtgärder på databasens innehåll. Systemet använder en form av tvåfaktorautentisering vid inloggning av vanliga användare, vilket ökar systemets övergripande säkerhet. En lyckad inloggning kräver både korrekta inloggningsuppgifter och en lyckad ansiktsverifiering. I verifieringssteget under den biometriska autentiseringen hämtas och dekrypteras den sparade embeddingen för den aktuella användaren med hjälp av dennes inloggningsuppgifter. En ny bild tas för att skapa en ny, tillfällig embedding. Därefter genomförs en avståndsberäkning mellan den sparade embeddingen och den tillfälliga embeddingen, som genererar ett värde som är större än eller lika med 0. Ett avståndsvärde närmre 0 innebär att vektorerna är mer lika. Avståndet jämförs med ett tröskelvärde. Ett avstånd mindre än eller lika med tröskelvärdet styrker identitet, medan motsatsen indikerar avsaknad av identitet. Avståndsberäkningen kan göras med en av två metoder, euklidisk avståndsmätning eller cosinusavstånd.

Lösenord hashas och foton och embeddings krypteras med hjälp av de utvalda meto-

derna som tillhandahålls av biblioteken `bcrypt` och `pycryptodome` (`Crypto.Cipher.AES`).

Interfaces används i stor utsträckning för att generera lös koppling, hög kohesion samt främja både utbytbarhet och testbarhet. Därmed kan exempelvis vald krypteringsteknik eller typ av databas enkelt substitueras, utan att resten av applikationen påverkas i stort. Dessutom kan implementationsdetaljer hållas dolda från anropande moduler.

DI används i stor utsträckning i applikationen, för att förhindra att moduler behöver skapa egna objekt av andra klasser och därmed få kännedom om implementationsdetaljer. De injicerade beroendena tvingar kommunikation mellan moduler att till största delen ske genom interfaces.

För att säkerställa att applikationen har en effektiv resurshantering under körning, har modulerna `ProcessManager` och `LifecycleManager` implementerats. Modulerna bidrar även till att processer hanteras korrekt vid nedstängning av hela systemet, för att förebygga minnesläckor.

Under utvecklingen har viss enhetstestning gjorts, för att säkerställa korrekt implementering av logik och felfri exekvering av enskilda moduler. Dessutom har manuell testning av det grafiska gränssnittet genomförts i hög grad under projektets gång, för att verifiera att applikationen visuellt representeras på ett önskvärt sätt, att GUI:t är responsivt samt kopplar användarinteraktioner till programlogik på ett korrekt sätt.
Manuell testning av moduler och applikationen i stort har även genomförts löpande under utvecklingen. I samtliga genomförda tester klarar applikationen av att särskilja en person från en annan i verifieringssteget, genom att avvisa en person under den biometriska autentiseringen om denne försöker logga in med korrekta inloggningsuppgifter från en annan användare. Detta tyder på att systemet kan förhindra förfalskningsattacker till en viss grad men ytterligare djupgående tester krävs för att utvärdera systemets övergripande motståndskraft mot denna typ av attack.
Tester visar även att applikationen kan ge avslag vid inloggning om en registrerad användare ändrar utseende eller om ljussättningen skiljer sig avsevärt från ljusdatan i den sparade embeddingen (False Negative). Därmed är brus i insamlad data ett relevant problem i systemet, vilket identifierades i det inledande kapitlet som ett av problemområdena inom ansiktsautentisering som kräver vidare utveckling.

En detaljerad användarguide finns att hitta i Appendix 2 under avsnitt B.1.

5.1 Systemöversikt

I denna översikt redovisas de centrala modulerna, med en kort sammanfattning av ansvarsfördelning, viktiga beroenden samt tillhörighet i MVC-strukturens olika lager. Interfaces samt moduler som innehåller delade typdefinitioner har utelämnats. De kan läsas om i under-underavsnitt 4.1.5.3, interfaces, samt under-underavsnitt 4.1.5.7, shared_types.

Tabell 5.1: Översikt av applikationens moduler

Modulnamn	**Ansvar**	**Beroenden**	**Lager**
AdminLoginView	Inloggningssida för administratörer. Tar emot och skickar användarinput samt tar emot valideringens resultat av användares inloggningsuppgifter. Navigerar till administratörsvyn.	Tkinter, Navigator, messagebox, IAdminService	View
AdminService	Funktioner för administratörer.	User, IUserRepository, ICryptoService	Controller
AdminView	Vy för administratörsåtgärder på registrerade användare i databasen.	Navigator, Tkinter, ttk, messagebox, IAdminService	View
App	Applikationens root-controller. Initierar huvudapplikationen och kopplar ihop controllers och services. Hanterar applikationens livscykel samt navigering mellan vyerna.	Tkinter, Navigator, AuthController, FaceVerifier, FaceCaptureService, DetectorModelType, LifecycleManager, IAdminService, ICryptoService, IFaceEmbedder, IProcessManager, IUserRepository	Controller
AuthService	Hanterar registrering, inloggning och validering av användare.	NumPy, OpenCV, IUserRepository, ICryptoService, IFaceCaptureService, IFaceEmbedder, IFaceVerifier, schemaläggningsfunktion	Controller

Modulnamn	**Ansvar**	**Beroenden**	**Lager**
camera_worker	Hanterar bakgrundsprocessen som ansvarar för kamera och ansiktsdetektion. Initierar detekteringsmodell, ett FaceProcessor- och ett FaceCapture-objekt. Returnerar resultat till huvudprocessen via en kö.	FaceDetector, FaceProcessor, FaceCapture, DetectorModelType, multiprocessing.Queue, IFaceDetector	-
CryptoService	Funktioner för hashning samt kryptering och dekryptering av data. Hanterar nycklar för säker lagring.	os, bcrypt, PyCryptodome/AES	Service
Database	Ansluter till en PostgreSQL-databas, skapar tabeller samt tillhandahåller CRUD-operationer.	os, psycopg2	Repository
FaceCaptureService	Startar och hanterar kameraprocessen. Kommunicerar resultat via en kö. Hanterar resurser och städning efter avslutade processer.	multiprocessing, DetectorModelType, camera_worker, IProcessManager	Service
FaceCapture	Visar kameran, ramar in och fångar ansikten samt returnerar resultat.	OpenCV, NumPy, time, FaceProcessor	Service
FaceDetector	Detekterar ansikten i bilder och returnerar BBox:ar för dessa. Valbar detektormodell.	OpenCV, NumPy, MTCNN, BBox, DetectorModelType	Service

Modulnamn	**Ansvar**	**Beroenden**	**Lager**
FaceEmbedder	Skapar embeddings från ansiktsbilder. Använder ett neuralt nätverk, FaceNet, förtränat på ett valbart ansiktsdataset, VGGFace2 eller CASIA-Webface.	OpenCV, NumPy, PyTorch, torchvision, facenet_pytorch, PIL, Embedding-ModelType	Service
FaceProcessor	Hanterar detektering, beskärning och utvidgning av marginal samt JPEG-kodning av ansiktsbilder.	OpenCV, NumPy, BBox, IFaceDetector	Service
FaceVerifier	Avgör identitet genom jämförelse av två embeddings med valbart mått för avståndsberäkning samt valbart tröskelvärde.	NumPy, scikit-learn, Metric	Service
LifecycleManager	Ansvarar för bakgrundsprocessers livslängd samt applikationens städfunktioner. Ser till att bakgrundsprocesser avslutas korrekt och att rätt funktioner exekveras vid nedstängning av applikationen.	multiprocessing, IProcessManager	-
LoginView	Hanterar navigering till inloggningssidorna för användare och administratörer.	Tkinter, Navigator	View

Modulnamn	**Ansvar**	**Beroenden**	**Lager**
main	Ansvarar för uppstart av applikationen. Initierar och injicerar beroenden, startar Tkinter-loopen samt skapar tabeller och administratörsanvändare i databasen.	os, dotenv, AdminController, Database, UserRepository, ProcessManager, CryptoService, FaceEmbedder, App, Embedding-ModelType, IAdminService, ICryptoService, IDatabase, IFaceEmbedder, IProcessManager, IUserRepository	Programstart
Navigator	Ansvarar för initiering av alla vyer och navigering i applikationen. En controller för applikationens vyer.	Tkinter, StartView, LoginView, RegisterView, UserLoginView, AdminLoginView, AdminView, IAdminService, IAuthService, exit-callback	Controller
ProcessManager	Hanterar applikationens bakgrundsprocesser i anslutning till kameran. Stänger ner och städar upp efter dessa vid avslut.	multiprocessing	-
RegisterView	Tar hand om registreringsprocessen för nya användare. Tar användarinput och initierar kameraprocessen för insamling av ansiktsdata.	Tkinter, messagebox, Navigator, IAuthService	View

Modulnamn	**Ansvar**	**Beroenden**	**Lager**
StartView	Start- och välkomstvy för applikationen. Hanterar navigering till registrerings- eller inloggningsvyn.	Tkinter, Navigator, exit-callback	View
User	Representerar en användare och dess data i systemet.		Model
UserLoginView	Inloggningssidan för vanliga användare. Tar emot användarinput, initierar ansiktsautentisering och hanterar resultat från kameraprocessen.	Tkinter, messagebox, Navigator, IAuthService	View
UserRepository	Hanterar databasåtgärder för användare och administratörer.	psycopg2, User, IDatabase	Repository

6

Diskussion

Systemet är utvecklat med två möjliga lösningar för ansiktsdetektion, Haar Cascades samt MTCNN. Utöver det finns två dataset implementerade som tränar modellen på att identifiera ansikten, VGGFace2 samt CASIA-Webface. Dessutom har möjligheten att välja ett av två avståndsmått integrerats. Förslagsvis kan framtida vidareutveckling av systemet dra nytta av en jämförelse mellan dessa implementationer för att bidra med relevanta insikter kring systemets prestanda, så att den mest effektiva lösningen kan fastställas. Den nuvarande produktprototypen möjliggör inte för den vanlige användaren att själv kombinera dessa lösningar efter eget önskemål, eftersom dessa val enbart går att nå på utvecklarnivå.

Då systemet krävde en framtidssäkrad lösning för biometrisk autentisering, valdes FaceNet-modellen för ansiktsdetektering. Initialt fanns en tanke om att implementera Dlib som potentiell lösning, eftersom Dlib kan erbjuda enklare integrering och generellt mindre belastning på systemet under körning. I litteraturstudien som gjordes under projektets inledande fas framhävdes dock FaceNet som det bättre valet gällande robusthet, precision och framtidsanpassning, varpå Dlib ratades. FaceNet anses dessutom vara en modernare lösning som är bättre anpassad till förändringar i ansiktsuttryck och ljusvariationer, är tränad på mer omfattande dataset och kan generera högkvalitativa embeddings. Lösningen var därmed bättre lämpad för projektets mål.

6.1 Etiska och säkerhetsrelaterade aspekter

Dataseten är valda utifrån ett etiskt perspektiv där seten innehåller data med en bred representation av personer av olika etnicitet, kön och åldrar men även en mängd olika variationer i position, ljus och bakgrunder. Systemet har därmed goda förutsättningar för att kunna urskilja olika ansikten i varierande miljöer, utan att diskriminera enskilda grupper. Utan att undersöka dataseten djupgående kan inga slutsatser dras kring hur väl det utvecklade systemet står sig mot diskriminering och bias i denna kontext, eller mot diskriminering gentemot individer som saknar utmärkande drag. De manuella testerna pekar på ett stabilt system utan tydlig diskriminering men ytterligare tester med fler testpersoner krävs för att verifiera systemets pålitlighet.

Då det utvecklade systemet enbart sparar data på en krypterad lokal databas, kan användare kan vara trygga i att personlig information stannar i systemet. Ingen data, krypterad eller okrypterad, delas med utomstående aktörer för exempelvis

analys, vilket stärker den personliga integriteten och förhindrar att personlig data hamnar utom kontroll för individen. Förslagsvis bör användare uppmärksammas på vilken data som sparas i databasen samt hur denna hanteras, för att öka användares förtroende till systemets säkerhet och samtidigt uppfylla det etiska perspektivet gällande transparens och samtycke som diskuterades inledningsvis.

Kryptering och lagring av foton är i sig inte en nödvändighet för applikationen och dess funktioner, eftersom foton inte behandlas ytterligare efter att en embedding har skapats. Ur ren säkerhetsaspekt kan det därmed argumenteras för att foton inte bör sparas alls, då läckta foton skulle kunna användas för att skapa nya embeddings och därmed utgöra en säkerhetsrisk. Då övergripande säkerhetsåtgärder har implementerats för att förhindra läckt data, beslutades det att behålla foton i databasen för att underlätta felsökning under utvecklingen av applikationen.

För att upprätthålla hög säkerhetsnivå i applikationen krävdes en kraftfull teknik för kryptering av embeddings. Informationen i en embedding består av en vektor i ett 512-dimensionellt rum, som används för avståndsmätning vid autentiseringstillfället. Avståndsmätningen krävde därmed att kryptering bevarade vektorns struktur i sin helhet, så att ursprungsvektorn kunde återskapas vid dekryptering. Traditionell hashning var därmed uteslutet, då denna metod medför att ursprungsvärdet inte går att återskapa och gör korrekta avståndsberäkningar på vektorerna omöjliga att genomföra. Kryptering med AES-256 GCM uppfyller dock kravet på att kunna återskapa den ursprungliga embeddingen, samtidigt som tekniken är säker, effektiv och välbeprövad. Under utvecklingen av applikationen undersöktes ytterligare ett annat innovativt alternativ för kryptering, nämligen en typ av teknik för enväga hashning som samtidigt bevarade embeddingens avståndsegenskaper. Hashningen medförde visserligen att datan inte gick att återskapa, men avståndsjämförselsen mellan två embeddings skulle fortfarande vara genomförbar på samma sätt som om de vore okrypterade. Lösningar för denna typ av kryptering var visserligen av intresse för projektet, men var vid utvecklingen av applikationen fortfarande under utveckling eller bakom en betalvägg och valdes därför bort för mer traditionell kryptering. Ur säkerhetsaspekt bedömdes kryptering med AES-256 GCM uppfylla de fastställda kraven för projektet gällande trygg och säker datalagring.

Vid eventuell distribution av applikationen bör den del av koden som syftar till att skapa tabeller och administratörskonto flyttas till en backend-/serversida eller exekveras som en del i installationsprocessen. Den vanlige användaren bör enbart ha tillgång till användarnära funktioner såsom registrering och inloggning, medan systemadministrativa funktioner ska hållas utanför användarupplevelsen. I praktiken uppnås detta till viss del redan genom att GUI:t tillhandahåller en separat inloggning för administratörer. Det kan dock ifrågasättas om även administrativa metoder bör flyttas till en backend-/serversida för att öka säkerheten och enbart ge systemägare tillgång till dessa, så att obehöriga helt kan hindras från att få tillgång till databasens innehåll och metoder för manipulering av datan genom GUI:t.

För att effektivt förhindra exempelvis spoofing, som diskuteras som en av svagheterna med ansiktsautentisering i det inledande kapitlet, krävs ytterligare bearbetning av applikationens säkerhetsrelaterade metoder innan eventuell distribution av sy-

stemet. Förslagsvis kan livevideon från kameran analyseras ett par ögonblick före och efter att bilden tas, för att på så vis upptäcka små förändringar i ansiktsuttrycket som därmed kan styrka att personen faktiskt sitter framför kameran i realtid. Djupdata från kameran skulle även kunna behandlas för att säkerställa att ett upptäckt ansikte inte i själva verket kommer från ett 2D-medium, exempelvis en film på en surfplatta. Dessa implementationer skulle stärka säkerheten ytterligare och ges därmed som ett direkt förslag för vidareutveckling.

6.2 Framtida vidareutveckling

Ytterligare vidareutveckling av applikationen skulle kunna innefatta direkt analys av ett ansikte utan krav på ytterligare inloggningsuppgifter vid autentisering. För att på ett effektivt sätt hitta ”närmsta granne” i en ständigt växande databas, kan logik för att analysera delar av en vektor för att snabbt centrera sökningen till ett mindre urval utvecklas för att optimera systemet. En sådan sökalgoritm bör ha låg tidskomplexitet för effektivisera programflödet. Inloggningsuppgifterna skulle istället kunna användas för att styrka användares identitet vid exempelvis ny fototagning eller ändring av inloggningsuppgifter. Att frångå inloggningsuppgifterna vid inloggningstillfället bör dock övervägas ur säkerhetsaspekt, eftersom det tar bort tvåfaktorautentiseringen och därmed sänker systemets övergripande säkerhet.

Systemet har utvecklats med hållbar utveckling i åtanke även över tid, både vad gäller komponenternas modularitet men även sociala aspekter och etiska frågor. Ett system för biometrisk autentisering bör vara framtidssäkrat och socialt hållbart, vilket i praktiken innebär att det bör finnas möjlighet att anpassa och vidareutveckla systemet så att individens rättigheter tas i beaktning utefter rådande etiska krav. Applikationens nuvarande utformning hindrar inte vidareutveckling från att möta de krav som ställs utefter framtida sociala eller etiska aspekter.

Teknik för ansiktsigenkänning kan i dagsläget inte betraktas som felfri, vilket gör en fortsatt generell vidareutveckling av tekniken nödvändig för att i framtiden kunna tillhandahålla en mer tillförlitlig säkerhetslösning för inloggningssystem som implementerar biometrisk autentisering. Inledningsvis diskuterades ett antal kritiska punkter som påverkar stabiliteten för ett system för autentisering genom ansiktsigenkänning. Efter projektets avslut kan det konstateras att den utvecklade applikationen till viss del speglar dessa problemområden.
Vad gäller både brus i insamlad data och intra-klassvariationer observerades det under de utförda testerna att den utvecklade prototypen har svårt att påvisa identitet om ljussättningen skiljer sig betydande eller om användarens utseende avviker för mycket från den sparade embeddningen. En ny frisyr eller glasögon kunde exempelvis ge avvisande utfall och systemet kan därmed konstateras vara känsligt mot dessa variationer. Ett så strikt system är oftast inte önskvärt och därmed föreslås ytterligare utveckling för att minimera andelen falska negativa utfall. Applikationens tröskelvärde kan modifieras (höjas) för att släppa igenom fler variationer i utseende och ljussättning, vilket dock kan resultera i att även andelen falska positiva utfall ökar. Ytterligare analys av tröskelvärdet med hjälp av metoder som VAL och FAR föreslås för att säkerställa ett välgrundat värde utifrån systemets förutsättningar.

Slutligen kan det antas att de övriga diskuterade problemområdena även de är relevanta utmaningar i den utvecklade applikationen och påverkar systemets precision. Systemet bör därmed genomgå ytterligare djupgående tester för att bedöma dess förmåga att hantera dessa begränsningar.

7

Slutsats

I den här kandidatuppsatsen har metoder för autentisering med hjälp av biometrisk ansiktsdata implementerats i en applikation för ansiktsigenkänning. Under projektets inledande fas utfördes en litteraturstudie, som därefter låg till grund för utvecklingen av en välstrukturerad applikation med utvalda tekniker vars huvudmål är att kunna särskilja en person från en annan med hjälp av en kamera. Slutprodukten av det utvecklade systemet utgör därmed en fungerande implementation av ansiktsautentisering. Pålitliga metoder för säker lagring samt hantering av lösenord och biometrisk data har integrerats som robusta säkerhetslösningar för ett system som tillämpar tvåfaktorautentisering.

Biometrisk data baseras på en individs ansiktsdrag och kräver inga kunskaps- eller ägandebaserade hjälpmedel i form av nycklar eller lösenord. Obehöriga aktörer får därmed svårare att göra intrång i systemet. Genom kryptering av användardata och säker lokal lagring utan extern åtkomst kan integriteten för systemets användare hållas hög. De tekniska utmaningarna och etiska riskerna som diskuterades inledningsvis har till viss del bemötts och utvärderats i förhållande till systemets prestanda men kräver ytterligare analys och utveckling för att åtgärda eller minimera identifierade svagheter.

Den utvecklade applikationen förhåller sig i så stor utsträckning som möjligt till designkonceptet MVC, för att möjliggöra en effektiv vidareutveckling samt öka utbytbarhet och testbarhet. Det prioriterades att göra applikationen så modulerbar som möjligt för att framtidssäkra systemet, så att exempelvis val av detektor, verifieringsprocess eller krypteringsteknik lätt kan ersättas med andra metoder och tekniska lösningar. Genom en tydlig uppdelning av ansvarsområden uppfyller applikationen även SoC, vilket förenklar enhetstestning då fel lättare kan isoleras. Användningen av interfaces och DI skapar låg koppling mellan modulerna, vilket innebär att målet att i huvudsak följa MVC-strukturen därmed har uppnåtts.

Ett enkelt grafiskt användargränssnitt har prioriterats, för att ge användare en lättförståelig applikation med smidig navigering.

En säker anslutning till en SQL-databas har integrerats i systemet. Metoder som syftar till att modifiera enskilda användares data har konstruerats på ett sådant sätt att SQL-injektion kan undvikas.

Det implementerade tröskelvärdet kan finjusteras för att uppnå önskad känslighet under verifieringsprocessen och därmed anpassa systemet till att vara mer eller mind-

re diskriminerande för att uppfylla specifika säkerhetskrav. Ett lägre tröskelvärde genererar striktare säkerhet, vilket även gör systemet mindre tillåtande för variationer i utseende.

Applikationen erbjuder ett till synes stabilt system för ansiktsigenkänning och ansiktsautentisering som enkelt kan byggas på med metoder för autentisering med hjälp av annan biometrisk data. Föreslagna områden för potentiell vidareutveckling är autentisering via fingeravtryck, röstigenkänning eller skanning av ögats iris. I kombination med flera autentiseringsvariabler, kan systemet stärkas och dess säkerhet höjas ytterligare vilket kan öka dess tillförlitlighet.

Litteraturförteckning

[1] R. W. Shirey, "Internet security glossary, version 2." `https://www.rfc-editor.org/rfc/rfc4949`, August 2007. RFC 4949, page 101, definition of Dictionary Attack.

[2] A. Jain, A. Ross, and S. Prabhakar, "An introduction to biometric recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 4–20, 2004.

[3] M. Killioğlu, M. Taşkiran, and N. Kahraman, "Anti-spoofing in face recognition with liveness detection using pupil tracking," in *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pp. 000087–000092, 2017.

[4] S. Beery, G. V. Horn, and P. Perona, "Recognition in terra incognita," *CoRR*, vol. abs/1807.04975, 2018.

[5] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pp. 67–74, 2018.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[7] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, 2001.

[8] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[9] S. Zhang, C. Chi, Z. Lei, and S. Z. Li, "Refineface: Refinement neural network for high performance face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 4008–4020, 2021.

[10] A. K. J. Stan Z. Li and J. Deng, *Handbook of Face Recognition*. Springer International Publishing, 2023.

[11] J. Lv, X. Shao, J. Xing, C. Cheng, and X. Zhou, "A deep regression architecture with two-stage re-initialization for high performance facial landmark detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3691–3700, 2017.

[12] W. Wu, C. Qian, S. Yang, Q. Wang, Y. Cai, and Q. Zhou, "Look at boundary: A boundary-aware face alignment algorithm," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2129–2138, 2018.

[13] Z.-H. Feng, J. Kittler, M. Awais, and X.-J. Wu, "Rectified wing loss for efficient and robust facial landmark localisation with convolutional neural networks," *International Journal of Computer Vision*, vol. 128, pp. 2126–2145, Sep 2020.

[14] H. Jin, S. Liao, and L. Shao, "Pixel-in-pixel net: Towards efficient facial landmark detection in the wild," *International Journal of Computer Vision*, vol. 129, pp. 3174–3194, Dec 2021.

[15] X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Z. Li, "Face alignment across large poses: A 3d solution," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 146–155, 2016.

[16] H. Imaoka, H. Hashimoto, K. Takahashi, A. F. Ebihara, J. Liu, A. Hayasaka, Y. Morishita, and K. Sakurai, "The future of biometrics technology: from face recognition to related applications," *APSIPA Transactions on Signal and Information Processing*, vol. 10, p. e9, 2021.

[17] N. Provos and D. Mazieres, "A future-adaptable password scheme," in *USENIX annual technical conference, FREENIX track*, vol. 1999, pp. 81–91, 1999.

[18] National Institute of Standards and Technology (NIST), "Announcing the Advanced Encryption Standard (AES)," Tech. Rep. FIPS PUB 197, U.S. Department of Commerce, November 2001. Federal Information Processing Standards Publication.

[19] M. Dworkin, "Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac," Tech. Rep. NIST Special Publication 800-38D, National Institute of Standards and Technology, November 2007. U.S. Department of Commerce.

[20] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *CoRR*, vol. abs/1411.7923, 2014.

A

Appendix 1 - Matematiska koncept inom ansiktsigenkänning

A.1 Triplet Loss

Vid träning av en modell med Triplet Loss-funktionen, sägs en embedding representeras av

$$f(x) \in \mathbb{R}^d \tag{A.1}$$

där x är bilden och f är funktionen som bäddar in bilden i ett d-dimensionellt euklidiskt rum. Embeddingen begränsas ytterligare genom att låta den leva på en d-dimensionell hypersfär och tvinga den till att uppfylla $\|f(x)\|_2 = 1$. På så vis kan alla ansikten projiceras på ytan till en d-dimensionell enhetssfär och kan därmed jämföras [6].

För att korrekt skapa detta embeddingrum tillämpas förlustfunktionen Triplet Loss för att träna modellen. I metoden används tre olika typer av bilder, där en är den ursprungliga bilden, x_i^a (”anchor”), vilken fungerar som utgångspunkt för jämförelse av bilderna. Den andra bilden betecknas x_i^p (positiv) och representerar en bild på samma person som återfinns i x_i^a. Slutligen används en bild som representerar en annan person, vilken betecknas x_i^n (negativ). Personen i x_i^n har alltså inte samma identitet som personen i x_i^a. Då olika identiteter ska kunna skiljas åt i embeddingrummet, kan dessa tre bilder därefter användas för att lära modellen relationen mellan ansiktena utöver att klassificera dem. För korrekt identifiering, är det viktigt att avstånden mellan ”anchor” och positiv-paren är kortare än mellan ”anchor” och negativ-paren.

$$\|f(x_i^a) - f(x_i^p) + \alpha\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 \,, \quad \forall (f(x_i^a), f(x_i^a), f(x_i^a)) \in T \tag{A.2}$$

Förlustfunktionen definieras såhär, där α är den marginal som ger separationen mellan de positiva och negativa paren.

$$L = \sum_i^N [\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha]_+ \tag{A.3}$$

A.2 Tröskelvärde

Låt $f(x_i), f(x_j) \in \mathbb{R}^d$ vara två embeddings, där L_2-avståndet (euklidiskt avstånd) mellan dem är:

$$\|f(x_i) - f(x_j)\|_2 = \sqrt{\sum_{k=1}^{N}(f_k(x_i) - f_k(x_j))^2} \tag{A.4}$$

$f_k(x_i)$ motsvarar k:te komponenten i embeddingvektorn $f(x_i)$.

FaceNet använder sig utav kvadratiskt L_2-avstånd i det euklidiska rummet för att mäta likheten mellan två ansikten. Avståndet mellan två embeddings x_i och x_j kan skrivas som:

$$D(x_i, x_j) = \|f(x_i) - f(x_j)\|_2^2 \tag{A.5}$$

För att kunna avgöra om avståndet mellan två embeddings representerar identitet eller två olika personer, används ett tröskelvärde d som kan beskrivas som:

$$D(x_i, x_j) \leq d \rightarrow \text{Samma person} \tag{A.6}$$

$$D(x_i, x_j) > d \rightarrow \text{Annan person} \tag{A.7}$$

Alla ansiktspar (i, j) med samma identitet betecknas $\mathcal{P}_{same}$, medan par med olika identitet betecknas $\mathcal{P}_{diff}$. Mängden par med samma identitet som även klassificerats korrekt, det vill säga antalet korrekta matchningar (*True Accept*), definieras som:

$$\text{TA}(d) = \{(i, j) \in \mathcal{P}_{same} \mid D(x_i, x_j) \leq d\} \tag{A.8}$$

Mängden par med olika identitet som felaktigt har klassificeras som samma, det vill säga antalet felaktigt matchningar (*False Accept*), definieras som:

$$\text{FA}(d) = \{(i, j) \in \mathcal{P}_{diff} \mid D(x_i, x_j) \leq d\} \tag{A.9}$$

För att mäta ett systems prestanda med hjälp av dessa indelningar, beräknas andelen korrekta igenkänningar bland alla positiva par (*Validation Rate*) samt den andel som motsvarar hur ofta modellen felaktigt tror att två olika personer är samma (*False Acceptance Rate*). Dessa definieras:

$$\text{VAL}(d) = \frac{|\text{TA}(d)|}{|\mathcal{P}_{same}|} \tag{A.10}$$

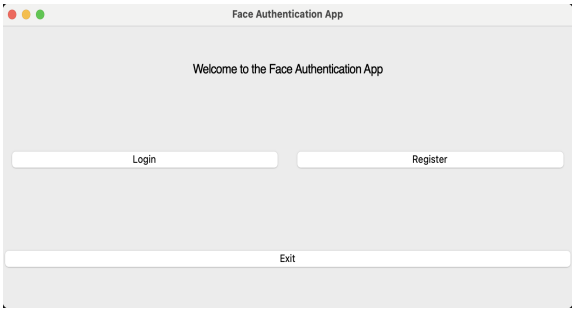
$$\text{FAR}(d) = \frac{|\text{FA}(d)|}{|\mathcal{P}_{diff}|} \tag{A.11}$$

B

Appendix 2 - Applikationen

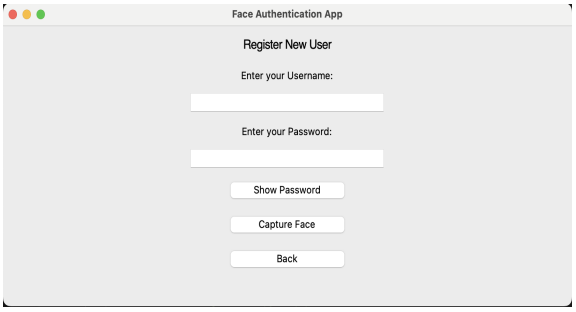
B.1 Användarguide

Startskärm:
Vid uppstart möts användaren av en startvy där denne kan välja att logga in (’Login’) eller registrera sig (’Register’).



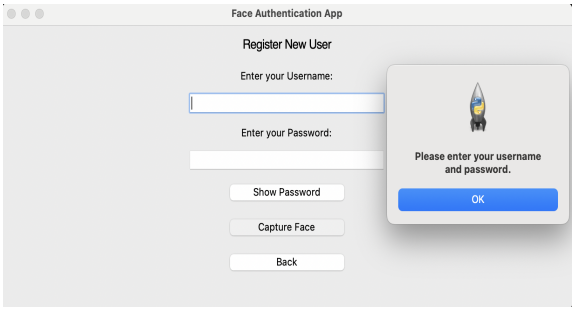
Figur B.1: StartView

Registrera ny användare:
Efter ett klick på ’Register’ tas användaren till applikationens registreringsskärm.



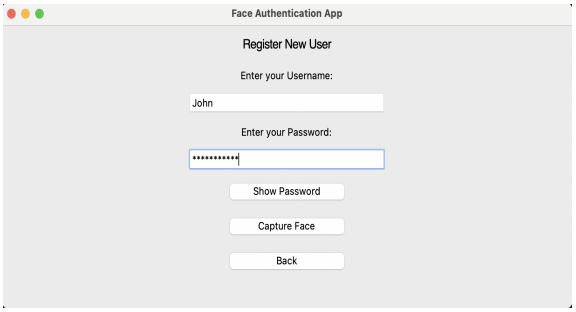
Figur B.2: RegisterView

Användaren måste välja ett användarnamn och lösenord. Ett tomt fält hindrar användaren från att gå vidare i registreringen.



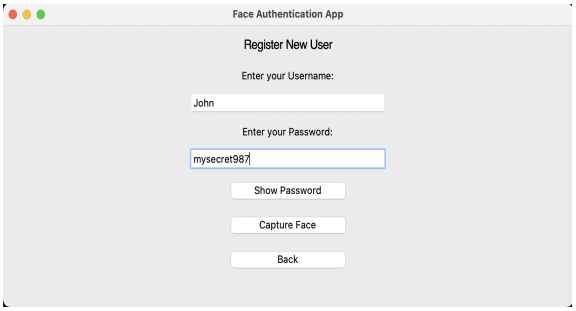
Figur B.3: RegisterView

Användaren väljer ett användarnamn och lösenord.



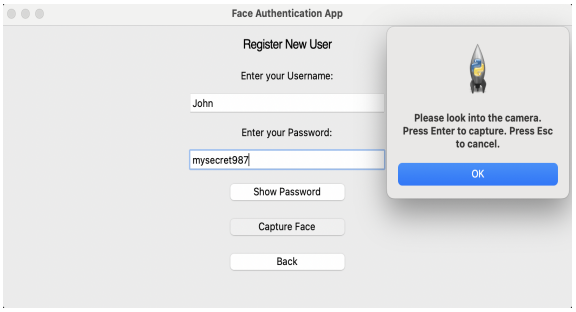
Figur B.4: RegisterView

Lösenordet kan synliggöras med ett knapptryck på 'Show Password'.



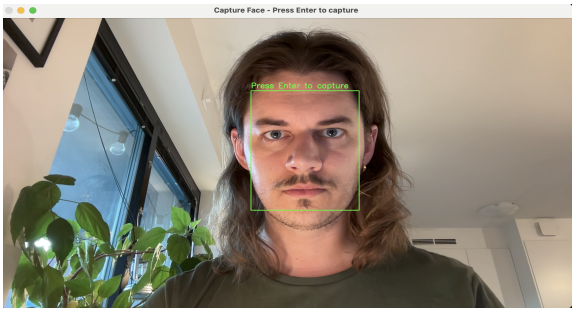
Figur B.5: RegisterView

Användaren trycker på 'Capture Face', vilket startar kameran och tar registreringsprocessen vidare.



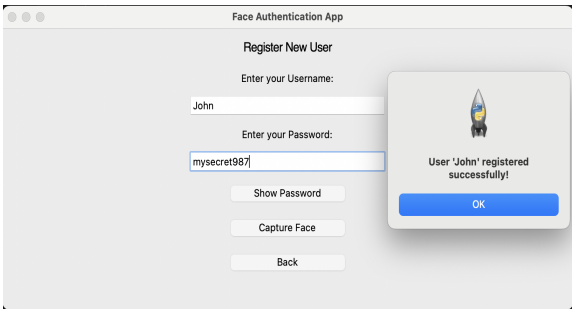
Figur B.6: RegisterView

Kameran startar och ger användaren direkt visuell feedback var i livefeeden ett ansikte detekteras i form av en grön rektangel som ramar in ansiktet.



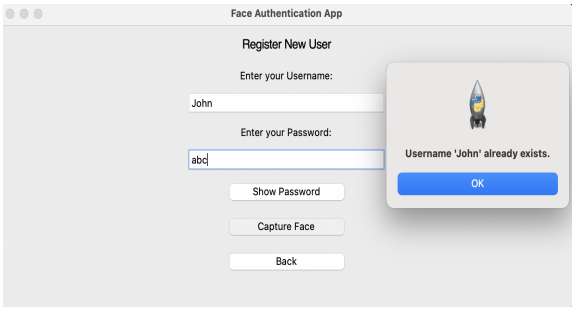
Figur B.7: Kameraprocessen startas för registrering

Ett knapptryck på 'Enter'-tangenten tar en bild. Om fotograferingen lyckas avslutas registreringsprocessen och den nya användaren finns därefter registrerad i databasen.



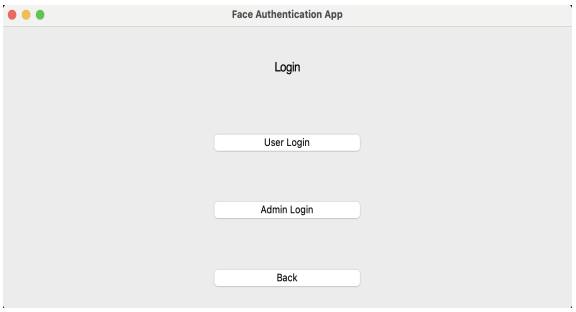
Figur B.8: RegisterView

Användarnamn måste vara unika. En ny användare kan därmed inte registreras med ett användarnamn som redan finns registrerat på en annan användare i databasen.



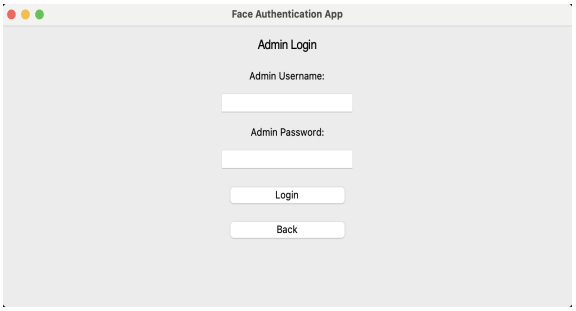
Figur B.9: RegisterView

Loginvy:
En användare kan antingen logga in som vanlig användare ('User Login') eller som administratör ('Admin Login').



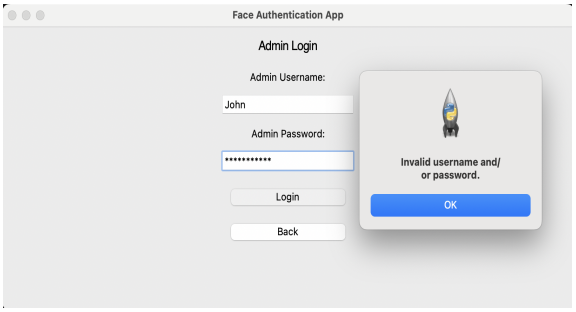
Figur B.10: LoginView

Logga in administratör:
Administratörer loggar in med behöriga användaruppgifter.



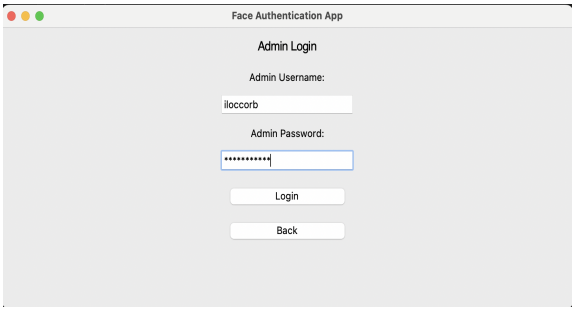
Figur B.11: AdminLoginView

Användare utan administratörsbehörighet hindras från att logga in till Adminvyn.



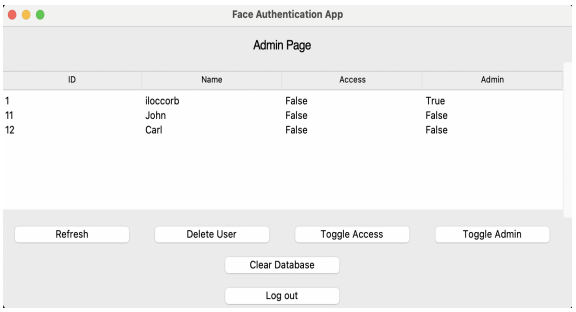
Figur B.12: AdminLoginView

En administratör loggar in med korrekta användaruppgifter.



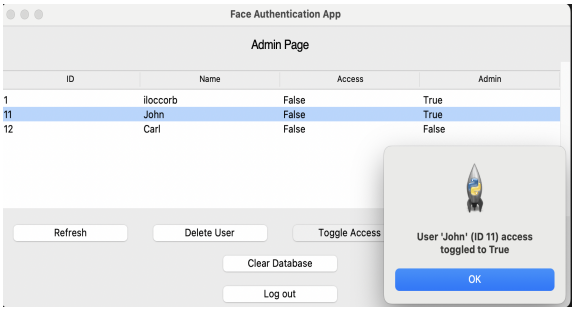
Figur B.13: AdminLoginView

Administratörsvy:
Inloggade administratörer kan manipulera innehållet i databasen.



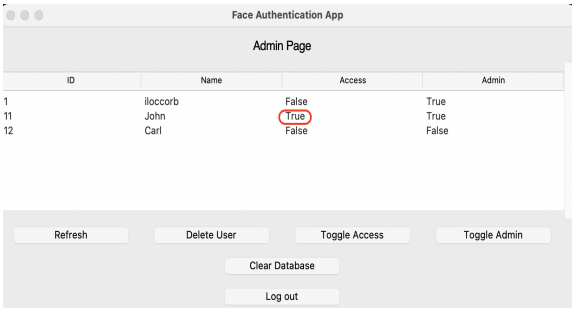
Figur B.14: AdminView

Bevilja eller neka access:
Enskilda användare kan ges eller fråntas access genom att markera användaren och klicka på 'Toggle Access'.



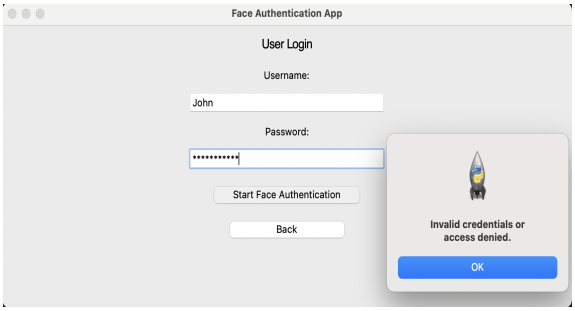
Figur B.15: AdminView

Listan med användare uppdateras omedelbart vid förändrad status.



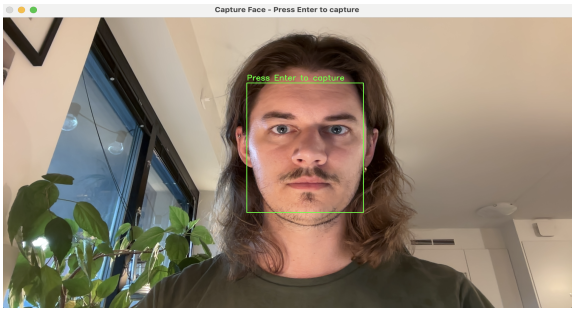
Figur B.16: AdminView

Logga in användare:
Vanliga användare loggar in från 'User Login'-vyn. En användare som saknar access nekas inloggning.



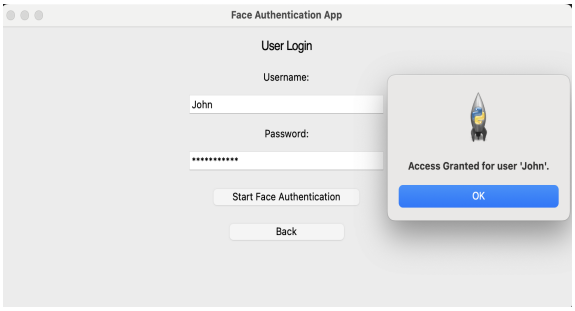
Figur B.17: UserLoginView

Användare med access tas vidare till kameravyn, där en ny bild tas med ett knapptryck på 'Enter'-tangenten.



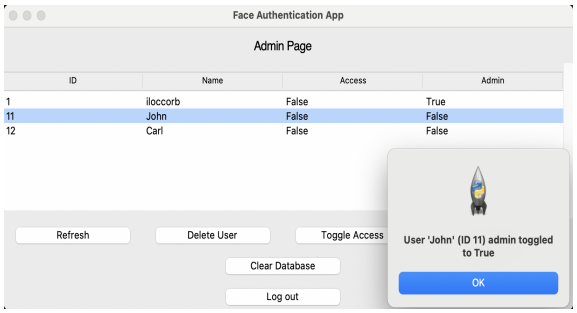
Figur B.18: Kameraprocessen startas för autentisering

En lyckad autentisering ger ett positivt gensvar och användaren ges access.



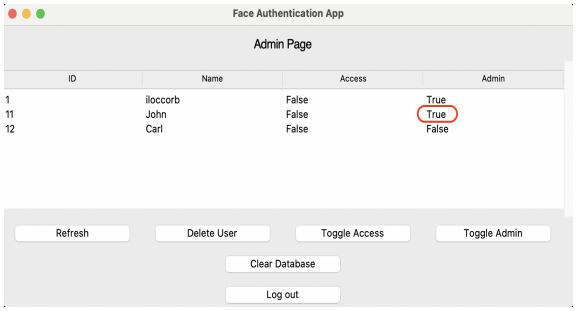
Figur B.19: UserLoginView

Bevilja eller neka adminbehörigheter:
Administratörer kan ge andra användare administratörsbehörigheter via Admin-vyn, genom att markera användaren och trycka på 'Toggle Admin'-knappen.



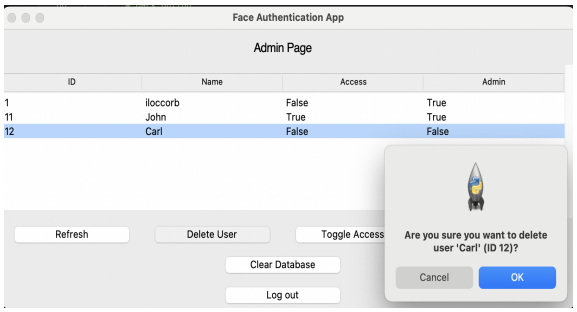
Figur B.20: AdminView

Listan med användare uppdateras omedelbart efter lyckad statusändring.



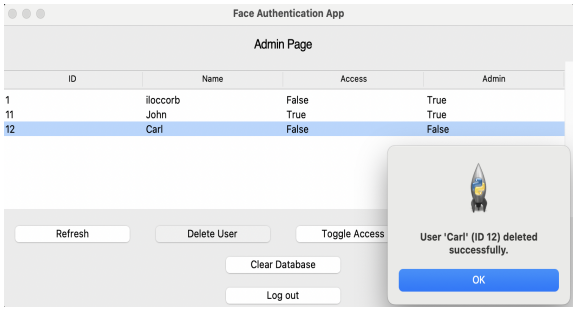
Figur B.21: AdminView

Radera enskilda användare:
En användare kan raderas ur databasen genom att markera användaren och trycka på 'Delete User'-knappen.



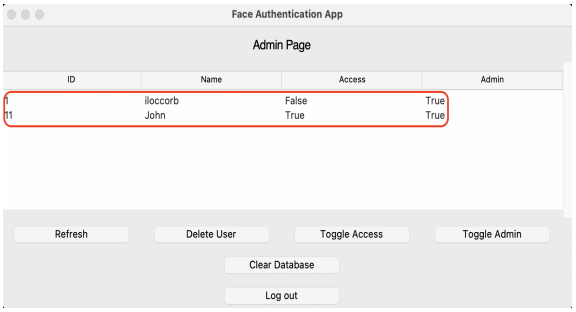
Figur B.22: AdminView

En lyckad radering bekräftas med ett meddelande.



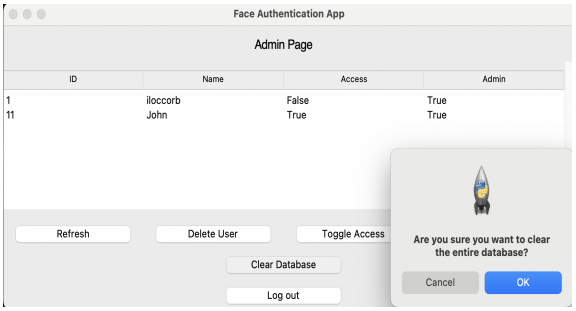
Figur B.23: AdminView

Listan med användare uppdateras efter lyckad radering.



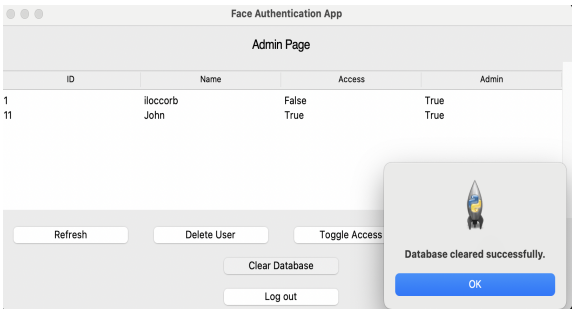
Figur B.24: AdminView

Radera alla användare:
Databasen kan helt tömmas på innehåll med ett knapptryck på 'Clear Database' och därefter 'OK'.



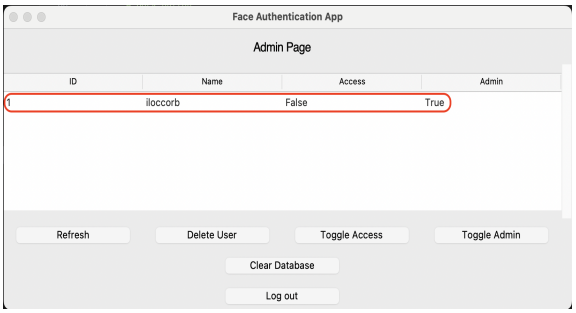
Figur B.25: AdminView

Ett meddelande bekräftar att innehållet i databasen har raderats.



Figur B.26: AdminView

Den fördefinierade administratörsanvändaren återskapas när databasen töms på innehåll, för att säkerställa att åtkomst med administratörsbehörigheter till databasen via applikationen alltid är möjlig.



Figur B.27: AdminView

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

GÖTEBORGS
UNIVERSITET

CHALMERS