TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Virumaa kolledž

AUTOMAATNE KOHALOLEK ARVUTINÄGEMISE ABIL

Automatic attendance using computer vision

TELEMAATIKA JA ARUKAD SÜSTEEMID ÕPPEKAVA LÕPUTÖÖ

Üliõpilane: Alexander Chelpanov
Üliõpilaskood: 182730EDTR
Juhendaja: Natalja Ivleva, lektor

Kohtla-Järve, 2023

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"...." ..... 20.....

Autor: .....
/ allkiri /

Töö vastab rakenduskõrgharidusõppe lõputööle/magistritööle esitatud nõuetele
"...." ..... 20.....

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud
"...." ..... 20.....

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

LIHTLITSENTS LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS JA REPRODUTSEERIMISEKS

Mina, Alexander Chelpanov (sünnikuupäev:17.06.1986)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

„Automaatne kohalolek arvutinägemise abil", mille juhendaja on Natalja Ivleva,

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil, kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu, kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

TalTech Inseneriteaduskond Virumaa kolledž

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Alexander Chelpanov, 182730EDTR

Õppekava, peaeriala: EDTR17/18, Telemaatika ja arukad süsteemid, Protsesside automatiseerimine

Juhendaja(d): lektor, Natalja Ivleva, natalja.ivleva@taltech.ee

Lõputöö teema:

(eesti keeles) Automaatne kohalolek arvutinägemise abil

(inglise keeles) Automatic attendance using computer vision

Lõputöö põhieesmärgid:

1. Automaatse kohalolekuarvestuse süsteemi loomine.

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.	Näotuvastuse raamatukogudega tutvumine	1.02.2023
2.	Rakenduste loomise tehnoloogiate uurimine	1.03.2023
3.	Andmebaasi loomine	15.03.2023
4.	Veebirakenduse loomine	1.04.2023
5.	Veebirakenduse testimine	20.04.2023
6.	Lõputöö kirjutamine (kirjalik osa)	1.05.2023

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** "22" mai 2023a

Alexander Chelpanov: ..... "....."..... 20.....a
/allkiri/

Natalja Ivleva: ..... "....."..... 20.....a
/allkiri/

Žanna Gratšjova: ..... "....."..... 20.....a
/allkiri/

SISUKORD

EESSÕNA

Käesolev lõputöö on valminud TalTech Virumaa kolledži tellimusel. Lõputöö teema pakkus välja lektor Natalja Ivleva.

Lõputöö valmimisele on aidanud telemaatika ja arukate süsteemide lektor Natalja Ivleva.

Võtmesõnad: InsightFace API, Fastapi, Streamlit, PostgreSQL, ORM, veebirakendus, arvutinägemine, diplomitöö.

LÜHENDITE JA TÄHISTE LOETELU

- **OpenCV** (Open Source Computer Vision) on avatud lähtekoodiga arvutinägemise ja masinõppimise tarkvara teek
- **Streamlit** on programmeerimiskeelel Python põhinev veebirakenduste loomiseks mõeldud vabavararaamistik
- **InsightFace** on integreeritud Pythoni teek 2D ja 3D näoanalüüsiks
- **FastAPI.** Pythonis kirjutatud API veebiraamistik
- **SQL** Structrured Query Language, Struktuurpäringukeel
- **ORM** Object-Relational Mapping, Objekt-relatsiooniline kaardistamine
- **API** Application programming interface, Rakendusliides

SISSEJUHATUS

Inimene saab põhiosa välismaailma puudutavast informatsioonist kasutades nägemismeelt ning seejärel töötleb andmeid visuaalse informatsiooni analüüsi- ja tõlgendamisaparaadi abil. Seetõttu tekib küsimus antud protsessi automatiseerimise võimaluse kotha eri tegevuste tehnilise lahendusena.

Huvitavaks teemaks on masinnägemise kasutamine inimese tuvastamiseks ja autentimiseks. Selliseid lahendusi saab kasutada erinevates organisatsioonides mitmete ülesannete täitmiseks: personali lubamine suletud alale, külastajate registreerimine, hädaolukordade ennetamine j.n.e. Antud lahendused töötavad sarnasel põhimõttel, kuid igaüks neist nõuab mõne biomeetrilise tunnuse kasutamist. See on kasutajale mugav, et autentimiseks pole juurdepääsukaarte vaja, piisab inimese kohalolekust.

Seega on asjakohane masinnägemisega töötamist võimaldavate arvutitehnoloogiate uurimine. Neid tehnoloogiaid saab edukalt rakendada erinevate probleemide lahendamiseks paljudes inimtegevuse valdkondades.

Töö eesmärgiks on välja töötada veebirakendus, mis annaks õpetajatele võimaluse õpilaste kohaloleku üle arvet pidada. Süsteem peab isiku tuvastama läbi kaamera ja fikseerima tema kohaloleku aja.

1 RAKENDUSE ÜLEVAADE

1.1 Näotuvastuse teekidega tutvumine

Python programmeerimiskeele valiku tegi autor selle laia kasutamisala ja lihtsuse tõttu. Pythoni keeles on mitu näotuvastuse teeki, millel on omad eelised ja puudused. Allpool on toodud lühike ülevaade mõnest neist:

1) OpenCV:

OpenCV on avatud lähtekoodiga teek, mis on mõeldud arvutinägemise, pilditöötluse ja üldkasutatavate arvutuslike algoritmide jaoks. OpenCV-l on näotuvastuse moodul, mis kasutab Haari kaskaade ja sügavaid neurovõrkusid (DNN). [1]

Eelised:

- Avatud lähtekood ja laialdane kasutusala tööstusharus.
- Ulatuslik tugi pilditöötluse ja arvutinägemise algoritmidele.
- Lihtne kasutada ja integreerida teiste teekidega.

Puudused:

- Ei ole spetsialiseeritud näotuvastuse teek ja võib vajada täiendavat seadistamist.
- Võib olla vähem täpne võrreldes mõnede teiste teegiga.

2) Dlib:

Dlib on kaasaegne avatud lähtekoodiga C++ teek, mis pakub ka Pythoni sidemeid. Dlib pakub võimsaid vahendeid näotuvastuseks, kasutades suunatud gradientide histogramme (HOG) ja sügavaid konvolutsioonilisi neurovõrke (CNN). [2]

Eelised:

- Avatud lähtekood ja aktiivne tugi.
- Kasutab masinõppe mudeleid nagu HOG ja CNN näo tuvastamiseks.

Puudused:

- Algajatele võib olla keeruline seadistada ja integreerida.
- Sõltub C++ teegist oma töös.

3) DeepFace:

DeepFace - see on avatud lähtekoodiga teek, mis on arendatud TensorFlow'i ja Keras'i baasil ning kasutab eelnevalt treenitud sügavaid närvivõrke näotuvastuseks.

DeepFace'il on hea täpsus ja seda saab kasutada nägude võrdlemiseks ja kasutajate autentimiseks. [3]

Eelised:

- Põhineb sügavatel närvivõrkudel, mis tagab kõrge näotuvastuse täpsuse.
- Toetab kasutajate autentimist.
- Kasutab eelnevalt arendatd mudeleid, mis võimaldab kiiret rakendamist ja integreerimist.

Puudused:

- Võib olla aeglane suurte piltide või video töötlemisel.
- Sõltub TensorFlow'ist ja Keras'ist, mis võib keerulisemaks muuta paigaldamist ja seadistamist.

4) Insightface:

InsightFace - see on avatud lähtekoodiga projekt, mis kasutab sügavat õppimist näotuvastuseks ning rakendab eelnevalt arendatud mudeleid, nagu ResNet ja MobileFaceNet. See teek pakub kõrget näotuvastuse täpsust ja võimaldab tuvastada näo võtmetäppe. [4]

Eelised:

- Kõrge näotuvastuse täpsus ja võime tuvastada näo võtmetäppe.
- Kasutab eelnevalt treenitud mudeleid, nagu ResNet ja MobileFaceNet.
- Avatud lähtekood ja aktiivne arendajate kogukond.

Puudused:

- Võib olla keeruline seadistada ja integreerida teiste Pythoni teekidega.
- Sõltub MXNet'ist, mis võib raskendada paigaldamist ja seadistamist.

Lõputöö raames valiti InsightFace API, kuna see pakub kõrget näotuvastuse täpsust tänu sügavatele närvivõrkudele ja eelnevalt arendatud mudelitele. InsightFace on üks juhtivaid teeki selles valdkonnas. API võimaldab hõlpsalt integreerida InsightFace funktsioone rakendusse.

1.2 Rakenduste loomise tehnoloogiate ülevaade

1.2.1 Andmebaas

Andmete salvestamiseks valis autor PostgreSQL. PostgreSQL on võimas avatud lähtekoodiga objektreklaamisüsteem (ORDBMS), mis pakub usaldusväärset ja tõhusat andmete salvestamist teie rakenduste ja süsteemide jaoks. PostgreSQL arendavad ja

toetavad arendajate kogukonnad, kellel on pikaajaline kogemus ning maine usaldusväärsuse ja jõudluse osas. [5]

Eelised:

- Tulemuslikkus: PostgreSQL pakub kõrget jõudlust tänu täiustatud päringute optimeerimisele, indekseerimisele ja paralleelsele töötlusele. See võimaldab tõhusat andmebaasi jõudlust isegi suurte andmekogumite korral.
- Skaleeritavus: PostgreSQL'i saab hõlpsasti skaleerida, võimaldades andmebaasi mahutada rohkem andmeid ja suurendada samal ajal jõudlust.
- Usaldusväärsus: PostgreSQL on tuntud oma usaldusväärsuse poolest, pakkudes turvalist andmete salvestamist, tehingute terviklikkust ja taastamismehhanisme andmete kadumise ennetamiseks.
- Avatud lähtekood: PostgreSQL on avatud lähtekoodiga süsteem, mis tähendab, et seda arendab ja toetab suur kogukond, mis tagab pideva arengu, turvapaikade kiire lahendamise ja uuenduste kättesaadavuse.

1.2.2 Serveri osa

Autor valis andmebaasiga suhtlemiseks FastAPI - kaasaegse, kõrge jõudlusega raamistiku veebirakenduste ja API-de loomiseks Pythoni keeles. See põhineb standarditel, nagu OpenAPI ja JSON Schema, ning kasutab Pythoni asünkroonseid võimalusi Starlette ja Pydantic teegi abil andmete töötlemise ja valideerimise jaoks. [6]

Eelised:

- Kiirus: FastAPI tagab kõrge jõudluse tänu asünkroonsele lähenemisviisile ja koodi optimeerimisele, muutes selle üheks kiireimaks Pythoni raamistikuks.
- Kasutusmugavus: FastAPI on väga intuitiivne ja lihtne kasutada, võimaldades arendajatel kiiresti luua API-sid ja veebirakendusi minimaalse koodiga ja erilise keerukuseta.
- Automaatne dokumentatsioon: FastAPI genereerib automaatselt teie API dokumentatsiooni OpenAPI standardi abil, mis hõlbustab teie API testimist, integreerimist ja hooldamist.
- Andmete valideerimine: FastAPI kasutab Pydanticut automaatseks sisendi- ja väljundandmete valideerimiseks, tagades tüübi “range” kontrolli ja vähendades vigade arvu teie rakenduses.

- Asünkroonne tugi: FastAPI toetab asünkroonset programmeerimist, kasutades async ja await võtmesõnu, võimaldades luua skaldeeritavaid jõudlusega rakendusi.
- Lihtne integreerimine: FastAPI integreerub lihtsalt teiste tööriistade ja raamatukogudega, nagu SQLAlchemy, Tortoise ORM ja teised, võimaldades teil ühendada erinevaid komponente oma rakendusega. [12]

1.2.3 Kasutajaliides

Veebiliidese loomiseks kasutas autor Streamlit - Pythoni põhine vahend veebirakenduste ja interaktiivsete juhtpaneelide loomiseks minimaalse koodi mahuga. See tööriist on mõeldud veebirakenduste arendusprotsessi lihtsustamiseks, eriti andmete töötlemise ja visualiseerimise jaoks. Streamlit võimaldab arendajatel luua veebirakendusi ilma eelmiste tehnoloogiate, nagu JavaScript, HTML või CSS sügava tundmseta.

Eelised:

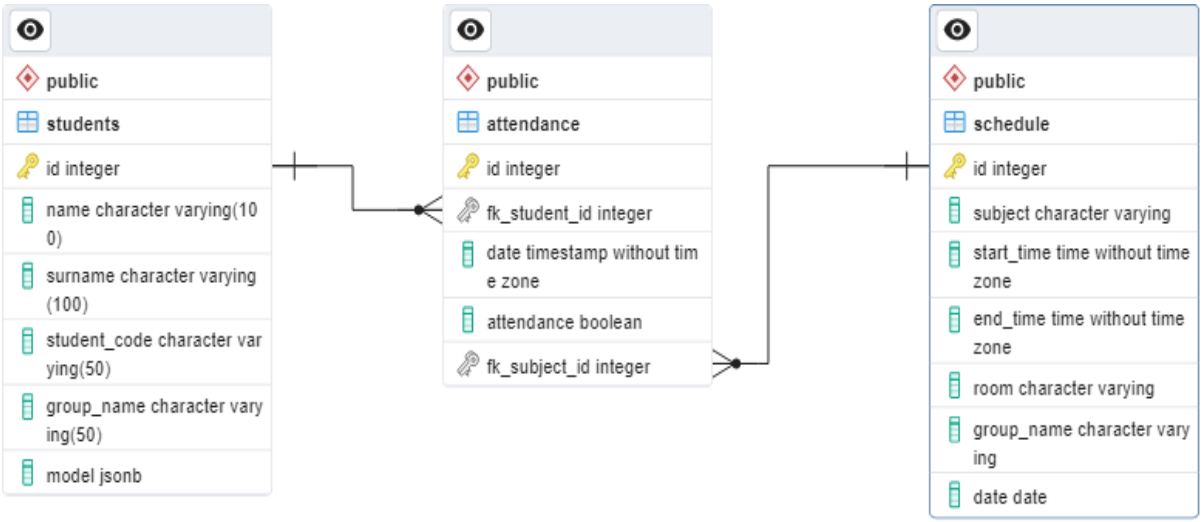
- Kiire prototüüpimine: Streamlit pakub lihtsat ja intuitiivset API-d veebirakenduste ja juhtpaneelide loomiseks, võimaldades kiiresti luua prototüüpe ja eksperimenteerida andmetega.
- Lihtne kasutamine: Streamlit on väga kasutajasõbralik, isegi neile, kellel puudub kogemus veebitehnoloogiatega töötamisel. See võimaldab arendajatel keskenduda andmete töötlemisele ja interaktiivsete visualiseeringute loomisele, selle asemel et õppida keerulisi raamistikke ja programmeerimiskeeli.
- Automaatne liidese värskendamine: Streamlit värskendab teie veebirakenduse liidest automaatselt, kui koodis tehakse muudatusi, hõlbustades iteratiivset arendus- ja testimisprotsessi.

Flask ja Django on traditsioonilisemad veebirakenduste raamistikud Pythonis. Need pakuvad suuremat paindlikkust ja kontrolli teie veebirakenduse üle, kuid nõuavad ka teadmisi veebitehnoloogiatest, nagu HTML, CSS ja JavaScript.

2 RAKENDUSE LOOMINE

2.1 Andmebaasi skeem

Andmebaasi skeem koosneb kolmest tabelist: students, attendance ja schedule. Tabel students sisaldab andmeid tudengite kohta. Tabel attendance sisaldab kohaloleku andmeid. Tabel schedule sisaldab graafiku andmeid, mis laaditakse väljastpoolt. (Joonis 2.1)



Joonis 2.1 Andmebaasi skeem

2.2 Serveri osa

2.2.1 SQLAlchemy ORM

SQLAlchemy - populaarne Pythoni teek, mis pakub tööriistu andmebaasidega töötamiseks, kasutades SQL ja objekt-relatsioonilist kaardistamist (ORM). See lihtsustab andmebaasi toimingute tegemist ja võimaldab arendajatel suhelda andmebaasidega Pythoni objektide ja SQL päringute abil. [8]

SQLAlchemy ORM: SQLAlchemy ORM on osa SQLAlchemy teegist, mis pakub kõrgema taseme liidest andmebaasi tabelite kaardistamiseks Pythoni klassidele ja lihtsustab andmebaasitoiminguid, abstraktiseerides SQL-i üksikasju.

Deklareeriv kaardistamine: SQLAlchemy ORM võimaldab määratleda andmebaasi skeemi ja kaardistamist, kasutades Pythoni klasse ja deklaratiivset süntaksit. Need klassid esindavad andmebaasi tabeleid ja määratlevad nende vahelisi seoseid.

CRUD-toimingud: SQLAlchemy ORM pakub intuitiivset liidest andmebaasisõnumite (CRUD) loomiseks, lugemiseks, uuendamiseks ja kustutamiseks Pythoni objektide abil. See genereerib automaatselt SQL-päringuid, mis põhinevad teie objektide manipuleerimisel.

Seosed: SQLAlchemy ORM toetab tabelite vaheliste seoste määratlemist, kasutades teie Pythoni klasside atribuute. See pakub võimalusi üks-ühele, üks-paljule ja palju-paljule seoste loomisele.

2.2.2 Andmebaasi ühendus

Andmebaasi ühendust kirjeldatakse failis database.py. (Joonis 2.2)

```
DATABASE_URL = f"postgresql://{USER}:{PASSWORD}@{SERVER}:{PORT}/{DB}"

SQLALCHEMY_DATABASE_URL = DATABASE_URL

Engine = create_engine(SQLALCHEMY_DATABASE_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()
```

Joonis 2.2 Andmebaasi ühendus

SQLALCHEMY_DATABASE_URL - see on ühendusrida, mis sisaldab serveri andmeid andmebaasiga ühenduse loomiseks.

Engine - see on sqlalchemy.engine.Engine klassi eksemplar, mis pakub ühendust andmebaasiga.

SessionLocal - see on seansi (session) loomiseks kasutatav ühendus andmebaasiga.

Base - see on ORM mudelite loomiseks kasutatav baasklass.

2.2.3 Mudelite loomine

SQLAlchemy mudelid on Pythoni klassid, mis esindavad andmebaasis olevaid tabeleid. Need võimaldavad töötada andmetega andmebaasis nagu tavaliste Pythoni objektidega. Andmebaas koosneb kolmest tabelist, mille jaoks on loodud kolm mudelit (klassi): Student (Joonis 2.3), Attendance (Joonis 2.4), Schedule (Joonis 2.5).

```
class Student(Base):
    __tablename__ = "students"
    id = Column(Integer,primary_key=True,index=True)
    name = Column(String, nullable=False)
    surname = Column(String, nullable=False)
    student_code = Column(String, nullable=False)
    group_name = Column(String, nullable=False)
    model = Column(JSON, nullable=False)

    attendance = relationship("Attendance", cascade = "all,delete", lazy='joined',
                              back_populates="student")
```

Joonis 2.3 Mudel Student students tabeli jaoks

```python
class Attendance(Base):
    __tablename__ = "attendance"
    id = Column(Integer,primary_key=True,index=True)
    fk_student_id = Column(Integer, ForeignKey("students.id"))
    date = Column(DateTime, nullable=False)
    attendance = Column(Boolean)
    fk_subject_id = Column(Integer, ForeignKey("schedule.id"))

    student = relationship("Student", lazy='joined', back_populates="attendance")
    schedule = relationship("Schedule", lazy='joined', back_populates="att")
```

Joonis 2.4 Mudel Attendance attendance tabeli jaoks

```python
class Schedule(Base):
    __tablename__ = "schedule"
    id = Column(Integer,primary_key=True,index=True)
    subject = Column(String, nullable=False)
    date = Column(Date, nullable=False)
    start_time = Column(Time, nullable=False)
    end_time = Column(Time, nullable=False)
    room = Column(String, nullable=False)
    group_name = Column(String, nullable=False)

    att = relationship("Attendance", lazy='joined', back_populates="schedule")
```

Joonis 2.5 Mudel Schedule schedule tabeli jaoks

Iga mudeli klass algab atribuudiga __tablename__, mis teatab SQLAlchemy'le, millise andmebaasi tabelile see vastab. Seejärel tulevad mudeli atribuudid igale veerule tabelis. Iga mudeli lõpus on kasutatud suhteid (relationship) loova funktsiooni, et luua seosed tabelite vahel.

2.2.4 Valideerimisskeemide loomine

Valideerimisskeemid on Pydantic abil määratletud klassid. Need tagavad tüübiohutuse ja andmete valideerimise. Igal mudelil on oma skeem, mis kontrollib andmete (tüübi) õigsust ja kui need ei ühti, teeb rakenduses erandi.

```python
class StudentBase(BaseModel):
    name: str
    surname: str
    student_code: str
    group_name: str

    class Config:
        orm_mode = True

class StudentDB(StudentBase):
    model: List[float]

    class Config:
        orm_mode = True

class StudentShow(StudentBase):
    id: int
```

Joonis 2.6 Valideerimisskeem students tabeli jaoks.

Tudengite tabeli jaoks on loodud kolm valideerimisskeemi (klassi):

• StudentBase – baasiline valideerimisskeem, mis pärineb BaseModel klassi omadustest. See skeem on aluseks valideerimisskeemidele, mis loovad kirjeid students tabelisse ja loevad sealt andmeid.

• StudentDB - valideerimisskeem, mis loob kirjeid students tabelisse.

• StudentShow - valideerimisskeem, mis loeb students tabelist andmeid. (Joonis 2.6)

```python
class AttendanceBase(BaseModel):
    date: datetime
    attendance: bool

    class Config:
        orm_mode = True

class AttendanceDB(AttendanceBase):
    student_code: str

class AttendanceShow(AttendanceBase):
    student: StudentShow

    class Config:
        orm_mode = True
```

Joonis 2.7 Valideerimisskeem attendance tabeli jaoks

Samamoodi on loodud valideerimisskeemid attendance tabeli (Joonis 2.7) ja schedule tabeli (Joonis 2.8) jaoks.

```python
class ScheduleBase(BaseModel):
    subject: str
    date: date
    start_time: time
    end_time: time
    group_name: str

    class Config:
        orm_mode = True

class ScheduleDB(ScheduleBase):
    room: str

class ScheduleShow(ScheduleBase):
    id: int
    room: str

    class Config:
        orm_mode = True
```

Joonis 2.8 Valideerimisskeem schedule tabeli jaoks

2.2.5 Andmete lisamine, lugemine, muutmine, kustutamine

Andmete lisamise, lugemise, muutmise ja andmebaasist kustutamise toimingute funktsioone kirjeldatakse failis crud.py. Täielik kood lisas LISA 1. Järgnevalt on toodud näited tunniplaani lisamisest (Joonis 2.9), tudengi andmete lugemisest andmebaasist, kasutades tudengi koodi (Joonis 2.10) ja kustutamisest. (Joonis 2.11)

```python
def add_schedule(schedule: models.Schedule, db: Session):
    new_schedule = models.Schedule(**schedule.dict())
    db.add(new_schedule)
    db.commit()
    db.refresh(new_schedule)
    return new_schedule
```

Joonis 2.9 Tunniplaani lisamise funktsioon

```python
def get_student_by_student_code(db: Session, student_code: str):
    return db.query(models.Student).filter(models.Student.student_code == student_code).first()
```

Joonis 2.10 Tudengi andmete lugemise funktsioon

```python
def delete_student_by_student_code(db: Session, student_code: str):
    student = db.query(models.Student).filter(models.Student.student_code == student_code).first()
    if not student:
        return {'Student not found'}
    db.delete(student)
    db.commit()
    return {"Student deleted": True}
```

Joonis 2.11 Tudengi andmete kustutamise funktsioon

2.2.6 API

Failis main.py asuvad meetodid HTTP-päringute töötlemiseks. Alguses tuleb luua app objekt. App objekt luuakse FastAPI eksemplarina. See objekt esindab FastAPI rakendust.

```
app = FastAPI()
```

Origins loetelu määrab CORS-ile lubatud päritolu loetelu. CORS on mehhanism, mis võimaldab veebilehe piiratud ressurssidel pärida teisest domeenist, erinevast domeenist, kust ressurss pärineb. Loetelu sisaldab URL-e, kus on sellele API-le lubatud päringuid teha. (Joonis 2.12)

```
origins = [
    "http://localhost:3000",
    "http://localhost",
    "http://localhost:8080",
    ]
```

Joonis 2.12 Loetelu origins

Funktsiooni app.add_middleware kasutatakse CORSMiddleware vahevara lisamiseks rakendusele. See vahevara käsitleb CORS-iga seotud päiseid ja lahendab domeenide vahelised päringud määratud allikast. Parameeter allow_origins seatakse lähtekohtade loendisse, mis lubab päringuid määratud lähtekohtadest. Suvandid allow_credentials, allow_methods, allow_headers ja expose_headers on seatud lubama taotlustes erinevaid HTTP-meetodeid, päiseid ja mandaate. (Joonis 2.13)

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
    expose_headers=["*"],
)
```

Joonis 2.13 Funktsioon app.add_middleware

Funktsioon get_db on määratletud sõltuvusena. Sõltuvused on funktsioonid, mida käivitatakse marsruudile või muule sõltuvusele kindla väärtuse andmiseks. Sel juhul avab get_db andmebaasi (db) seansi teistele sellest sõltuvatele funktsioonidele. See loob seansi rakendusega SessionLocal() ja edastab selle tootlusavaldusega. Kui sõltuv funktsioon on lõpetatud, käivitatakse viimane plokk, milles andmebaasi seanss suletakse käsuga db.close(). (Joonis 2.14)

```python
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Joonis 2.14 Funktsioon get_db

Esitatud on näiteks HTTP-päringute töötlemise funktsioonid, sealhulgas tudengite andmete päringu töötlemine (Joonis 2.15), ajakava lisamine (Joonis 2.16) ja selle kustutamine (Joonis 2.17). Kõik päringute töötlemise funktsioonid on täielikult esitatud LISA 2 rakenduses.

```python
@app.get("/students", response_model=List[student.StudentShow])
async def read_students(db: Session = Depends(get_db)):
    students = crud.get_students(db)
    return students
```

Joonis 2.15 Funktsioon tudengite andmete lugemise päringu töötlemiseks

```python
@app.post("/add/schedule")
async def add_schedule(schedule: schedule.ScheduleDB, db: Session = Depends(get_db)):
    new_schedule = crud.add_schedule(schedule=schedule, db=db)
    return new_schedule
```

Joonis 2.16 Funktsioon uue tunniplaani lisamise päringu töötlemiseks

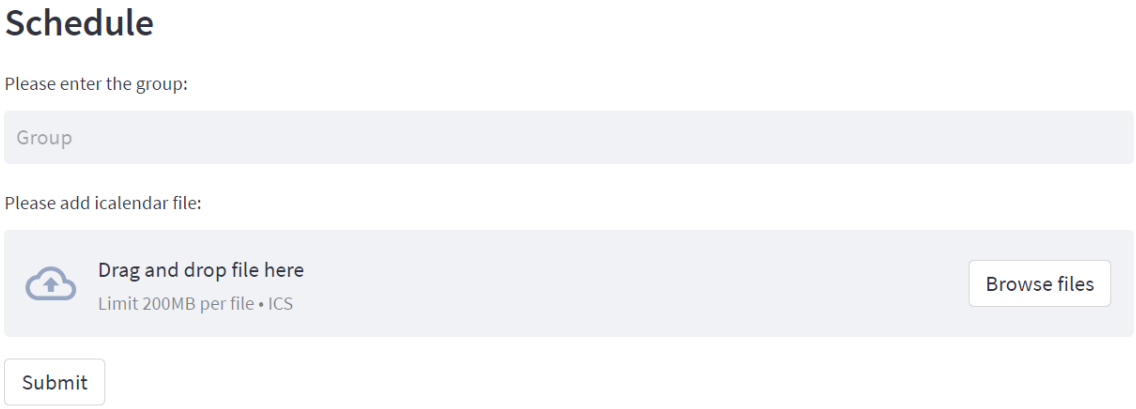
```python
@app.delete("/delete/schedule")
async def delete_schedule(db: Session = Depends(get_db)):
    return crud.delete_schedule(db=db)
```

Joonis 2.17 Funktsioon tunniplaani kustutamise päringu töötlemiseks.

2.3 Kasutajaliides

2.3.1 Tunniplaani leht

Tunniplaani lehel saab ülesse laadida tunniplaani, mis salvestatakse andmebaasi ja seda kasutatakse tudengite kohaloleku määramiseks. Kõigepealt tuleb sisestada grupp, millele tunniplaani laaditakse, seejärel tuleb laadida tunniplaani fail ics-vormingus. (Joonis 2.18)



Joonis 2.18 Tunniplaani laadimise leht.

Tunniplaani faili (ICS) töötlemiseks on vaja installida täiendavad teegid: icalendar ja pytz.

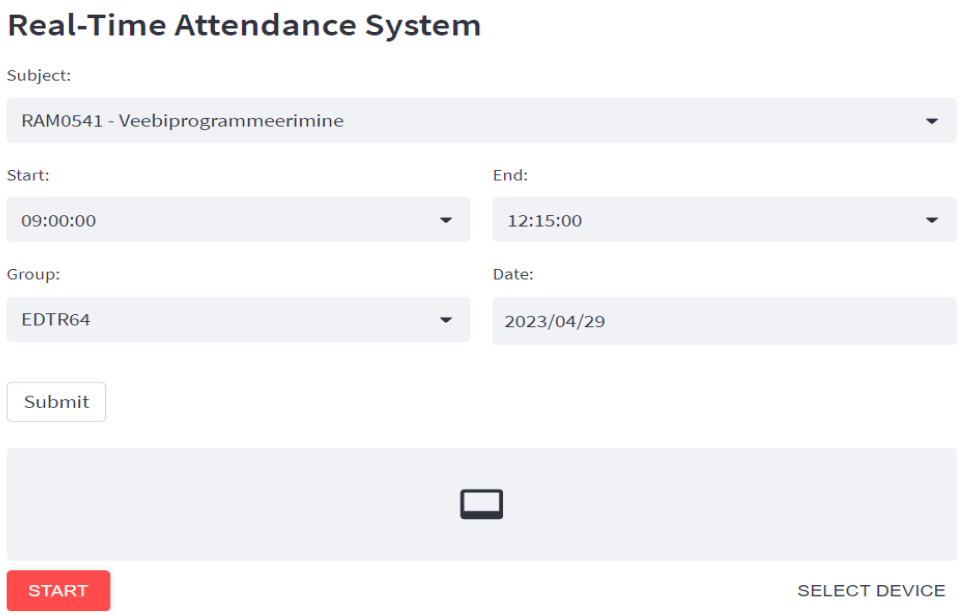
```
st.subheader('Schedule ')

group = st.text_input(label='Please enter the group:',placeholder='Group', key='group_input')
schedule = st.file_uploader("Please add icalendar file:", type='ics')
if st.button('Submit', key='submit'):
    if not group:
        st.error('Group is required')
    if schedule is None:
        st.error('Schedule file is required')
    else:
        if schedule is not None:
            cal = icalendar.Calendar.from_ical(schedule.read())
            group = group.replace(" ", "").upper()
            dataList = []
            tz = pytz.timezone('Europe/Tallinn')
            for event in cal.walk('VEVENT'):
                subject = event['SUMMARY'].to_ical().decode('utf-8')
                start_time = event['DTSTART'].dt
                start_time = start_time.replace(tzinfo=pytz.utc)
                start_time = start_time.astimezone(tz)
                date = start_time.strftime('%Y-%m-%d')
                start_time = start_time.strftime('%H:%M:%S')
                end_time= event['DTEND'].dt
                end_time = end_time.replace(tzinfo=pytz.utc)
                end_time = end_time.astimezone(tz)
                end_time = end_time.strftime('%H:%M:%S')
                room = event['LOCATION'].to_ical().decode('utf-8')
                data = {"subject": subject, "date": date,
                        "start_time": start_time, "end_time": end_time,
                        "room": room, "group_name": group}
                dataList.append(data)
            with st.spinner("Adding data to database..."):
                for data in dataList:
                    res = requests.post(url='http://127.0.0.1:8000/add/schedule',
                                        data=json.dumps(data))
                st.success("Data sucessfully added to database")
```

Joonis 2.19 Kood ics faili töötlemiseks ja andmete serverisse saatmiseks

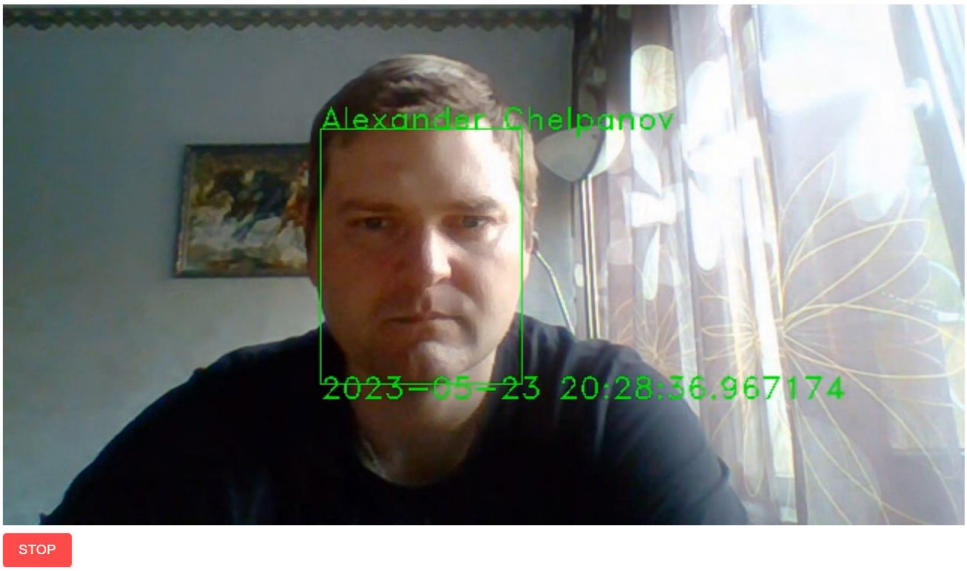
2.3.2 Kohaloleku registreerimise leht

Real time prediction lehel toimub reaalajas isiku tuvastamine veebikaamera abil. Selleks peab valima aine, tunni alguse ja lõpu aja, rühma ning kuupäeva. Seejärel vajutada "Submit" ja andmebaasist laaditakse alla tudengite tuvastamise mudelid antud rühmast. (Joonis 2.20)

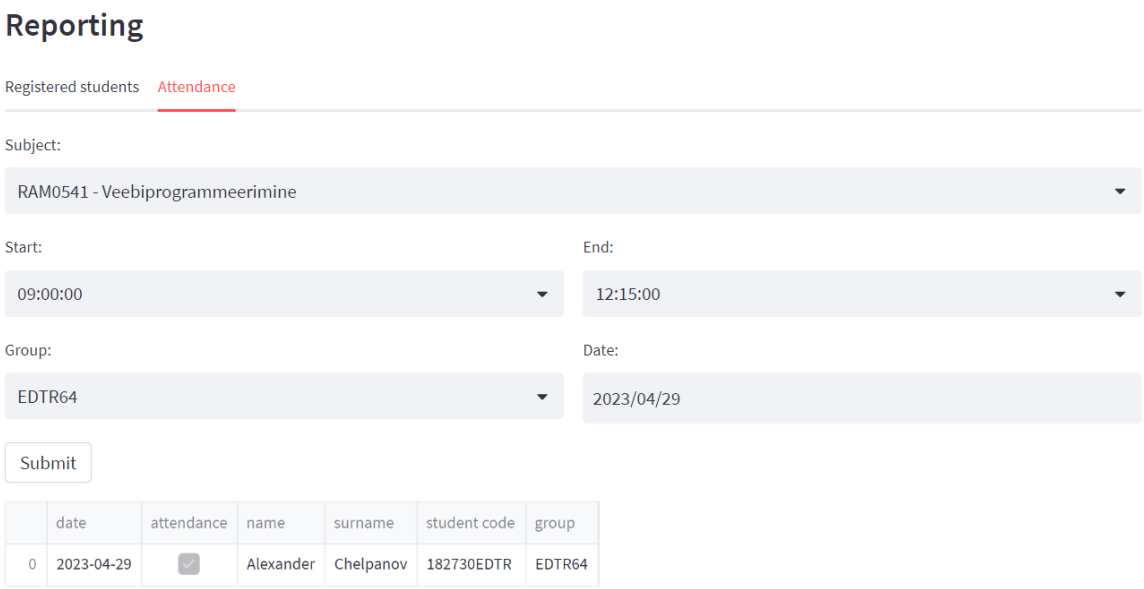
Nupuga "START" käivitatakse isiku tuvastamise protsess. "SELECT DEVICE" abil saab valida kaamera.



Joonis 2.20 Kaamera abil toimuva automaatse tudengite arvestuse leht



Joonis 2.21 Näotuvastuse protsess



Joonis 2.22 Repost lehel andmebaasis kohaloleku salvestamise tulemus

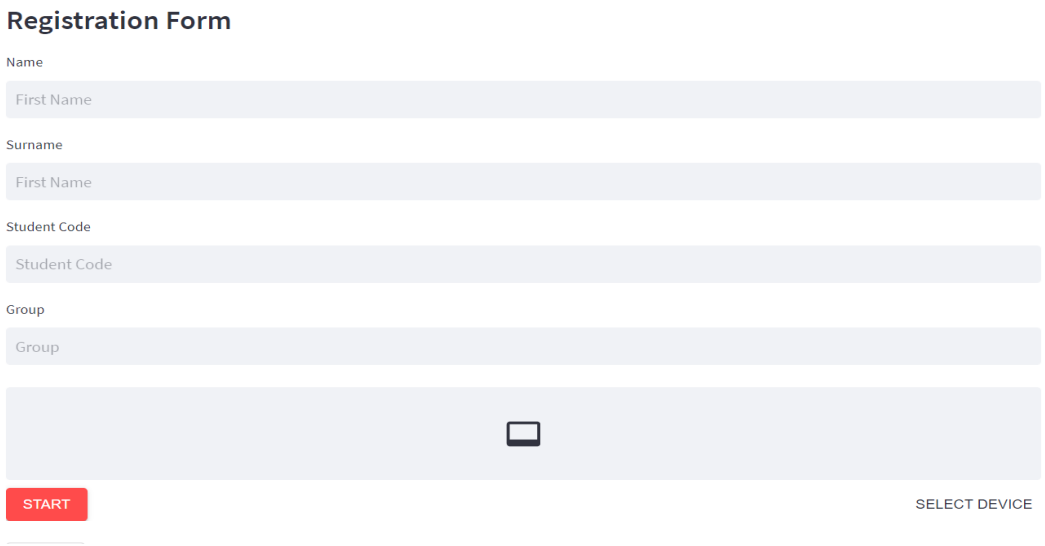
```
if st.button('Submit', key='submit'):
    if 'submitted' not in st.session_state:
        st.session_state['submitted'] = True

if 'submitted' in st.session_state:
    if st.session_state.submitted:
        with st.spinner('Retriving Data from database ...'):
            face_db = face_rec.retrive_data(group=group)
            subject_id = face_rec.get_subject_id(subject=subject,
                                                 group_name=group, start_time=start_time,
                                                 end_time=end_time, date=date)
            if subject_id is False:
                st.error('This schedule not found', icon="🚨")
            if subject_id:
                att_dict = getCodeList(face_db, subject_id)
            st.header('Group {group} models:')
            st.dataframe(face_db)
            st.success("Data sucessfully retrived from database")

realtimepred = face_rec.RealTimePred()

def video_frame_callback(frame):
    img = frame.to_ndarray(format="bgr24")
    pred_img = realtimepred.face_prediction(img,face_db,
                                            'model',['name','surname','student_code'],thresh=0.5)
    if subject_id:
        att_dict = realtimepred.saveLogs(subject_id=subject_id, date=date, att_dict=att_dict)
    return av.VideoFrame.from_ndarray(pred_img, format="bgr24")

webrtc_streamer(key="realtimePrediction", video_frame_callback=video_frame_callback)
```

Joonis 2.23 Näotuvastuse lehe kood

Lisakood on failis face_rec.py (Joonis 3).

2.3.3 Registreerimisvormi leht

Registreerimisvormi leht on mõeldud tudengite andmete kogumiseks ja isikutuvastuse mudeli loomiseks. Tuleb sisestada järgmised andmed: eesnimi, perekonnanimi, tudengikood ja grupp. Kui vajutate nuppu "START", käivitatakse näidiste kogumine näotuvastuse mudeli loomiseks.



Joonis 2.24 Tudengite registreerimisleht

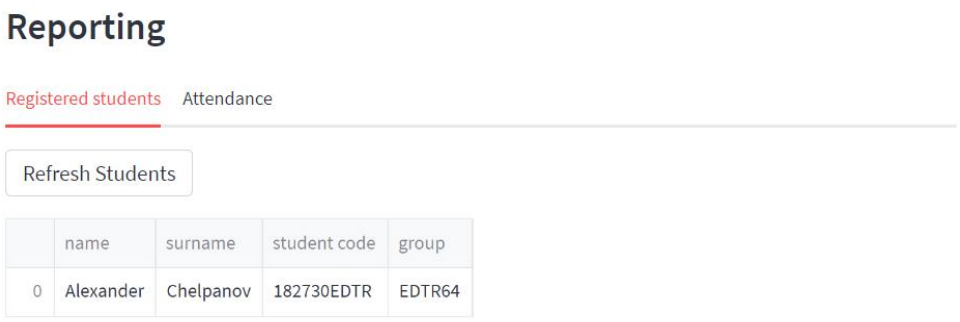
```
name = st.text_input(label='Name',placeholder='First Name')
surname = st.text_input(label='Surname',placeholder='First Name')
student_code = st.text_input(label='Student Code',placeholder='Student Code')
group_name = st.text_input(label='Group',placeholder='Group')

# step-2: Collect facial embedding of that person
def video_callback_func(frame):
    img = frame.to_ndarray(format='bgr24') # 3d array bgr
    reg_img, embedding = registration_form.get_embedding(img)
    # two step process
    # 1st step save data into local computer txt
    if embedding is not None:
        with open('face_embedding.txt',mode='ab') as f:
            np.savetxt(f,embedding)

    return av.VideoFrame.from_ndarray(reg_img,format='bgr24')

webrtc_streamer(key='registration',video_frame_callback=video_callback_func)

# step-3: save the data in database
if st.button('Submit'):
    return_value = registration_form.save_data_in_db(name=name, surname=surname,
                                                     student_code=student_code, group_name=group_name)
    if return_value == True:
        st.success(f"{name} {surname} registered sucessfully")
    elif return_value == 'name_false':
        st.error('Please enter the name: name cannot be empty or spaces')
    elif return_value == 'surname_false':
        st.error('Please enter the surname: surname cannot be empty or spaces')
    elif return_value == 'student_code_false':
        st.error('Please enter yout student code: student code cannot be empty or spaces')
    elif return_value == 'group_name_false':
        st.error('Please enter the group: group cannot be empty or spaces')
    elif return_value == 'file_false':
        st.error('face_embedding.txt is not found. Please refresh the page and execute again.')
```

Joonis 2.9 Tudengite registreerimislehe kood

Lisakood on failis face_reg.py (LISA 3)

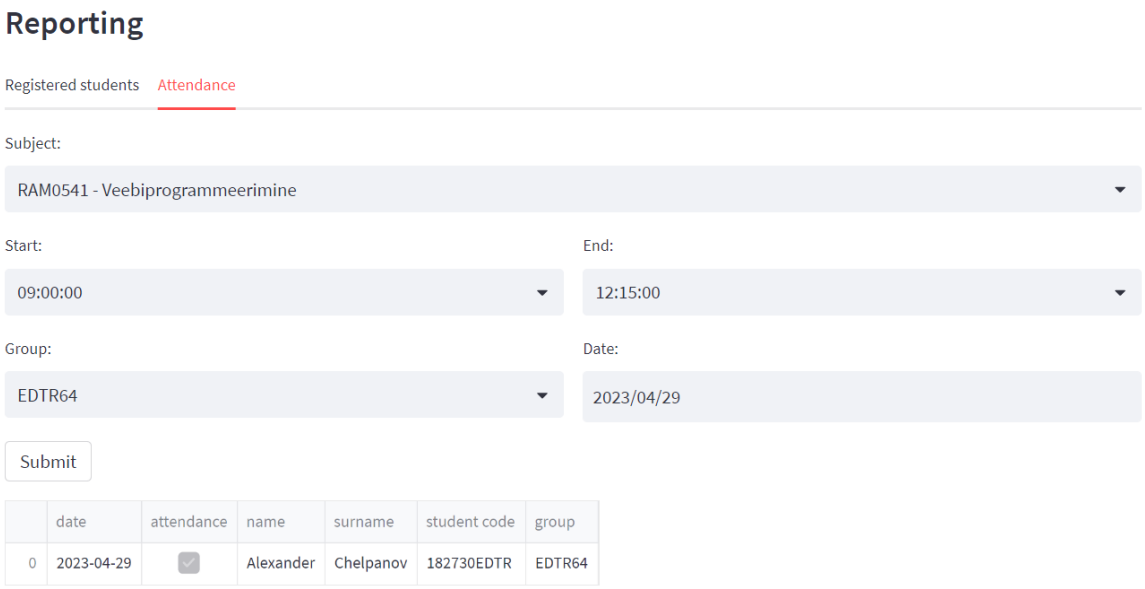
2.3.4 Aruannete leht

Aruandluse lehel saab vaadata juba registreerunud tudengid, andmebaasis registreeritud tudengite vahekaardil (Joonis 2.24).



Joonis 2.10 Registreeritud tudengite vahekaart

Kohaloleku vahekaardil kuvatakse otsimise teel tudengite kohaloleku andmed. (Joonis 2.25)



Joonis 2.11 Kohaloleku andmete vahekaart koos otsinguga

Lehe kood on LISA 4-s.

3 RAKENDUSE EDASIARENDAMINE

Üliõpilaste kohaloleku automaatse arvestuse rakendus, mis kasutab arvutinägemist, omab potentsiaali edasiseks arenguks. Siin on mõned võimalikud arenguperspektiivid:

- Täpsuse parandamine: Kaasaegsed arvutinägemise tehnoloogiad arenevad pidevalt, mis võimaldab parandada näotuvastuse täpsust ja üliõpilaste identifitseerimist. Edusammud näotuvastuse algoritmides ja süvaõppes vähendavad vigade tõenäosust ja parandavad süsteemi usaldusväärsust.

- Funktsionaalsuse laiendamine: Kohaloleku arvestamise rakendust saab täiendada, et pakkuda eri lisafunktsioone. Näiteks võib lisada automaatse kohalolekuaruannete genereerimise ja integreerimise õppeprotsessi juhtimissüsteemidega.

- Platvormi laiendamine: Rakendust saab kohandada erinevatele platvormidele, sealhulgas mobiilseadmetele. See võimaldab üliõpilastel ja õpetajatel kontrollida kohalolekuinfot reaalajas ning saada teatisi igal ajal ja igal pool.

- Andmeanalüüs: Kogutud kohalolekuga seotud andmeid saab kasutada analüüsimiseks ja trendide tuvastamiseks. Näiteks võib läbi viia uuringu kohaloleku ja üliõpilaste akadeemilise edukuse seose kohta või tuvastada loengute vahelejätmise põhjused. See võib aidata parimate lahenduste leidmisel haridusprotsessi parendamiseks.

KOKKUVÕTTE

Lõputöö teema: Tudengite automaatne kohaloleku arvestus arvutinägemise abil.

Käesolev lõputöö on jaotatud kolme peatükki. Esimeses peatükis uuritakse olemasolevaid meetodeid ja erinevaid raamatukogusid inimeste näotuvastuseks: OpenCV, Dlib, DeepFace, InsightFace. Samuti uuritakse tehnoloogiaid, mis on valitud veebirakenduse loomiseks: SQLAlchemy, FastAPI, Streamlit. Teises peatükis käsitletakse rakenduse arendamist: andmebaasi loomine, serveripoolne osa andmebaasiga suhtlemiseks, kasutajaliides. Serveripoolne osa teostab kõik toimingud andmebaasiga suhtlemiseks. Kasutajaliides võimaldab registreerida üliõpilasi tundides, laadida üles kohaloleku märkimiseks graafikuid ja kuvada kohalolekuaruandeid. Kolmandas peatükis antakse ülevaade edasistest rakenduse arendamise suundadest.

Autori eesmärgiks oli luua veebirakendus, mis võimaldab automaatselt arvestada üliõpilaste kohalolekut ja salvestada see andmebaasi. Autor valis tehnoloogiavaramuks Pythoni programmeerimiskeele, FastAPI raamistiku, PostgreSQL andmebaasisüsteemi ja Streamlit teeki.

Tulemuseks oli veebirakendus, mis võimaldab automaatselt jälgida tudengite kohalolekut ja salvestada kogutud info andmebaasi.

SUMMARY

Theme of graduation work Automatic registration of students' attendance with the help of computer vision.

This thesis is divided into three chapters. The first chapter explores existing methods and various libraries for human face recognition: OpenCV, Dlib, DeepFace, InsightFace. The technologies that were chosen to create a web application were also considered: SQLAlchemy, FastAPI, Streamlit. The second chapter deals with the development of the application: the creation of a database, the server part for interacting with the database, the user interface. The server part performs all operations of interaction with the database. The user interface allows you to register students for classes, download a schedule for attendance records, and display attendance reports. The third chapter provides an overview of further directions for the development of the application.

The author's goal was to create a web application for automatically recording student attendance and saving it to a database. The author chose a technology stack: Python programming language, FastAPI framework, PostgreSQL DBMS, Streamlit library.

As a result, a web application was created that allows you to automatically record students with saving to a database.

KASUTATUD ALLIKAD

1. OpenCV, „About OpenCV", OpenCV team, 2023. [Online] Available: https://opencv.org/about/ (15.03.2023)
2. Dlib, „Dlib Overview", [Online] Available: http://dlib.net/intro.html (15.03.2023)
3. Analyticsindiamag, „ Face recognition system using deepface", [Online] Available: https://analyticsindiamag.com/face-recognition-system-using-deepfacewith-python-codes/ (17.03.2023)
4. Insight Face Github Repository. Available:https://github.com/deepinsight/insightface (03.2023)
5. PostgreSQL, „About PostgreSQL", The PostgreSQL Global Development Group, 1996 - 2023. [Online]. Available: https://www.postgresql.org/about/ [02. 04. 2023].
6. MIT, „FastAPI Features", 2023. [Online]. Available: https://fastapi.tiangolo.com/features/ [10. 04. 2023].
7. Yancy Dennis, „ Flask vs Django vs Streamlit ", MIT, 2023. [Online]. Available: https://python.plainenglish.io/flask-vs-django-vs-streamlit [17. 04. 2023].
8. SQLAlchemy, „SQLAlchemy Overview", SQLAlchemy authors and contributors, 2007 - 2023. [Online]. Available: https://docs.sqlalchemy.org/en/20/intro.html [05. 04. 2023].
9. FastAPI Documentation. Available: https://fastapi.tiangolo.com/tutorial/sql-databases/ (02.2023)
10. Streamlit Documentation. Available: https://docs.streamlit.io/ (03.2023)
11. SQLAlchemy Documentation. Available: https://docs.sqlalchemy.org/en/20/ (02.2023)
12. PostgreSQL Documentation. Available: https://www.postgresql.org/docs/current/index.html (02.2023)

LISAD

LISA 1. CRUD

```python
def get_student_by_id(db: Session, student_id: int):
    return db.query(models.Student).filter(models.Student.id == student_id).first()

def get_student_by_student_code(db: Session, student_code: str):
    return db.query(models.Student).filter(models.Student.student_code == student_code).first()

def get_student_by_name_and_surname(db: Session, name: str, surname: str):
    return db.query(models.Student).filter(models.Student.name == name,
                                           models.Student.surname == surname).first()

def get_students_by_group(db: Session, group_name: str):
    group_name = group_name.upper()
    group = db.query(models.Student).filter(models.Student.group_name == group_name).all()
    return group

def get_students(db: Session):
    return db.query(models.Student).all()

def get_schedule(db: Session):
    return db.query(models.Schedule).all()

def delete_student_by_id(db: Session, id: int):
    student = db.query(models.Student).filter(models.Student.id == id).first()
    if not student:
        return {'Student not found'}
    db.delete(student)
    db.commit()
    return {"Student deleted": True}

def delete_student_by_student_code(db: Session, student_code: str):
    student = db.query(models.Student).filter(models.Student.student_code == student_code).first()
    if not student:
        return {'Student not found'}
    db.delete(student)
    db.commit()
    return {"Student deleted": True}

def delete_group(db: Session, group_name: str):
    group_students = db.query(models.Student).filter(models.Student.group_name == group_name).all()
    if not group_students:
        return {'Group not found'}
    for student in group_students:
        db.delete(student)
    db.commit()
    return {"Group deleted": True}
```

```python
def add_student(db:Session, student: student.StudentDB):
    new_student = models.Student(**student.dict())
    new_student.group_name = new_student.group_name.replace(" ", "").upper()
    new_student.name = new_student.name.strip().lower().capitalize()
    new_student.surname = new_student.surname.strip().lower().capitalize()
    new_student.student_code = new_student.student_code.replace(" ", "").upper()
    db.add(new_student)
    db.commit()
    db.refresh(new_student)
    return  new_student

def get_attendance(db: Session):
    return db.query(models.Attendance).join(models.Student).all()

def get_attendance_by_group(group_name: str, db: Session):
    return db.query(models.Attendance).join(models.Student).filter(models.Student.group_name
                                                                   == group_name).all()

def get_attendance_by_name_and_surname(name: str, surname: str, db: Session):
    return db.query(models.Attendance).join(models.Student).filter(models.Student.name
                                                                   == name,
                                                                   models.Student.surname
                                                                   == surname).all()

def get_attendance_by_name_surname_group(name: str, surname: str, group_name: str,  db: Session):
    return db.query(models.Attendance).join(models.Student).filter(models.Student.name
                                                                   == name,
                                                                   models.Student.surname
                                                                   == surname,
                                                                   models.Student.group_name
                                                                   == group_name).all()

def get_attendance(db: Session):
    return db.query(models.Attendance).join(models.Student).all()

def add_attendance(attendance, db: Session):
    student_code = attendance.student_code
    student = get_student_by_student_code(student_code=student_code, db=db)
    if not student:
        return {"Student found": False}
    subject = attendance.subject
    date = attendance.date
    attendance = attendance.attendance
    new_attendance = models.Attendance(fk_student_id=student.id, subject=subject,
                                       date=date, attendance=attendance)
    db.add(new_attendance)
    db.commit()
    db.refresh(new_attendance)
    return new_attendance
```

```python
def add_schedule(schedule: models.Schedule, db: Session):
    new_schedule = models.Schedule(**schedule.dict())
    db.add(new_schedule)
    db.commit()
    db.refresh(new_schedule)
    return new_schedule

def delete_schedule_by_id(db: Session, id: int):
    schedule = db.query(models.Schedule).filter(models.Schedule.id == id).first()
    if not schedule:
        return {'Schedule not found'}
    db.delete(schedule)
    db.commit()
    return {"Schedule deleted": True}

def delete_schedule(db: Session):
    scheduleList = db.query(models.Schedule).all()
    if not scheduleList:
        return {'Schedule not found'}
    for schedule in scheduleList:
        db.delete(schedule)
    db.commit()
    return {"Schedule deleted": True}
```


```python
def delete_schedule(db: Session):
    scheduleList = db.query(models.Schedule).all()
    if not scheduleList:
        return {'Schedule not found'}
    for schedule in scheduleList:
        db.delete(schedule)
    db.commit()
    return {"Schedule deleted": True}
```


```python
def get_subject_id(subject: str, start_time: time, end_time: time,
                   group_name: str, date: date, db: Session):
    subject = db.query(models.Schedule).filter(models.Schedule.date == date,
                                               models.Schedule.subject == subject,
                                               models.Schedule.start_time == start_time,
                                               models.Schedule.end_time == end_time,
                                               models.Schedule.group_name == group_name
                                                ).first()
    if not subject:
        return {'subject not found'}
    return subject.id
```

LISA 2. API

```python
@app.get("/students", response_model=List[student.StudentShow])
async def read_students(db: Session = Depends(get_db)):
    students = crud.get_students(db)
    return students

@app.get("/attendance", response_model=List[attendance.AttendanceShow])
async def read_attendencies(db: Session = Depends(get_db)):
    attendance = crud.get_attendance(db)
    return attendance

@app.get("/schedule", response_model=List[schedule.ScheduleShow])
async def read_attendencies(db: Session = Depends(get_db)):
    schedule = crud.get_schedule(db)
    return schedule

@app.get("/schedule/{subject}/{group_name}/{start_time}/{end_time}/{date}")
async def get_subject_id(subject: str, start_time: time, end_time: time,
                         group_name: str, date: date, db: Session = Depends(get_db)):
    subject_id = crud.get_subject_id(subject=subject, start_time=start_time,
                                     end_time=end_time, group_name=group_name,
                                     date=date,  db=db)
    return subject_id

@app.get("/attendance/{group_name}", response_model=List[attendance.AttendanceShow])
async def read_attendencies_by_group(group_name: str, db: Session = Depends(get_db)):
    attendance = crud.get_attendance_by_group(group_name=group_name, db=db)
    return attendance

@app.get("/students/{id}", response_model=student.StudentShow)
async def read_student_by_id(id: int, db: Session = Depends(get_db)):
    student = crud.get_student_by_id(student_id=id, db=db)
    return student

@app.get("/students/groups/{group_name}", response_model=List[student.StudentShow])
async def read_students_by_group(group_name: str, db: Session = Depends(get_db)):
    group = crud.get_students_by_group(group_name=group_name, db=db)
    return group

@app.get("/students/groups/{group_name}/models", response_model=List[student.StudentDB])
async def read_students_models_by_group(group_name: str, db: Session = Depends(get_db)):
    group = crud.get_students_by_group(group_name=group_name, db=db)
    return group

@app.post("/add/student")
async def add_student(student: student.StudentDB, db: Session = Depends(get_db)):
    new_student = crud.add_student(student=student, db=db)
    return new_student

@app.post("/add/attendance")
async def add_attendance(attendance: attendance.AttendanceDB, db: Session = Depends(get_db)):
    new_attendance = crud.add_attendance(attendance=attendance, db=db)
    return new_attendance
```

```python
@app.post("/add/schedule")
async def add_schedule(schedule: schedule.ScheduleDB, db: Session = Depends(get_db)):
    new_schedule = crud.add_schedule(schedule=schedule, db=db)
    return new_schedule

@app.delete("/schedule/id/{id}")
async def delete_schedule_by_id(id: int, db: Session = Depends(get_db)):
    return crud.delete_schedule_by_id(id=id, db=db)

@app.delete("/delete/schedule")
async def delete_schedule(db: Session = Depends(get_db)):
    return crud.delete_schedule(db=db)

@app.delete("/students/id/{id}")
async def delete_student(id: int, db: Session = Depends(get_db)):
    return crud.delete_schedule_by_id(id=id, db=db)

@app.delete("/students/student_code/{student_code}")
async def delete_student(student_code: str, db: Session = Depends(get_db)):
    return crud.delete_student_by_student_code(student_code=student_code, db=db)

@app.delete("/students/groups/{group_name}")
async def delete_group(group_name: str, db: Session = Depends(get_db)):
    return crud.delete_group(group_name=group_name, db=db)
```

LISA 3 FACE_REC

```python
# Retrive Data from database
def retrive_data(group: str):
    req = requests.get(f"http://127.0.0.1:8000/students/groups/{group}/models")
    j = req.json()
    models = pd.DataFrame.from_dict(j)
    return models

def retrive_students():
    req = requests.get(f"http://127.0.0.1:8000/students")
    j = req.json()
    students = pd.DataFrame.from_dict(j)
    return students

def get_subject_id(subject: str, group_name: str, start_time: str, end_time: str, date: date):
    req = requests.get(f"http://127.0.0.1:8000/schedule/{subject}/{group_name}/{start_time}/{end_time}/{date}")
    id = req.json()
    return id

def get_attendance_dict(code_list, subject_id):
    att_dict = {}
    for code in code_list:
        req = requests.get(url='http://127.0.0.1:8000/attendace_check/{subject_id}/{code}')
        att = req.json()
        att_dict[code] = att
    print(att_dict)
    return att_dict
```

```python
def getScheduleLists():
    req = requests.get(url='http://127.0.0.1:8000/schedule')
    j = req.json()
    schedule = pd.DataFrame.from_dict(j)
    if schedule.empty is False:
        subject_list = set(schedule['subject'].values.tolist())
        start_list = set(schedule['start_time'].values.tolist())
        end_list = set(schedule['end_time'].values.tolist())
        room_list = set(schedule['room'].values.tolist())
        group_list = set(schedule['group_name'].values.tolist())
        date_list = set(schedule['date'].values.tolist())
        return subject_list, start_list, end_list, room_list, group_list, date_list
    return [], [], [], [], [], []

# configure face analysis
faceapp = FaceAnalysis(name='buffalo_sc',root='insightface_model', providers = ['CPUExecutionProvider'])
faceapp.prepare(ctx_id = 0, det_size=(640,640), det_thresh = 0.5)

# ML Search Algorithm
def ml_search_algorithm(dataframe,feature_column,test_vector,
                        name_surname_code=['name','surname','student_code'],thresh=0.5):
    """
    cosine similarity base search algorithm
    """
    # step-1: take the dataframe (collection of data)
    dataframe = dataframe.copy()
    # step-2: Index face embeding from the dataframe and convert into array
    X_list = dataframe[feature_column].tolist()
    x = np.asarray(X_list)

    # step-3: Cal. cosine similarity
    similar = pairwise.cosine_similarity(x,test_vector.reshape(1,-1))
    similar_arr = np.array(similar).flatten()
    dataframe['cosine'] = similar_arr

    # step-4: filter the data
    data_filter = dataframe.query(f'cosine >= {thresh}')
    if len(data_filter) > 0:
        # step-5: get the person name
        data_filter.reset_index(drop=True,inplace=True)
        argmax = data_filter['cosine'].argmax()
        student_name, student_surname, student_code = data_filter.loc[argmax][name_surname_code]

    else:
        student_name = 'Unknown'
        student_surname = 'Unknown'
        student_code = 'Unknown'
    return student_name, student_surname, student_code
```

```python
### Real Time Prediction
class RealTimePred:
    def __init__(self):
        self.logs = dict(name=[],surname=[],student_code=[])

    def reset_dict(self):
        self.logs = dict(name=[],surname=[],student_code=[])

    def saveLogs(self, subject_id: str, date: date, att_dict: dict):
        # step-1: create a logs dataframe
        dataframe = pd.DataFrame(self.logs)
        # step-2: drop the duplicate information (distinct student code)
        # dataframe.drop_duplicates('student_code',inplace=True)
        # step-3: push data to  database (list)
        # encode the data
        name_list = dataframe['name'].tolist()
        surname_list = dataframe['surname'].tolist()
        student_code_list = dataframe['student_code'].tolist()
        data = {}
        for name, surname, code in zip(name_list, surname_list, student_code_list):
            if name != 'Unknown' and surname != 'Unknown':
                if att_dict[code] == False:
                    att = True
                    data = {"subject_id": subject_id, "date": date.strftime("%Y-%m-%d"), "attendance": att,
                            "student_code": code}
                    res = requests.post(url='http://127.0.0.1:8000/add/attendance', data=json.dumps(data))
                    att_dict[code] = True

        self.reset_dict()

    def face_prediction(self,test_image, dataframe,feature_column,
                        name_surname_code=['name','surname', 'student_code'],thresh=0.5):
        # step-1: find the time
        current_time = str(datetime.now())

        # step-1: take the test image and apply to insight face
        results = faceapp.get(test_image)
        test_copy = test_image.copy()
        # step-2: use for loop and extract each embedding and pass to ml_search_algorithm

        for res in results:
            x1, y1, x2, y2 = res['bbox'].astype(int)
            embeddings = res['embedding']
            student_name, student_surname, student_code = ml_search_algorithm(dataframe,
                                                                              feature_column,
                                                                              test_vector=embeddings,
                                                                              name_surname_code=name_surname_code,
                                                                              thresh=thresh)
            text_gen = student_name + ' ' + student_surname

            if student_name == 'Unknown':
                color =(0,0,255) # bgr'
                text_gen = 'Unknown'
            else:
                color = (0,255,0)

            cv2.rectangle(test_copy,(x1,y1),(x2,y2),color)


            cv2.putText(test_copy,text_gen,(x1,y1),cv2.FONT_HERSHEY_DUPLEX,0.7,color,1)
            cv2.putText(test_copy,current_time,(x1,y2+10),cv2.FONT_HERSHEY_DUPLEX,0.7,color,1)
            # save info in logs dict

            self.logs['name'].append(student_name)
            self.logs['surname'].append(student_surname)
            self.logs['student_code'].append(student_code)

        return test_copy
```

```python
class RegistrationForm:
    def __init__(self):
        self.sample = 0
    def reset(self):
        self.sample = 0

    def get_embedding(self,frame):
        # get results from insightface model
        results = faceapp.get(frame,max_num=1)
        embeddings = None
        for res in results:
            self.sample += 1
            x1, y1, x2, y2 = res['bbox'].astype(int)
            cv2.rectangle(frame, (x1,y1),(x2,y2),(0,255,0),1)
            # put text samples info
            text = f"samples = {self.sample}"
            cv2.putText(frame,text,(x1,y1),cv2.FONT_HERSHEY_DUPLEX,0.6,(255,255,0),2)

            # facial features
            embeddings = res['embedding']

        return frame, embeddings
```

```python
def save_data_in_db(self, name, surname, student_code, group_name):
    if not name:
        return 'name_false'
    if not surname:
        return 'surname_false'
    if not student_code:
        return 'student_code_false'
    if not group_name:
        return 'group_name_false'

    # if face_embedding.txt exists
    if 'face_embedding.txt' not in os.listdir():
        return 'file_false'

    # step-1: load "face_embedding.txt"
    x_array = np.loadtxt('face_embedding.txt',dtype=np.float32) # flatten array

    # step-2: convert into array (proper shape)
    received_samples = int(x_array.size/512)
    x_array = x_array.reshape(received_samples,512)
    x_array = np.asarray(x_array)

    # step-3: cal. mean embeddings
    x_mean = x_array.mean(axis=0)
    x_mean = x_mean.astype(np.float32)
    model = x_mean.tolist()

    # step-4: save this into database
    data = {"name": name, "surname": surname, "student_code": student_code, "group_name": group_name, "model": model}
    res = requests.post(url='http://127.0.0.1:8000/add/student', data=json.dumps(data))

    os.remove('face_embedding.txt')
    self.reset()

    return True
```

LISA 4 REPORT

```
def load_attendance():
    # extract all data from  database
    req = requests.get(f"http://127.0.0.1:8000/students")
    dict = req.json()
    list = pd.json_normalize(dict)
    return list

# tabs to show the info
tab1, tab2 = st.tabs(['Registered students', 'Attendance'])
with tab1:
    if st.button('Refresh Students'):
        # Retrive the data from Database
        with st.spinner('Retriving students from DB ...'):
            students = face_rec.retrive_students()
            students.rename(columns={'student_code':'student code', 'group_name':'group' }, inplace=True)
            st.write(students)
with tab2:
        subject_list, start_list, end_list, room_list, group_list, date_list = getScheduleLists()

        subject = st.selectbox('Subject:', subject_list)

        col1, col2 = st.columns(2)
        with col1:
            start_time = st.selectbox('Start:', start_list)
        with col2:
            end_time = st.selectbox('End:', end_list)

        col3, col4 = st.columns(2)
        with col3:
            group = st.selectbox('Group: ', group_list)
        with col4:
            date = st.date_input('Date:', datetime.date(2023, 1, 1))

        if st.button('Submit', key='submit'):
            if 'submitted' not in st.session_state:
                st.session_state['submitted'] = True


    if 'submitted' in st.session_state:
        if st.session_state.submitted:
            with st.spinner('Retriving Data from database ...'):
                subject_id = get_subject_id(subject=subject, group_name=group, start_time=start_time,
                                            end_time=end_time, date=date)

                if subject_id is False:
                    st.error('This schedule not found', icon="🚨")
                else:
                    req = requests.get(f"http://127.0.0.1:8000/attendance/{group}/{subject_id}")
                    dict = req.json()
                    list = pd.json_normalize(dict)
                    list.rename(columns={'student.name': 'name', 'student.surname':'surname',
                                         'student.student_code':'student code',
                                         'student.group_name':'group'}, inplace=True)
                    st.write(list)
```