Based on your constraints and requirements, here's a focused MVP specification with functional and non-functional requirements tailored for your 1-month timeline:

### Functional Requirements (Core MVP)

1. \*\*User Management\*\*

- Admin/Invigilator role: Create/manage sessions, enroll students, generate reports

- Student role: Biometric authentication only

2. \*\*Pre-enrollment Module\*\*

- Capture: Matricule, full name, department, level

- Store 2 fingerprint templates (index fingers)

- Department-level access control (HODs can only see their department)

3. \*\*Course & Session Management\*\*

- CSV upload for course lists (course code, name, matricules with CA marks)

- Create exam sessions: Course selection + time window

- Active session dashboard (real-time status)

4. \*\*Biometric Authentication\*\*

- 1:N fingerprint matching against department cohort

- Automatic session detection based on course enrollment

- 3-try fallback with manual override option

- CA mark display with complaint flagging

5. \*\*Reporting\*\*

- Real-time authenticated student list

- Post-session error reports (missing CA, failed auth)

- Complaint logs with timestamps

6. \*\*System Administration\*\*

- Department-level access control

- Session audit logs

- Biometric data encryption at rest

### Non-Functional Requirements (Prioritized)

1. \*\*Performance\*\*:

- Authentication response < 3s (100 students queued)

- Support 5 concurrent sessions per department

- Handle 500+ student records per department

2. \*\*Scalability\*\*:

- Modular architecture (school > department > level)

- Database partitioning by department

- Stateless authentication API

3. \*\*Reliability\*\*:

- Offline mode for mobile devices (local cache)

- Automatic session timeout

- Biometric template backup/restore

4. \*\*Security\*\*:

- JWT authentication

- Fingerprint template encryption

- Role-based access control

5. \*\*Usability\*\*:

- < 5-minute invigilator training

- Intuitive mobile UI (large buttons, clear status)

- Accessible error messages

6. \*\*Cost Efficiency\*\*:

- Work with $100-150 fingerprint scanners

- Deployable on mid-range Android devices

- Low bandwidth consumption

### Excluded from MVP

1. University-wide integration

2. Fee receipt validation

3. Form B verification

4. Multi-campus support

5. Real-time university dashboard

6. Facial recognition fallback

### Tech Stack Recommendation

| \*\*Component\*\* | \*\*Technology\*\* | \*\*Justification\*\* |

|---------------|----------------|-------------------|

| \*\*Backend\*\* | FastAPI (Python) | Async, lightweight, easy OAuth2 |

| \*\*Database\*\* | PostgreSQL + Redis | ACID compliance + cache for auth |

| \*\*Mobile\*\* | Flutter (Dart) | Single codebase for admin/student |

| \*\*Biometric\*\* | Futronic FS80 | Android-compatible, dust-resistant |

| \*\*Matching\*\* | SourceAFIS | Open-source, high accuracy |

| \*\*Deployment\*\* | Docker | Consistent environments |

| \*\*Auth\*\* | JWT | Stateless, scalable |

### Critical Path for 1-Month Delivery

\*\*Week 1: Core Infrastructure\*\*

```mermaid

gantt

title Week 1: Foundation

dateFormat YYYY-MM-DD

section Database

Schema design :a1, 2023-10-02, 2d

PostgreSQL setup :after a1, 3d

section Backend

FastAPI skeleton :2023-10-02, 2d

JWT auth implementation :2023-10-04, 3d

section Mobile

Flutter login UI :2023-10-03, 4d

```

\*\*Week 2: Enrollment & Session Mgmt\*\*

```mermaid

gantt

title Week 2: Core Modules

dateFormat YYYY-MM-DD

section Enrollment

Fingerprint capture UI :2023-10-09, 3d

Template storage API :2023-10-11, 4d

section Course Mgmt

CSV parser :2023-10-09, 2d

Session CRUD API :2023-10-11, 4d

```

\*\*Week 3: Authentication Engine\*\*

```mermaid

gantt

title Week 3: Biometric Core

dateFormat YYYY-MM-DD

section Matching

SourceAFIS integration :2023-10-16, 4d

Dept-level search algo :2023-10-19, 3d

section Mobile Auth

Scanner integration :2023-10-16, 3d

Session detection UI :2023-10-19, 3d

```

\*\*Week 4: Reporting & Polish\*\*

```mermaid

gantt

title Week 4: Finalization

dateFormat YYYY-MM-DD

section Reporting

Error log API :2023-10-23, 3d

PDF generator :2023-10-25, 3d

section Testing

NAHPI pilot dataset :2023-10-23, 2d

Stress testing :2023-10-25, 3d

Deployment prep :2023-10-27, 2d

```

### Department-Focused Data Model

```sql

-- Hierarchical structure for NAHPI

CREATE TABLE departments (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL -- Civil, Computer, etc.

);

CREATE TABLE levels (

id SERIAL PRIMARY KEY,

department\_id INT REFERENCES departments(id),

name VARCHAR(10) NOT NULL -- Level 300, Level 400

);

-- Minimal student info for MVP

CREATE TABLE students (

id VARCHAR(20) PRIMARY KEY, -- Matricule

full\_name VARCHAR(100) NOT NULL,

level\_id INT REFERENCES levels(id),

fingerprint1 BYTEA NOT NULL, -- Encrypted template

fingerprint2 BYTEA NOT NULL

);

-- Course registry

CREATE TABLE courses (

code VARCHAR(10) PRIMARY KEY,

name VARCHAR(100) NOT NULL,

ca\_data JSONB -- {matricule: ca\_mark}

);

-- Exam session

CREATE TABLE sessions (

id UUID PRIMARY KEY,

course\_code VARCHAR(10) REFERENCES courses(code),

start TIMESTAMPTZ NOT NULL,

"end" TIMESTAMPTZ NOT NULL,

department\_id INT REFERENCES departments(id)

);

```

### Authentication Workflow Pseudocode

```python

def authenticate(fingerprint\_template, location):

# 1. Get department from location

dept = get\_department(location)

# 2. Perform department-scoped search

matches = sourceafis.search(

template=fingerprint\_template,

candidates=Student.objects.filter(department=dept)

)

# 3. Find active sessions for matched student

if match.score > 0.85:

active\_sessions = Session.get\_active\_for\_student(match.student.id)

# 4. Verify CA eligibility

for session in active\_sessions:

if session.course.ca\_data.get(match.student.id) is not None:

return {

"status": "AUTHENTICATED",

"student": match.student,

"session": session,

"ca\_mark": session.course.ca\_data[match.student.id]

}

return {"status": "NO\_VALID\_SESSION"}

return {"status": "NO\_MATCH"}

```

### Validation Strategy for MVP

1. \*\*Test Data\*\*:

- 5 departments

- 3 levels per department

- 50 students per level

- 2 courses per level

2. \*\*Hardware Simulation\*\*:

- Use fingerprint images from public datasets

- Android device emulator with virtual scanner

3. \*\*Success Metrics\*\*:

- 95%+ authentication accuracy

- < 5% false rejection rate

- 30 students/minute throughput

This specification gives you a deployable MVP for NAHPI that solves the core problems of:

1. Reducing invigilator workload by 70%+

2. Eliminating impersonation

3. Automating CA validation

The department-centric design ensures scalability while keeping initial complexity manageable. Focus on Computer Engineering department first before expanding to other departments in NAHPI.

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from sqlalchemy import create\_engine, Column, String, Binary, Integer, Float, DateTime, Boolean, ForeignKey

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from datetime import datetime

import base64 # Placeholder for fingerprint template encoding

app = FastAPI()

# Database setup

DATABASE\_URL = "postgresql://user:password@localhost:5432/biometric\_db"

engine = create\_engine(DATABASE\_URL)

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

Base = declarative\_base()

# Database Models

class Student(Base):

\_\_tablename\_\_ = "students"

matriculation\_number = Column(String, primary\_key=True)

name = Column(String, nullable=False)

fingerprint\_template = Column(Binary, nullable=False)

class CourseList(Base):

\_\_tablename\_\_ = "course\_lists"

id = Column(Integer, primary\_key=True)

course\_code = Column(String, nullable=False)

matriculation\_number = Column(String, ForeignKey("students.matriculation\_number"))

ca\_mark = Column(Float)

class ExamSession(Base):

\_\_tablename\_\_ = "exam\_sessions"

id = Column(Integer, primary\_key=True)

course\_code = Column(String, nullable=False)

course\_name = Column(String, nullable=False)

start\_time = Column(DateTime, nullable=False)

end\_time = Column(DateTime, nullable=False)

class Attendance(Base):

\_\_tablename\_\_ = "attendance"

id = Column(Integer, primary\_key=True)

session\_id = Column(Integer, ForeignKey("exam\_sessions.id"))

matriculation\_number = Column(String, ForeignKey("students.matriculation\_number"))

authenticated = Column(Boolean, default=False)

timestamp = Column(DateTime, default=datetime.utcnow)

class ErrorLog(Base):

\_\_tablename\_\_ = "error\_logs"

id = Column(Integer, primary\_key=True)

session\_id = Column(Integer, ForeignKey("exam\_sessions.id"))

matriculation\_number = Column(String, ForeignKey("students.matriculation\_number"))

error\_type = Column(String)

details = Column(String)

timestamp = Column(DateTime, default=datetime.utcnow)

Base.metadata.create\_all(bind=engine)

# Pydantic Models

class StudentCreate(BaseModel):

matriculation\_number: str

name: str

fingerprint\_template: str # Base64-encoded for simplicity

class AuthenticationRequest(BaseModel):

fingerprint\_template: str

session\_id: int

class ErrorReport(BaseModel):

session\_id: int

matriculation\_number: str

error\_type: str

details: str

# Endpoints

@app.post("/enroll")

async def enroll\_student(student: StudentCreate):

db = SessionLocal()

try:

# Decode fingerprint template (placeholder)

fingerprint\_data = base64.b64decode(student.fingerprint\_template)

db\_student = Student(

matriculation\_number=student.matriculation\_number,

name=student.name,

fingerprint\_template=fingerprint\_data

)

db.add(db\_student)

db.commit()

return {"message": "Student enrolled successfully"}

except Exception as e:

db.rollback()

raise HTTPException(status\_code=400, detail=str(e))

finally:

db.close()

@app.post("/authenticate")

async def authenticate\_student(auth: AuthenticationRequest):

db = SessionLocal()

try:

# Placeholder: Match fingerprint (use scanner SDK in production)

fingerprint\_data = base64.b64decode(auth.fingerprint\_template)

student = db.query(Student).filter(Student.fingerprint\_template == fingerprint\_data).first()

if not student:

raise HTTPException(status\_code=404, detail="Student not found")

# Check if session is active

session = db.query(ExamSession).filter(

ExamSession.id == auth.session\_id,

ExamSession.start\_time <= datetime.utcnow(),

ExamSession.end\_time >= datetime.utcnow()

).first()

if not session:

raise HTTPException(status\_code=400, detail="Invalid or expired session")

# Check course list for CA mark

course\_entry = db.query(CourseList).filter(

CourseList.course\_code == session.course\_code,

CourseList.matriculation\_number == student.matriculation\_number

).first()

if not course\_entry:

raise HTTPException(status\_code=400, detail="Student not enrolled in course")

# Mark as authenticated

attendance = Attendance(

session\_id=auth.session\_id,

matriculation\_number=student.matriculation\_number,

authenticated=True

)

db.add(attendance)

db.commit()

return {

"matriculation\_number": student.matriculation\_number,

"name": student.name,

"ca\_mark": course\_entry.ca\_mark,

"session\_id": session.id

}

except Exception as e:

db.rollback()

raise HTTPException(status\_code=400, detail=str(e))

finally:

db.close()

@app.post("/report-error")

async def report\_error(error: ErrorReport):

db = SessionLocal()

try:

error\_log = ErrorLog(

session\_id=error.session\_id,

matriculation\_number=error.matriculation\_number,

error\_type=error.error\_type,

details=error.details

)

db.add(error\_log)

db.commit()

return {"message": "Error reported successfully"}

except Exception as e:

db.rollback()

raise HTTPException(status\_code=400, detail=str(e))

finally:

db.close()

@app.get("/reports/{session\_id}")

async def get\_reports(session\_id: int):

db = SessionLocal()

try:

# Authenticated students

authenticated = db.query(Attendance).filter(

Attendance.session\_id == session\_id,

Attendance.authenticated == True

).all()

# Error logs

errors = db.query(ErrorLog).filter(ErrorLog.session\_id == session\_id).all()

return {

"authenticated": [

{"matriculation\_number": a.matriculation\_number, "timestamp": a.timestamp}

for a in authenticated

],

"errors": [

{"matriculation\_number": e.matriculation\_number, "error\_type": e.error\_type, "details": e.details}

for e in errors

]

}

except Exception as e:

raise HTTPException(status\_code=400, detail=str(e))

finally:

db.close()

CREATE TABLE departments (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL, -- e.g., Computer Engineering

school VARCHAR(100) NOT NULL -- e.g., NAHPI

);

CREATE TABLE levels (

id SERIAL PRIMARY KEY,

department\_id INTEGER REFERENCES departments(id),

level\_name VARCHAR(50) NOT NULL -- e.g., Level 100, Level 200

);

CREATE TABLE courses (

id SERIAL PRIMARY KEY,

department\_id INTEGER REFERENCES departments(id),

level\_id INTEGER REFERENCES levels(id),

course\_code VARCHAR(50) UNIQUE NOT NULL, -- e.g., MATENG2101

course\_name VARCHAR(100) NOT NULL -- e.g., Engineering Maths

);

CREATE TABLE students (

matriculation\_number VARCHAR(50) PRIMARY KEY,

name VARCHAR(100) NOT NULL,

department\_id INTEGER REFERENCES departments(id),

level\_id INTEGER REFERENCES levels(id),

fingerprint\_template BYTEA NOT NULL -- Encrypted fingerprint

);

CREATE TABLE course\_lists (

id SERIAL PRIMARY KEY,

course\_id INTEGER REFERENCES courses(id),

matriculation\_number VARCHAR(50) REFERENCES students(matriculation\_number),

ca\_mark FLOAT CHECK (ca\_mark >= 0 AND ca\_mark <= 30) -- Assuming CA is out of 30

);

CREATE TABLE exam\_sessions (

id SERIAL PRIMARY KEY,

course\_id INTEGER REFERENCES courses(id),

start\_time TIMESTAMP NOT NULL,

end\_time TIMESTAMP NOT NULL

);

CREATE TABLE attendance (

id SERIAL PRIMARY KEY,

session\_id INTEGER REFERENCES exam\_sessions(id),

matriculation\_number VARCHAR(50) REFERENCES students(matriculation\_number),

authenticated BOOLEAN DEFAULT FALSE,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE error\_logs (

id SERIAL PRIMARY KEY,

session\_id INTEGER REFERENCES exam\_sessions(id),

matriculation\_number VARCHAR(50) REFERENCES students(matriculation\_number),

error\_type VARCHAR(50), -- e.g., CA\_MARK\_ISSUE, AUTH\_FAILED

details TEXT,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE admins (

id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL -- Hashed password

);

-- Indexes for performance

CREATE INDEX idx\_students\_fingerprint ON students USING hash (fingerprint\_template);

CREATE INDEX idx\_course\_lists\_course\_id ON course\_lists (course\_id);

CREATE INDEX idx\_exam\_sessions\_start\_time ON exam\_sessions (start\_time);  
  
  
  
from fastapi import FastAPI, HTTPException, Depends, Security

from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials

from pydantic import BaseModel

from sqlalchemy import create\_engine, Column, Integer, String, Binary, Float, DateTime, Boolean, ForeignKey

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker, Session

from jose import JWTError, jwt

from passlib.context import CryptContext

from datetime import datetime, timedelta

import base64

import os

app = FastAPI()

security = HTTPBearer()

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

SECRET\_KEY = os.urandom(32).hex() # Replace with secure key in production

ALGORITHM = "HS256"

# Database Setup

DATABASE\_URL = "postgresql://user:password@localhost:5432/biometric\_db"

engine = create\_engine(DATABASE\_URL)

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

Base = declarative\_base()

# SQLAlchemy Models

class University(Base):

\_\_tablename\_\_ = "universities"

university\_id = Column(Integer, primary\_key=True)

name = Column(String, nullable=False)

class School(Base):

\_\_tablename\_\_ = "schools"

school\_id = Column(Integer, primary\_key=True)

name = Column(String, nullable=False)

university\_id = Column(Integer, ForeignKey("universities.university\_id"))

class Department(Base):

\_\_tablename\_\_ = "departments"

department\_id = Column(Integer, primary\_key=True)

name = Column(String, nullable=False)

school\_id = Column(Integer, ForeignKey("schools.school\_id"))

class Level(Base):

\_\_tablename\_\_ = "levels"

level\_id = Column(Integer, primary\_key=True)

name = Column(String, nullable=False)

department\_id = Column(Integer, ForeignKey("departments.department\_id"))

class Course(Base):

\_\_tablename\_\_ = "courses"

course\_id = Column(Integer, primary\_key=True)

course\_code = Column(String, unique=True, nullable=False)

course\_name = Column(String, nullable=False)

department\_id = Column(Integer, ForeignKey("departments.department\_id"))

level\_id = Column(Integer, ForeignKey("levels.level\_id"))

class Student(Base):

\_\_tablename\_\_ = "students"

matriculation\_number = Column(String, primary\_key=True)

name = Column(String, nullable=False)

department\_id = Column(Integer, ForeignKey("departments.department\_id"))

level\_id = Column(Integer, ForeignKey("levels.level\_id"))

fingerprint\_template = Column(Binary, nullable=False)

class CourseList(Base):

\_\_tablename\_\_ = "course\_lists"

id = Column(Integer, primary\_key=True)

course\_id = Column(Integer, ForeignKey("courses.course\_id"))

matriculation\_number = Column(String, ForeignKey("students.matriculation\_number"))

ca\_mark = Column(Float, nullable=True) # Nullable for initial enrollment

class ExamSession(Base):

\_\_tablename\_\_ = "exam\_sessions"

session\_id = Column(Integer, primary\_key=True)

course\_id = Column(Integer, ForeignKey("courses.course\_id"))

admin\_id = Column(Integer, ForeignKey("admins.admin\_id"))

start\_time = Column(DateTime, nullable=False)

end\_time = Column(DateTime, nullable=False)

class Attendance(Base):

\_\_tablename\_\_ = "attendance"

id = Column(Integer, primary\_key=True)

session\_id = Column(Integer, ForeignKey("exam\_sessions.session\_id"))

matriculation\_number = Column(String, ForeignKey("students.matriculation\_number"))

authenticated = Column(Boolean, default=False)

timestamp = Column(DateTime, default=datetime.utcnow)

class ErrorLog(Base):

\_\_tablename\_\_ = "error\_logs"

id = Column(Integer, primary\_key=True)

session\_id = Column(Integer, ForeignKey("exam\_sessions.session\_id"))

matriculation\_number = Column(String, ForeignKey("students.matriculation\_number"))

error\_type = Column(String)

details = Column(String)

timestamp = Column(DateTime, default=datetime.utcnow)

class Admin(Base):

\_\_tablename\_\_ = "admins"

admin\_id = Column(Integer, primary\_key=True)

username = Column(String, unique=True, nullable=False)

password\_hash = Column(String, nullable=False)

department\_id = Column(Integer, ForeignKey("departments.department\_id"))

Base.metadata.create\_all(bind=engine)

# Pydantic Models

class AdminLogin(BaseModel):

username: str

password: str

class StudentCreate(BaseModel):

matriculation\_number: str

name: str

department\_id: int

level\_id: int

fingerprint\_template: str # Base64-encoded for now

course\_ids: list[int] # List of course IDs for enrollment

# Authentication

def verify\_password(plain\_password, hashed\_password):

return pwd\_context.verify(plain\_password, hashed\_password)

def create\_access\_token(data: dict):

to\_encode = data.copy()

expire = datetime.utcnow() + timedelta(minutes=60)

to\_encode.update({"exp": expire})

return jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)

def get\_current\_admin(credentials: HTTPAuthorizationCredentials = Security(security), db: Session = Depends(get\_db)):

try:

payload = jwt.decode(credentials.credentials, SECRET\_KEY, algorithms=[ALGORITHM])

username: str = payload.get("sub")

if username is None:

raise HTTPException(status\_code=401, detail="Invalid token")

admin = db.query(Admin).filter(Admin.username == username).first()

if not admin:

raise HTTPException(status\_code=401, detail="Admin not found")

return admin

except JWTError:

raise HTTPException(status\_code=401, detail="Invalid token")

# Dependency

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

# Endpoints

@app.post("/login")

async def login(admin: AdminLogin, db: Session = Depends(get\_db)):

db\_admin = db.query(Admin).filter(Admin.username == admin.username).first()

if not db\_admin or not verify\_password(admin.password, db\_admin.password\_hash):

raise HTTPException(status\_code=401, detail="Invalid credentials")

token = create\_access\_token({"sub": admin.username})

return {"access\_token": token, "token\_type": "bearer"}

@app.post("/enroll", dependencies=[Depends(get\_current\_admin)])

async def enroll\_student(student: StudentCreate, admin: Admin = Depends(get\_current\_admin), db: Session = Depends(get\_db)):

try:

# Validate department and level

department = db.query(Department).filter(Department.department\_id == student.department\_id).first()

if not department or department.department\_id != admin.department\_id:

raise HTTPException(status\_code=403, detail="Not authorized for this department")

level = db.query(Level).filter(Level.level\_id == student.level\_id, Level.department\_id == student.department\_id).first()

if not level:

raise HTTPException(status\_code=400, detail="Invalid level for department")

# Validate matriculation number format (e.g., NAHPI/2023/1234)

if not student.matriculation\_number.startswith("NAHPI/"):

raise HTTPException(status\_code=400, detail="Invalid matriculation number format")

# Validate courses

for course\_id in student.course\_ids:

course = db.query(Course).filter(Course.course\_id == course\_id, Course.department\_id == student.department\_id, Course.level\_id == student.level\_id).first()

if not course:

raise HTTPException(status\_code=400, detail=f"Course {course\_id} invalid for department or level")

# Decode fingerprint (placeholder; replace with SDK)

fingerprint\_data = base64.b64decode(student.fingerprint\_template)

# Enroll student

db\_student = Student(

matriculation\_number=student.matriculation\_number,

name=student.name,

department\_id=student.department\_id,

level\_id=student.level\_id,

fingerprint\_template=fingerprint\_data

)

db.add(db\_student)

# Enroll in courses (CourseList entries)

for course\_id in student.course\_ids:

db\_course\_list = CourseList(

course\_id=course\_id,

matriculation\_number=student.matriculation\_number,

ca\_mark=None # CA mark to be updated later

)

db.add(db\_course\_list)

db.commit()

return {"message": "Student enrolled successfully"}

except Exception as e:

db.rollback()

raise HTTPException(status\_code=400, detail=str(e))

# Placeholder for future verification endpoint

@app.post("/verify-enrollment")

async def verify\_enrollment(fingerprint\_template: str, db: Session = Depends(get\_db)):

try:

fingerprint\_data = base64.b64decode(fingerprint\_template)

student = db.query(Student).filter(Student.fingerprint\_template == fingerprint\_data).first()

if not student:

raise HTTPException(status\_code=404, detail="Student not found")

# Get enrolled courses

course\_lists = db.query(CourseList).join(Course).filter(

CourseList.matriculation\_number == student.matriculation\_number

).all()

return {

"matriculation\_number": student.matriculation\_number,

"name": student.name,

"department\_id": student.department\_id,

"level\_id": student.level\_id,

"courses": [

{"course\_id": cl.course\_id, "course\_code": cl.course.course\_code, "ca\_mark": cl.ca\_mark}

for cl in course\_lists

]

}

except Exception as e:

raise HTTPException(status\_code=400, detail=str(e))