```python
# Activate python virtual enviroment and then start the uvicorn server.
# cd scripts; ./activate; cd .. ; uvicorn main:app --reload

from array import array
from lib2to3.pytree import Base
from optparse import Option
from typing import Optional
from urllib import response
from fastapi import FastAPI, Query
from pymongo import MongoClient
from bson.objectid import ObjectId
from bson.json_util import dumps as bson_dumps
from json import loads as json_loads
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel

DB = MongoClient("localhost")["GibJohn"]

COLLECTION = {
    "student-accounts": DB["student-accounts"],
    "teacher-accounts": DB["teacher-accounts"]
}

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

def BsonToJson(bson_data):
    parsed_data = json_loads(bson_dumps(bson_data))
    try:
        parsed_data["_id"] = parsed_data["_id"]["$oid"]
    except:
        pass
    return parsed_data

@app.get("/")
def root() -> dict:
    def _checkDB() -> dict:
        try:
            response = MongoClient("localhost",
serverSelectionTimeoutMS=2000).server_info()
            print(BsonToJson(response))
            return "running"
        except:
            return "down"

    return{
        "status":{
            "backend": "running",
            "database": _checkDB()
        }
    }


```

```python
def Document(collection_name:str, query: dict, return_id:bool = False) -> dict:
    response = COLLECTION[collection_name].find_one(query)
    parsed_response = BsonToJson(response)

    if parsed_response in (None, "null", "Null"):
        return {"exists": False}

    if return_id == True:
        return {"exists": True, "_id": parsed_response["_id"] }
    return {"exists": True, "_id": None}

class RegisterDetails(BaseModel):
    email: str = Query(default= "AnEmailAddress@gmail.com", min_length=3, regex="[a-z]+@[a-z]+.[a-z]+") # Fixed the regex for emails. "a@a.com" was not working before.
    # Need to add validation
    password: str = Query(default= "Ex@mple!Pa55w0rd92", min_length=8)
    date_of_birth: str = Query(default = "03-12-2022", min_length=10, regex="[0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]")


#Student register
@app.post("/register/student")
def StudentRegister(student_details:RegisterDetails) -> dict:
    """Endpoint for students to login.

    Args:
        StudentRegister (student_details:RegisterDetails): Endpoint, to allow for student to register.

    Returns:
        first output - dict: Error message telling the user, that an email given already exists.
        second output - dict: Success message, and returns the ID of the object stored in the database.
        third output - dict: Error message, telling the user the request was nott able to send to the database
    """
    if Document(collection_name= "student-accounts", query= {"email": student_details.email})["exists"]:
        return "Email already exists."

    print(f"STUDENT_DETAILS:\n{student_details}")
    print(student_details.date_of_birth)
    try:
        response = COLLECTION["student-accounts"].insert_one({
            "email": student_details.email,
            "password": student_details.password,
            "date_of_birth": student_details.date_of_birth
        })
        return {
            "status": "success",
            "response" : str(response.inserted_id)
        }
    except:
        return {
            "status":"fail",
            "reason": "Was not able to send request."
        }

#Teacher register
```

```python
@app.post("/register/teacher")
def TeacherRegister(teacher_details:RegisterDetails) -> dict:
    """Endpoint for Teachers to login.

    Args:
        TeacherRegister (teacher_details:RegisterDetails): Endpoint, to allow for student to register.

    Returns:
        first output - dict: Error message telling the user, that an email given already exists.
        second output - dict: Success message, and returns the ID of the object stored in the database.
        third output - dict: Error message, telling the user the request was nott able to send to the database
    """
    if Document(collection_name= "student-accounts", query= {"email": teacher_details.email})["exists"]:
        return "Email already exists."

    try:
        response = COLLECTION["student-accounts"].insert_one({
            "email": teacher_details.email,
            "password": teacher_details.password,
            "date_of_birth": teacher_details.date_of_birth
        })
        return {
            "status": "success",
            "response" : str(response.inserted_id)
        }
    except:
        return {
            "status":"fail",
            "reason": "Was not able to send request."
        }

# Need to add validation
class LoginDetails(BaseModel):
    email: str
    password: str #= Query(default= "Ex@mple!Pa55w0rd92", min_length=8)

@app.post("/login/student")
def StudentLogin(login_details: LoginDetails):

    response = COLLECTION["student-accounts"].find_one({"email": login_details.email, "password": login_details.password})
    response = BsonToJson(response)

    if response in (None, "null", "Null"):
        return {"exists": False}

    return {"exists": True, "user_id": response["_id"], "name": response["email"].split("@")[0]}

@app.post("/login/teacher")
def TeacherLogin(login_details: LoginDetails):
    document = Document(collection_name= "teacher-accounts", query= {"email": login_details.email, "password": login_details.password}, return_id= True)
    print(document)
    if document["exists"]:
        return {"exists":True, "user_id": document["_id"]}
    return {"exists":False, "user_id": None}

class UserQuery(BaseModel):
    student: bool
    id: str
@app.post("/user/quiz/stats/overall")
def QuizStats(user:UserQuery):
    #Serving hard coded values, so that we can see how the pie chart looks like on the frontend.
    #Otherwise, the logic within the docstring is tested, and works fine.
    """
    if user.student == True:
        if Document(collection_name="student-accounts", query={"_id": ObjectId(user.id)})["exists"]:
            user_stats_query = COLLECTION["quiz-stats"].find_one({"_id": ObjectId(user.id)})
            user_stats_query = BsonToJson(user_stats_query)
            return user_stats_query
        else:
            return "User does not exist."
    else:
        if Document(collection_name="teacher-accounts", query={"_id": ObjectId(user.id)})["exists"]:
            user_stats_query = COLLECTION["quiz-stats"].find_one({"_id": ObjectId(user.id)})
            user_stats_query = BsonToJson(user_stats_query)
            return user_stats_query
        else:
            return "User does not exist."
    """

    return {"done": 24, "attempted": 23,"remaining": 5}


@app.post("/user/classes")
def UserClasses(user:UserQuery):
    print(user)
    print({"class":{"name": "Wow", "image_path": "/", "path": "/"}})

    if Document(collection_name="student-accounts", query={"_id": ObjectId(user.id)})["exists"]:
        user_classes = COLLECTION["student-accounts"].find_one({"_id": ObjectId(user.id)})
        user_classes = BsonToJson(user_classes)
        return user_classes["owned-classes"]

    elif Document(collection_name="teacher-accounts", query={"_id": ObjectId(user.id)})["exists"]:
        user_classes = COLLECTION["teacher-accounts"].find_one({"_id": ObjectId(user.id)})
        user_classes = BsonToJson(user_classes)
        return user_classes["owned-classes"]

    else:
        return "Error finding user."


class CreateClass(BaseModel):
    name: str
    owner_id: str
    organisation_id: Optional[str] = None

@app.post("/user/classes/create")
def CreateClass(the_class:CreateClass):
    if Document(collection_name="student-accounts", query={"_id": ObjectId(the_class.owner_id)})["exists"]:
        try:
            COLLECTION["student-accounts"].update_one({"_id": ObjectId(the_class.owner_id)}, {"$push":{"owned-classes": {
                "name": the_class.name,
                "student_ids": [],
                "organisation_id": the_class.organisation_id
            }}})
            return {"updated": True}
        except:
            return {"updated": False}


    elif Document(collection_name="teacher-accounts", query={"_id": ObjectId(the_class.owner_id)})["exists"]:
        try:
            COLLECTION["teacher-accounts"].update_one({"_id": ObjectId(the_class.owner_id)}, {"$push":{"owned-classes": {
                "name": the_class.name,
                "student_ids": [],
                "organisation_id": the_class.organisation_id
            }}})
            return {"updated": True}
        except:
            return {"updated": False}

    else:
        return "Error finding user."

```