

```
1 # Activate python virtual enviroment and then start the uvicorn server.
2 # cd scripts; ./activate; cd .. ; uvicorn main:app --reload
3
4 from array import array
5 from lib2to3.pytree import Base
6 from optparse import Option
7 from typing import Optional
8 from urllib import response
9 from fastapi import FastAPI, Query
10 from pymongo import MongoClient
11 from bson.objectid import ObjectId
12 from bson.json_util import dumps as bson_dumps
13 from json import loads as json_loads
14 from fastapi.middleware.cors import CORSMiddleware
15 from pydantic import BaseModel
16
17 DB = MongoClient("localhost")["GibJohn"]
18
19 COLLECTION = {
20     "student-accounts": DB["student-accounts"],
21     "teacher-accounts": DB["teacher-accounts"]
22 }
23
24 app = FastAPI()
25 app.add_middleware(
26     CORSMiddleware,
27     allow_origins=["*"],
28     allow_credentials=True,
29     allow_methods=["*"],
30     allow_headers=["*"],
31 )
32
33 def BsonToJson(bson_data):
34     parsed_data = json_loads(bson_dumps(bson_data))
35     try:
36         parsed_data["_id"] = parsed_data["_id"]["$oid"]
37     except:
38         pass
39     return parsed_data
40
41 @app.get("/")
42 def root() -> dict:
43     def _checkDB() -> dict:
44         try:
45             response = MongoClient("localhost",
serverSelectionTimeoutMS=2000).server_info()
46             print(BsonToJson(response))
47             return "running"
48         except:
49             return "down"
50
51     return{
52         "status":{
53             "backend": "running",
54             "database": _checkDB()
55         }
56     }
57
58
```

```

59 def Document(collection_name:str, query: dict, return_id:bool = False) -> dict:
60     response = COLLECTION[collection_name].find_one(query)
61     parsed_response = BsonToJson(response)
62
63     if parsed_response in (None, "null", "Null"):
64         return {"exists": False}
65
66     if return_id == True:
67         return {"exists": True, "_id": parsed_response["_id"]}
68     return {"exists": True, "_id": None}
69
70 class RegisterDetails(BaseModel):
71     email: str = Query(default= "AnEmailAddress@gmail.com", min_length=3, regex="[a-
72     z]+@[a-z]+.[a-z]+") # Fixed the regex for emails. "a@a.com" was not working before.
73     # Need to add validation
74     password: str = Query(default= "Ex@mp!e!Pa55w0rd92", min_length=8)
75     date_of_birth: str = Query(default= "03-12-2022", min_length=10, regex="[0-9][0-
76     9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]")
77
78 #Student register
79 @app.post("/register/student")
80 def StudentRegister(student_details:RegisterDetails) -> dict:
81     """Endpoint for students to login.
82
83     Args:
84         StudentRegister (student_details:RegisterDetails): Endpoint, to allow for
85         student to register.
86
87     Returns:
88         first output - dict: Error message telling the user, that an email given
89         already exists.
90         second output - dict: Success message, and returns the ID of the object
91         stored in the database.
92         third output - dict: Error message, telling the user the request was nott
93         able to send to the database
94     """
95     if Document(collection_name= "student-accounts", query= {"email":
96     student_details.email})["exists"]:
97         return "Email already exists."
98
99     print(f"STUDENT_DETAILS:\n{student_details}")
100     print(student_details.date_of_birth)
101     try:
102         response = COLLECTION["student-accounts"].insert_one({
103             "email": student_details.email,
104             "password": student_details.password,
105             "date_of_birth": student_details.date_of_birth
106         })
107         return {
108             "status": "success",
109             "response" : str(response.inserted_id)
110         }
111     except:
112         return {
113             "status":"fail",
114             "reason": "Was not able to send request."
115         }
116
117 #Teacher register

```

```
112 @app.post("/register/teacher")
113 def TeacherRegister(teacher_details:RegisterDetails) -> dict:
114     """Endpoint for Teachers to login.
115
116     Args:
117         TeacherRegister (teacher_details:RegisterDetails): Endpoint, to allow for
student to register.
118
119     Returns:
120         first output - dict: Error message telling the user, that an email given
already exists.
121         second output - dict: Success message, and returns the ID of the object
stored in the database.
122         third output - dict: Error message, telling the user the request was nott
able to send to the database
123     """
124     if Document(collection_name= "student-accounts", query= {"email":
teacher_details.email})["exists"]:
125         return "Email already exists."
126
127     try:
128         response = COLLECTION["student-accounts"].insert_one({
129             "email": teacher_details.email,
130             "password": teacher_details.password,
131             "date_of_birth": teacher_details.date_of_birth
132         })
133         return {
134             "status": "success",
135             "response" : str(response.inserted_id)
136         }
137     except:
138         return {
139             "status":"fail",
140             "reason": "Was not able to send request."
141         }
142
143 # Need to add validation
144 class LoginDetails(BaseModel):
145     email: str
146     password: str = Query(default= "Ex@mp!e!Pa55w0rd92", min_length=8)
147
148 @app.post("/login/student")
149 def StudentLogin(login_details: LoginDetails):
150
151     response = COLLECTION["student-accounts"].find_one({"email": login_details.email,
"password": login_details.password})
152     response = BsonToJson(response)
153
154     if response in (None, "null", "Null"):
155         return {"exists": False}
156
157     return {"exists": True, "user_id": response["_id"], "name":
response["email"].split("@")[0]}
158
159 @app.post("/login/teacher")
160 def TeacherLogin(login_details: LoginDetails):
161     document = Document(collection_name= "teacher-accounts", query= {"email":
login_details.email, "password": login_details.password}, return_id= True)
162     print(document)
163     if document["exists"]:
```

```
164         return {"exists": True, "user_id": document["_id"]}
165     return {"exists": False, "user_id": None}
166
167 class UserQuery(BaseModel):
168     student: bool
169     id: str
170 @app.post("/user/quiz/stats/overall")
171 def QuizStats(user: UserQuery):
172     #Serving hard coded values, so that we can see how the pie chart looks like on
the frontend.
173     #Otherwise, the logic within the docstring is tested, and works fine.
174     """
175     if user.student == True:
176         if Document(collection_name="student-accounts", query={"_id":
ObjectId(user.id)})["exists"]:
177             user_stats_query = COLLECTION["quiz-stats"].find_one({"_id":
ObjectId(user.id)})
178             user_stats_query = BsonToJson(user_stats_query)
179             return user_stats_query
180         else:
181             return "User does not exist."
182     else:
183         if Document(collection_name="teacher-accounts", query={"_id":
ObjectId(user.id)})["exists"]:
184             user_stats_query = COLLECTION["quiz-stats"].find_one({"_id":
ObjectId(user.id)})
185             user_stats_query = BsonToJson(user_stats_query)
186             return user_stats_query
187         else:
188             return "User does not exist."
189     """
190
191     return {"done": 24, "attempted": 23, "remaining": 5}
192
193
194 @app.post("/user/classes")
195 def UserClasses(user: UserQuery):
196     print(user)
197     print({"class": {"name": "Wow", "image_path": "/", "path": "/"}})
198
199     if Document(collection_name="student-accounts", query={"_id": ObjectId(user.id)})
["exists"]:
200         user_classes = COLLECTION["student-accounts"].find_one({"_id":
ObjectId(user.id)})
201         user_classes = BsonToJson(user_classes)
202         return user_classes["owned-classes"]
203
204     elif Document(collection_name="teacher-accounts", query={"_id":
ObjectId(user.id)})["exists"]:
205         user_classes = COLLECTION["teacher-accounts"].find_one({"_id":
ObjectId(user.id)})
206         user_classes = BsonToJson(user_classes)
207         return user_classes["owned-classes"]
208
209     else:
210         return "Error finding user."
211
212
213 class CreateClass(BaseModel):
214     name: str
```

```
215     owner_id: str
216     organisation_id: Optional[str] = None
217
218 @app.post("/user/classes/create")
219 def CreateClass(the_class: CreateClass):
220     if Document(collection_name="student-accounts", query={"_id":
221 ObjectId(the_class.owner_id)})["exists"]:
222         try:
223             COLLECTION["student-accounts"].update_one({"_id":
224 ObjectId(the_class.owner_id)}, {"$push":{"owned-classes": {
225                 "name": the_class.name,
226                 "student_ids": [],
227                 "organisation_id": the_class.organisation_id
228             }}})
229             return {"updated": True}
230         except:
231             return {"updated": False}
232
233     elif Document(collection_name="teacher-accounts", query={"_id":
234 ObjectId(the_class.owner_id)})["exists"]:
235         try:
236             COLLECTION["teacher-accounts"].update_one({"_id":
237 ObjectId(the_class.owner_id)}, {"$push":{"owned-classes": {
238                 "name": the_class.name,
239                 "student_ids": [],
240                 "organisation_id": the_class.organisation_id
241             }}})
242             return {"updated": True}
243         except:
244             return {"updated": False}
245
246     else:
247         return "Error finding user."
```