



PYTHON BOOTCAMP

www.jomhack.com

FASTAPI (MONGO)



FastAPI:

- Web framework for building APIs (endpoint)
- Type hints support
- Interactive API documentation (Swagger UI)
- `pip install fastapi uvicorn`

Pydantic:

- Data validation and parsing using Python type hints.
- IDE support with type hints
- JSON serialization/deserialization (Python to JSON to Python)
- `pip install pydantic`

FASTAPI (MONGO)



Hypertext Transfer Protocol(HTTP) Methods:

- @app.get() - GET requests (Read)
- @app.post() - POST requests (Create)
- @app.put() - PUT requests (Update)
- @app.delete() - DELETE requests (Delete)
- @app.patch() - PATCH requests (Partial Update)

FASTAPI (MONGO)



Imports & Initialization:

```
2  from fastapi import FastAPI, HTTPException, status
3  from contextlib import asynccontextmanager
4  from pydantic import BaseModel, EmailStr
5  from typing import List, Optional
6  from datetime import datetime
7  from bson.objectid import ObjectId
8  from mongo_database import DatabaseManager
9  import os
10 from dotenv import load_dotenv
11
12 load_dotenv()
13
14 app = FastAPI(title="MongoDB Database API", version="1.0.0")
```

BaseModel:

```
16 # Pydantic models for request/response
17 class UserCreate(BaseModel):
18     name: str
19     email: EmailStr
20     age: int
21
22 class UserResponse(BaseModel):
23     id: str
24     name: str
25     email: str
26     age: int
27     created_at: datetime
28
29 class PostCreate(BaseModel):
30     user_id: str
31     title: str
32     content: str
33
34 class PostResponse(BaseModel):
35     id: str
36     user_id: str
37     title: str
38     content: str
39     created_at: datetime
40
41 class PostResponseForUser(BaseModel):
42     id: str
43     title: str
44     content: str
45     created_at: datetime
46
47 # Initialize database
48 try:
49     db = DatabaseManager()
50 except Exception as e:
51     print(f"Failed to connect to MongoDB: {e}")
52     db = None
```

FASTAPI (MONGO)



Event handler:

```
54 @app.on_event("startup")
55 async def startup_event():
56     if db is None:
57         raise Exception("Failed to connect to MongoDB")
58
59 @app.on_event("shutdown")
60 async def shutdown_event():
61     if db:
62         db.close_connection()
```

post(CREATE):

```
83 @app.get("/")
84 async def root():
85     return {"message": "MongoDB Database API", "version": "1.0.0"}
86
87 @app.post("/users/", response_model=dict, status_code=status.HTTP_201_CREATED)
88 async def create_user(user: UserCreate):
89     """Create a new user"""
90     try:
91         user_id = db.create_user(user.name, user.email, user.age)
92         if user_id:
93             return {"message": "User created successfully", "user_id": user_id}
94         else:
95             raise HTTPException(
96                 status_code=status.HTTP_400_BAD_REQUEST,
97                 detail="Failed to create user. Email might already exist."
98             )
99     except Exception as e:
100         raise HTTPException(
101             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
102             detail=f"Internal server error: {str(e)}"
103         )
```

FASTAPI (MONGO)



get(READ):

```
105 @app.get("/users/", response_model=List[UserResponse])
106 async def get_all_users():
107     """Get all users"""
108     try:
109         users = db.get_all_users()
110         return [
111             UserResponse(
112                 id=user['_id'],
113                 name=user['name'],
114                 email=user['email'],
115                 age=user['age'],
116                 created_at=user['created_at']
117             )
118             for user in users
119         ]
120     except Exception as e:
121         raise HTTPException(
122             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
123             detail=f"Internal server error: {str(e)}"
124         )
```

get(READ):

```
126 @app.get("/users/{user_id}", response_model=UserResponse)
127 async def get_user(user_id: str):
128     """Get a specific user by ID"""
129     try:
130         if not ObjectId.is_valid(user_id):
131             raise HTTPException(
132                 status_code=status.HTTP_400_BAD_REQUEST,
133                 detail="Invalid user ID format"
134             )
135
136         user = db.users_collection.find_one({"_id": ObjectId(user_id)})
137
138         if not user:
139             raise HTTPException(
140                 status_code=status.HTTP_404_NOT_FOUND,
141                 detail="User not found"
142             )
143
144         return UserResponse(
145             id=str(user['_id']),
146             name=user['name'],
147             email=user['email'],
148             age=user['age'],
149             created_at=user['created_at']
150         )
151     except HTTPException:
152         raise
153     except Exception as e:
154         raise HTTPException(
155             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
156             detail=f"Internal server error: {str(e)}"
157         )
```

FASTAPI (MONGO)



post(CREATE):

```
159 @app.post("/posts/", response_model=dict, status_code=status.HTTP_201_CREATED)
160 async def create_post(post: PostCreate):
161     """Create a new post"""
162     try:
163         if not ObjectId.is_valid(post.user_id):
164             raise HTTPException(
165                 status_code=status.HTTP_400_BAD_REQUEST,
166                 detail="Invalid user ID format"
167             )
168
169         # Check if user exists
170         user = db.users_collection.find_one({"_id": ObjectId(post.user_id)})
171         if not user:
172             raise HTTPException(
173                 status_code=status.HTTP_404_NOT_FOUND,
174                 detail="User not found"
175             )
176
177         post_id = db.create_post(post.user_id, post.title, post.content)
178         if post_id:
179             return {"message": "Post created successfully", "post_id": post_id}
180         else:
181             raise HTTPException(
182                 status_code=status.HTTP_400_BAD_REQUEST,
183                 detail="Failed to create post"
184             )
185     except HTTPException:
186         raise
187     except Exception as e:
188         raise HTTPException(
189             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
190             detail=f"Internal server error: {str(e)}"
191         )
```

get(READ):

```
193 @app.get("/users/{user_id}/posts", response_model=List[PostResponseForUser])
194 async def get_user_posts(user_id: str):
195     """Get all posts by a specific user"""
196     try:
197         if not ObjectId.is_valid(user_id):
198             raise HTTPException(
199                 status_code=status.HTTP_400_BAD_REQUEST,
200                 detail="Invalid user ID format"
201             )
202
203         # Check if user exists
204         user = db.users_collection.find_one({"_id": ObjectId(user_id)})
205         if not user:
206             raise HTTPException(
207                 status_code=status.HTTP_404_NOT_FOUND,
208                 detail="User not found"
209             )
210
211         posts = db.get_user_posts(user_id)
212         return [
213             PostResponseForUser(
214                 id=post['_id'],
215                 title=post['title'],
216                 content=post['content'],
217                 created_at=post['created_at']
218             )
219             for post in posts
220         ]
221     except HTTPException:
222         raise
223     except Exception as e:
224         raise HTTPException(
225             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
226             detail=f"Internal server error: {str(e)}"
227         )
```

FASTAPI (MONGO)



get(READ):

```
229 @app.get("/posts/", response_model=List[PostResponse])
230 async def get_all_posts():
231     """Get all posts"""
232     try:
233         posts = list(db.posts_collection.find().sort("created_at", -1))
234
235         # Convert ObjectId to string for response
236         for post in posts:
237             post['_id'] = str(post['_id'])
238             post['user_id'] = str(post['user_id'])
239
240         return [
241             PostResponse(
242                 id=post['_id'],
243                 user_id=post['user_id'],
244                 title=post['title'],
245                 content=post['content'],
246                 created_at=post['created_at']
247             )
248             for post in posts
249         ]
250     except Exception as e:
251         raise HTTPException(
252             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
253             detail=f"Internal server error: {str(e)}"
254         )
```

delete(DELETE):

```
256 @app.delete("/users/{user_id}", response_model=dict)
257 async def delete_user(user_id: str):
258     """Delete a user and all their posts"""
259     try:
260         if not ObjectId.is_valid(user_id):
261             raise HTTPException(
262                 status_code=status.HTTP_400_BAD_REQUEST,
263                 detail="Invalid user ID format"
264             )
265
266         # Check if user exists
267         user = db.users_collection.find_one({"_id": ObjectId(user_id)})
268         if not user:
269             raise HTTPException(
270                 status_code=status.HTTP_404_NOT_FOUND,
271                 detail="User not found"
272             )
273
274         success = db.delete_user(user_id)
275         if success:
276             return {"message": "User deleted successfully"}
277         else:
278             raise HTTPException(
279                 status_code=status.HTTP_400_BAD_REQUEST,
280                 detail="Failed to delete user"
281             )
282     except HTTPException:
283         raise
284     except Exception as e:
285         raise HTTPException(
286             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
287             detail=f"Internal server error: {str(e)}"
288         )
```

FASTAPI (MONGO)



delete(DELETE):

```
290 @app.delete("/posts/{post_id}", response_model=dict)
291 async def delete_post(post_id: str):
292     """Delete a specific post"""
293     try:
294         if not ObjectId.is_valid(post_id):
295             raise HTTPException(
296                 status_code=status.HTTP_400_BAD_REQUEST,
297                 detail="Invalid post ID format"
298             )
299
300         result = db.posts_collection.delete_one({"_id": ObjectId(post_id)})
301
302         if result.deleted_count == 0:
303             raise HTTPException(
304                 status_code=status.HTTP_404_NOT_FOUND,
305                 detail="Post not found"
306             )
307
308         return {"message": "Post deleted successfully"}
309     except HTTPException:
310         raise
311     except Exception as e:
312         raise HTTPException(
313             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
314             detail=f"Internal server error: {str(e)}"
315         )
```

put(UPDATE):

```
317 @app.put("/users/{user_id}", response_model=dict)
318 async def update_user(user_id: str, user_update: UserCreate):
319     """Update a user's information"""
320     try:
321         if not ObjectId.is_valid(user_id):
322             raise HTTPException(
323                 status_code=status.HTTP_400_BAD_REQUEST,
324                 detail="Invalid user ID format"
325             )
326
327         # Check if user exists
328         existing_user = db.users_collection.find_one({"_id": ObjectId(user_id)})
329         if not existing_user:
330             raise HTTPException(
331                 status_code=status.HTTP_404_NOT_FOUND,
332                 detail="User not found"
333             )
334
335         # Update user
336         result = db.users_collection.update_one(
337             {"_id": ObjectId(user_id)},
338             {"$set": {
339                 "name": user_update.name,
340                 "email": user_update.email,
341                 "age": user_update.age
342             }}
343         )
344
345         if result.modified_count > 0:
346             return {"message": "User updated successfully"}
347         else:
348             return {"message": "No changes made to user"}
349
350     except HTTPException:
351         raise
352     except Exception as e:
353         raise HTTPException(
354             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
355             detail=f"Internal server error: {str(e)}"
356         )
```

FASTAPI (MONGO)



put(UPDATE):

```
358 @app.put("/posts/{post_id}", response_model=dict)
359 async def update_post(post_id: str, title: str, content: str):
360     """Update a post's title and content"""
361     try:
362         if not ObjectId.is_valid(post_id):
363             raise HTTPException(
364                 status_code=status.HTTP_400_BAD_REQUEST,
365                 detail="Invalid post ID format"
366             )
367
368         # Check if post exists
369         existing_post = db.posts_collection.find_one({"_id": ObjectId(post_id)})
370         if not existing_post:
371             raise HTTPException(
372                 status_code=status.HTTP_404_NOT_FOUND,
373                 detail="Post not found"
374             )
375
376         # Update post
377         result = db.posts_collection.update_one(
378             {"_id": ObjectId(post_id)},
379             {"$set": {
380                 "title": title,
381                 "content": content
382             }}
383         )
384
385         if result.modified_count > 0:
386             return {"message": "Post updated successfully"}
387         else:
388             return {"message": "No changes made to post"}
389
390     except HTTPException:
391         raise
392     except Exception as e:
393         raise HTTPException(
394             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
395             detail=f"Internal server error: {str(e)}"
396         )
```

Entry_point:

```
398 if __name__ == "__main__":
399     import uvicorn
400     uvicorn.run(app, host="0.0.0.0", port=8001)
```