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

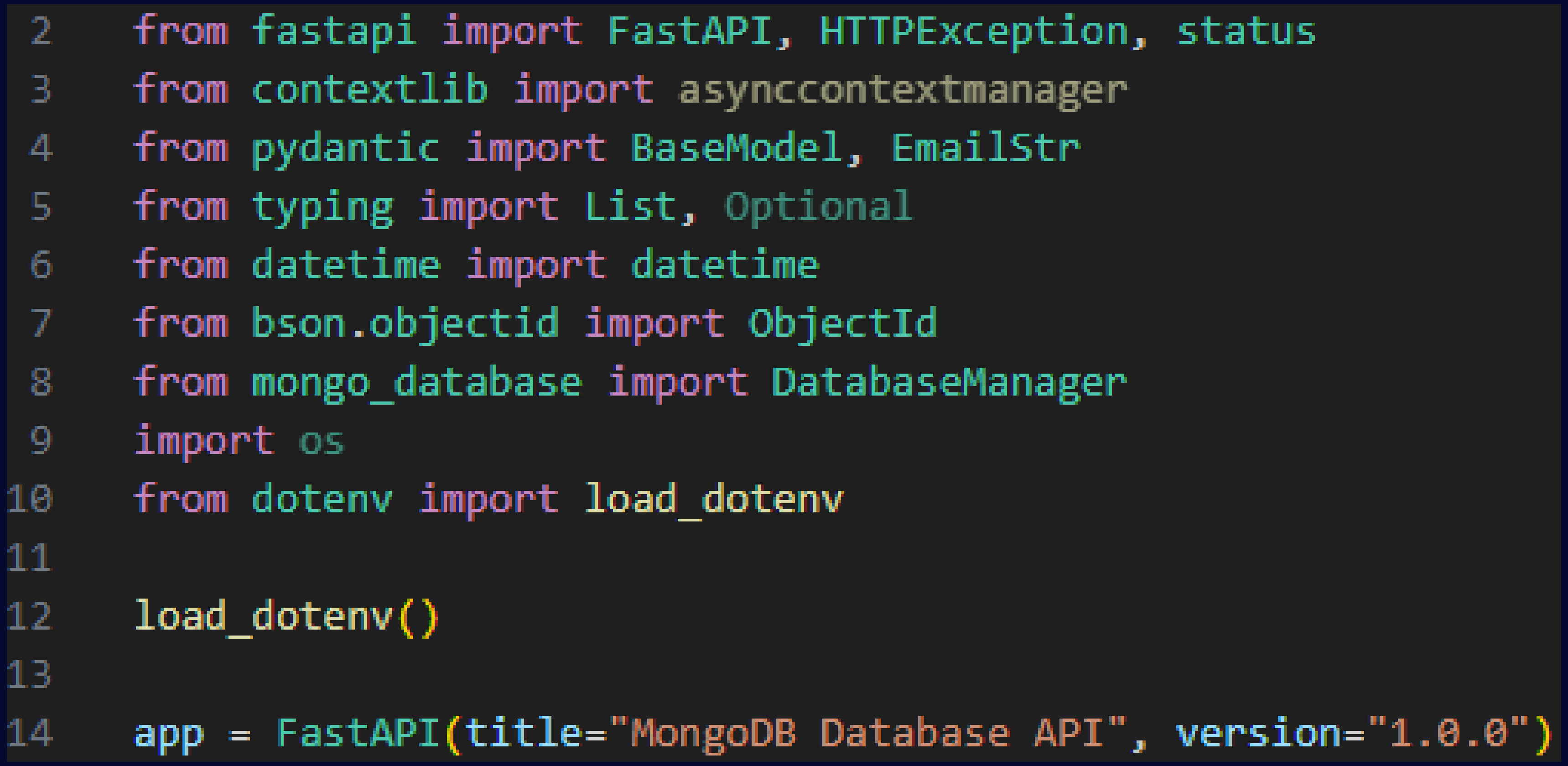
FASTAPI (MONGO)

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)

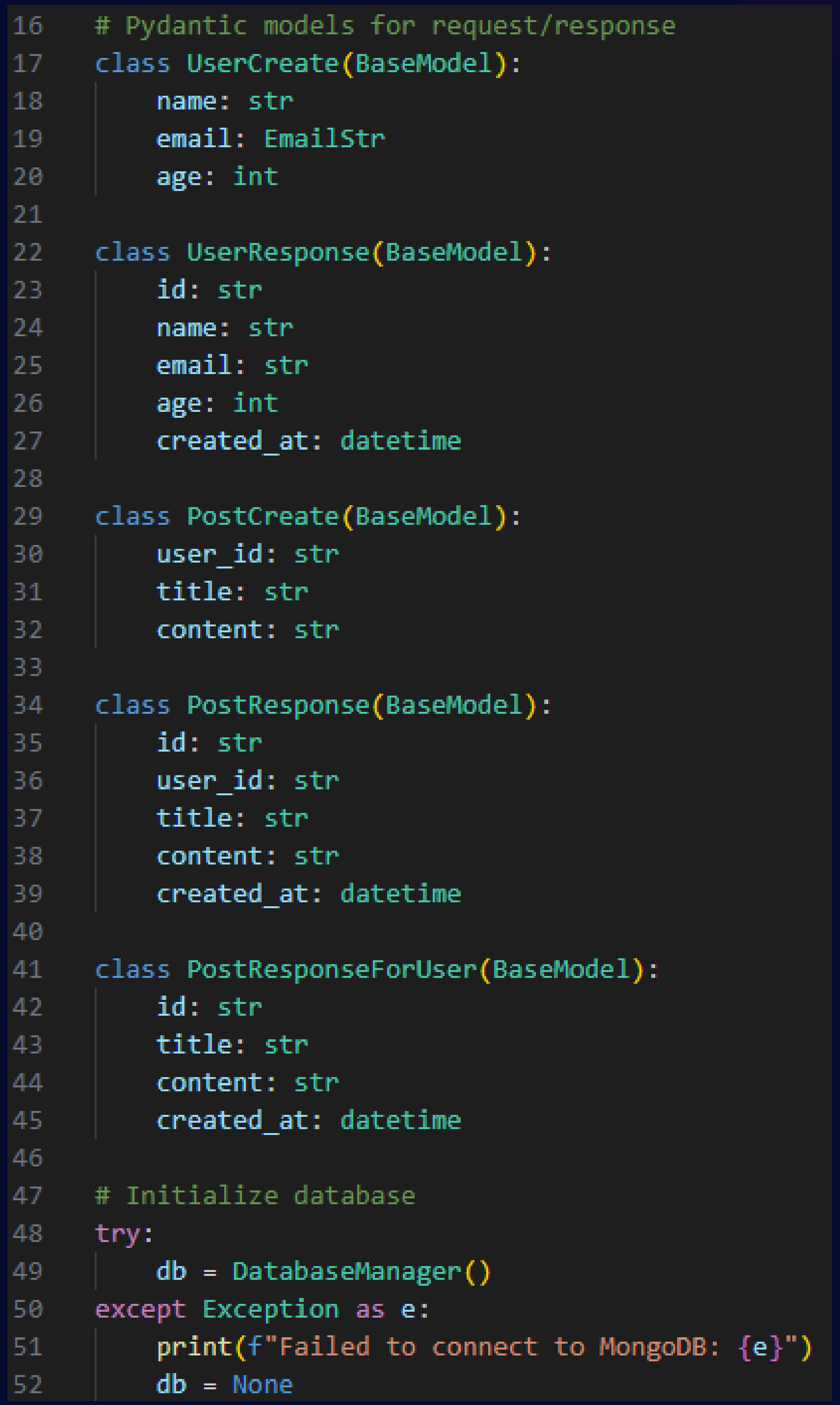
FASTAPI (MONGO)

Imports & Initialization:



```python
from fastapi import FastAPI, HTTPException, status
from contextlib import asynccontextmanager
from pydantic import BaseModel, EmailStr
from typing import List, Optional
from datetime import datetime
from bson.objectid import ObjectId
from mongo_database import DatabaseManager
import os
from dotenv import load_dotenv

load_dotenv()

app = FastAPI(title="MongoDB Database API", version="1.0.0")
```

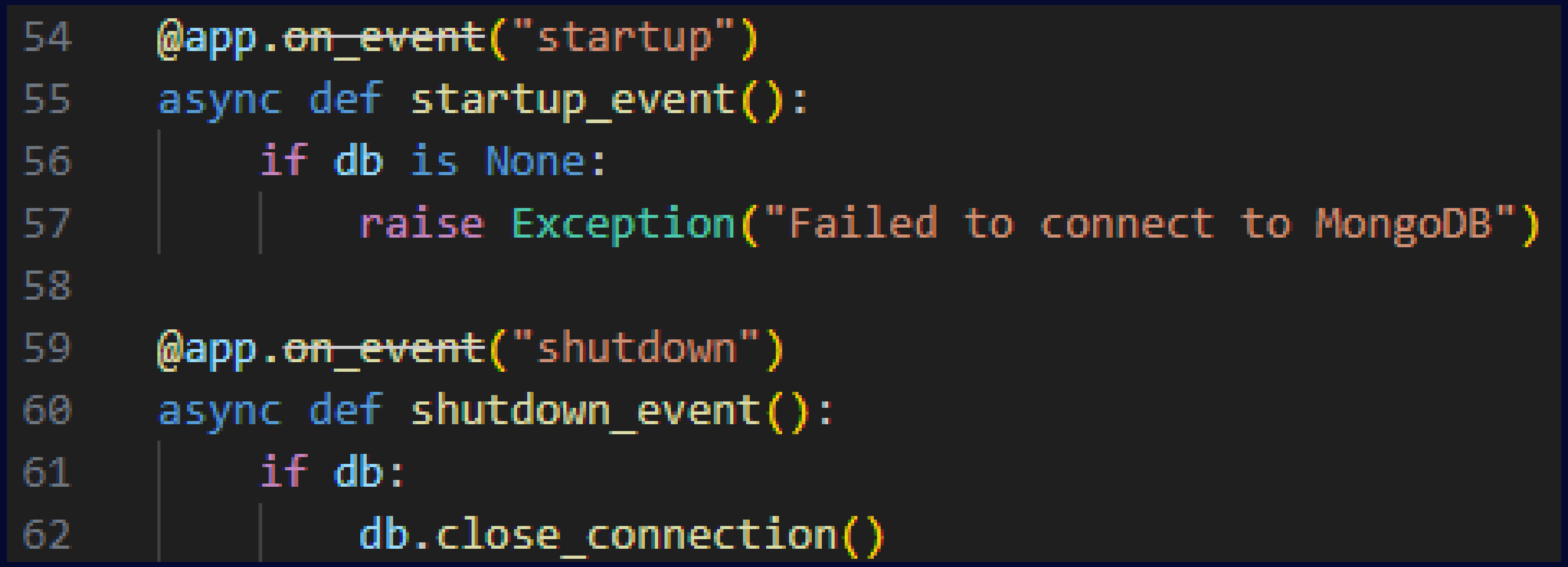
BaseModel:



```python
# Pydantic models for request/response
class UserCreate(BaseModel):
    name: str
    email: EmailStr
    age: int

class UserResponse(BaseModel):
    id: str
    name: str
    email: str
    age: int
    created_at: datetime

class PostCreate(BaseModel):
    user_id: str
    title: str
    content: str

class PostResponse(BaseModel):
    id: str
    user_id: str
    title: str
    content: str
    created_at: datetime

class PostResponseForUser(BaseModel):
    id: str
    title: str
    content: str
    created_at: datetime

# Initialize database
try:
    db = DatabaseManager()
except Exception as e:
    print(f"Failed to connect to MongoDB: {e}")
    db = None
```

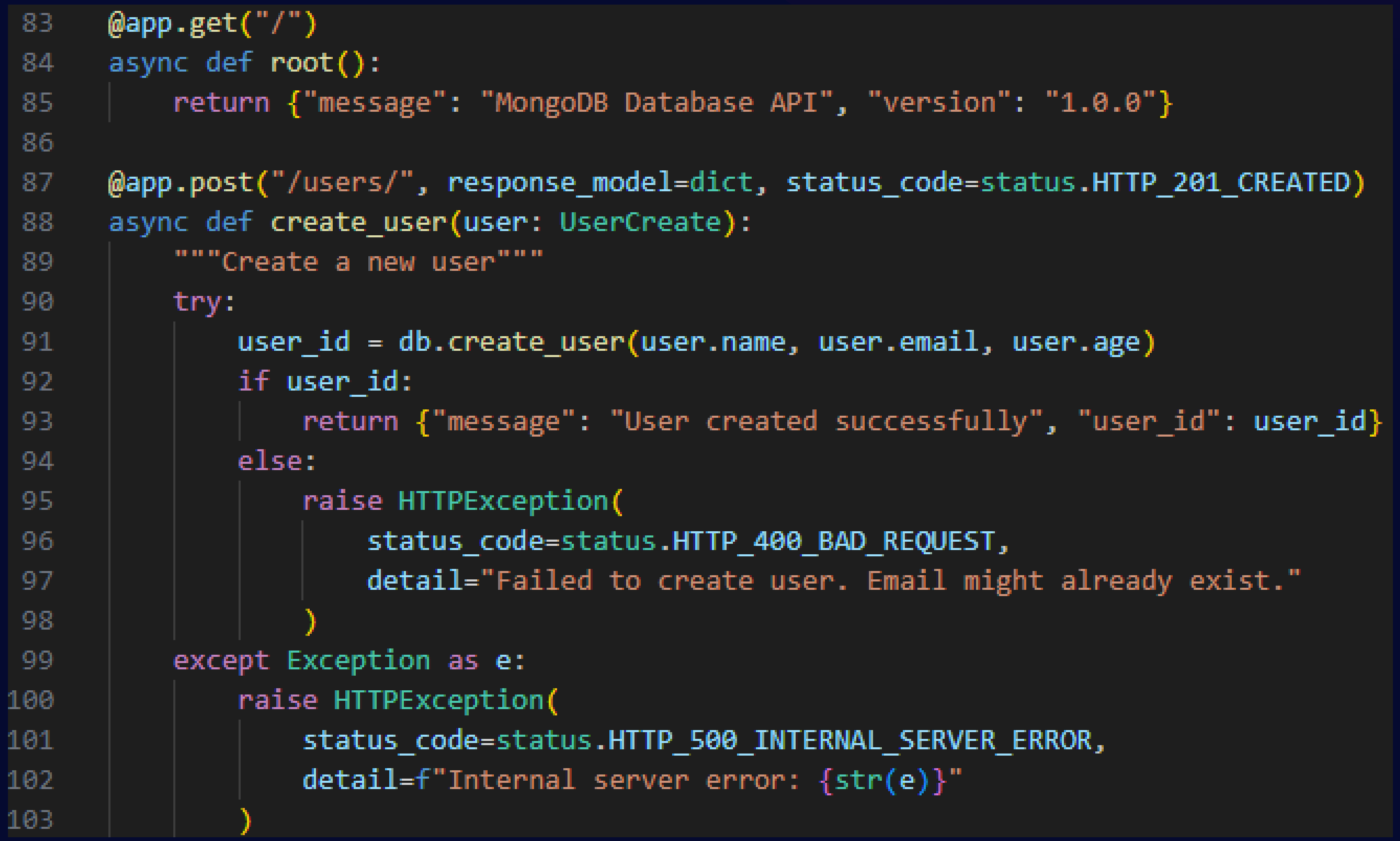
FASTAPI (MONGO)

Event handler:



```python
@app.~~on_event~~("startup")
async def startup_event():
    if db is None:
        raise Exception("Failed to connect to MongoDB")

@app.~~on_event~~("shutdown")
async def shutdown_event():
    if db:
        db.close_connection()
```

post(CREATE):

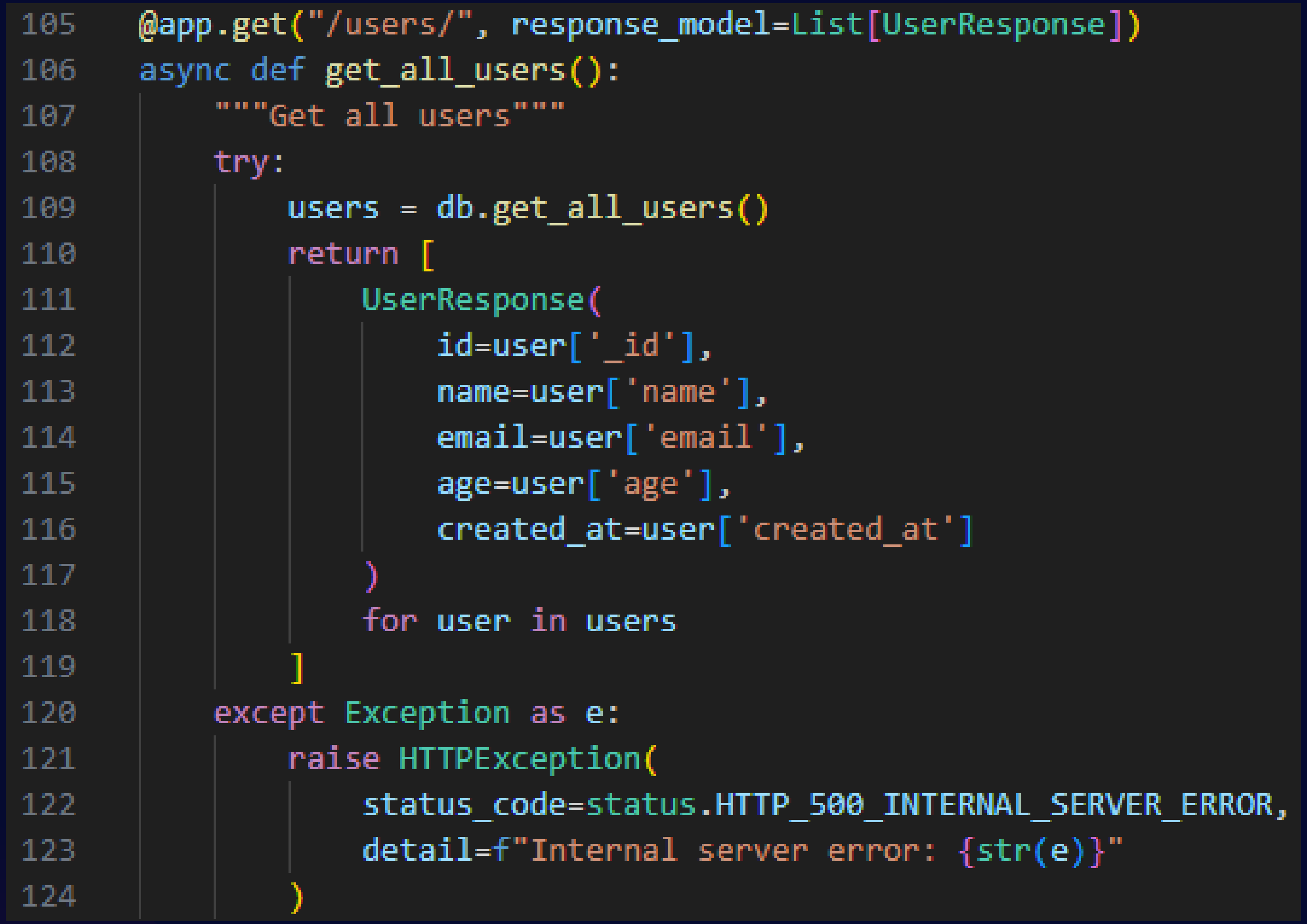


```python
@app.get("/")
async def root():
    return {"message": "MongoDB Database API", "version": "1.0.0"}

@app.post("/users/", response_model=dict, status_code=status.HTTP_201_CREATED)
async def create_user(user: UserCreate):
    """Create a new user"""
    try:
        user_id = db.create_user(user.name, user.email, user.age)
        if user_id:
            return {"message": "User created successfully", "user_id": user_id}
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Failed to create user. Email might already exist."
            )
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

FASTAPI (MONGO)

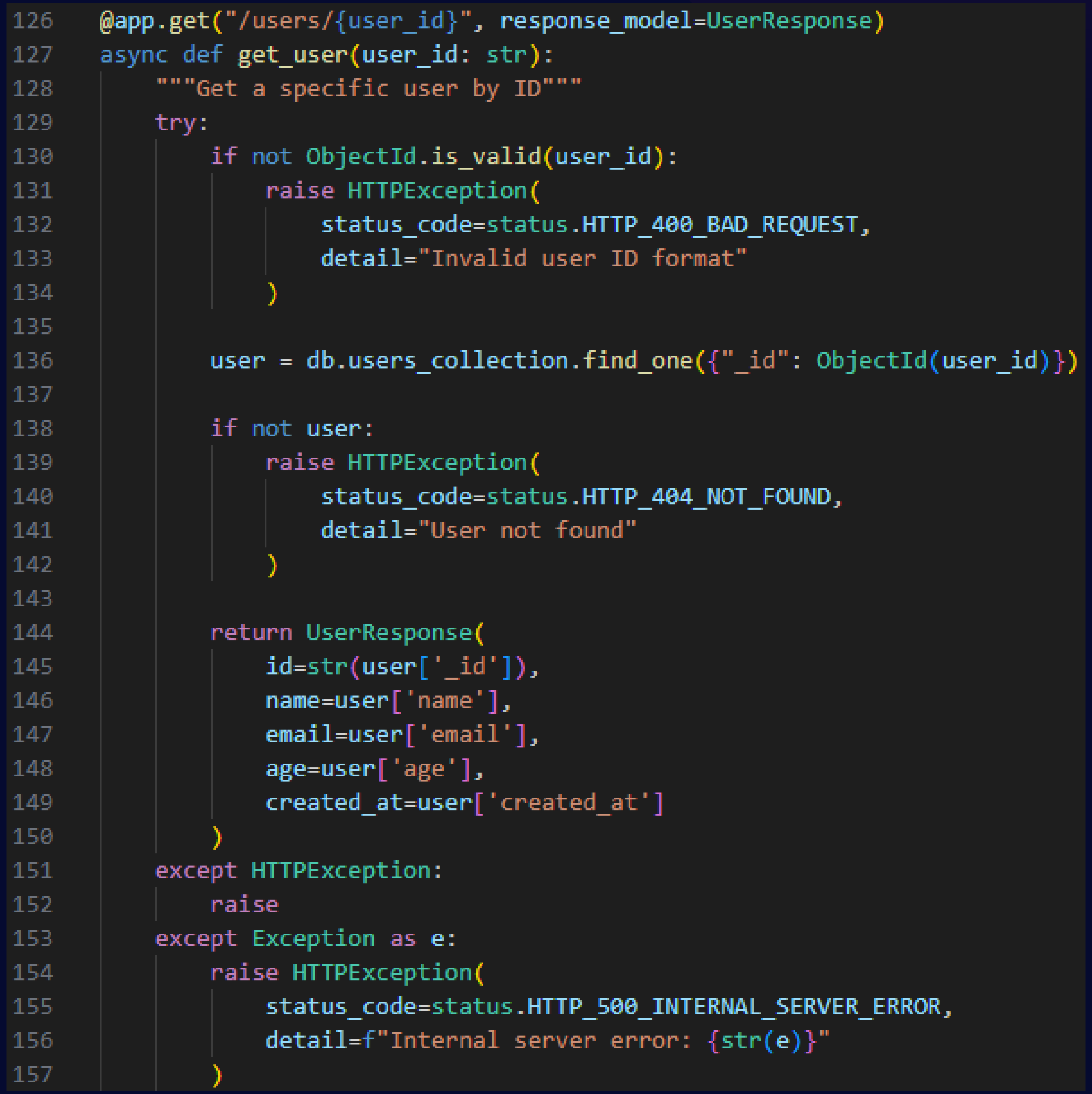
JOMHACK
INSPIRE INNOVATION

get(READ):



```python
@app.get("/users/", response_model=List[UserResponse])
async def get_all_users():
    """Get all users"""
    try:
        users = db.get_all_users()
        return [
            UserResponse(
                id=user['_id'],
                name=user['name'],
                email=user['email'],
                age=user['age'],
                created_at=user['created_at']
            )
            for user in users
        ]
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

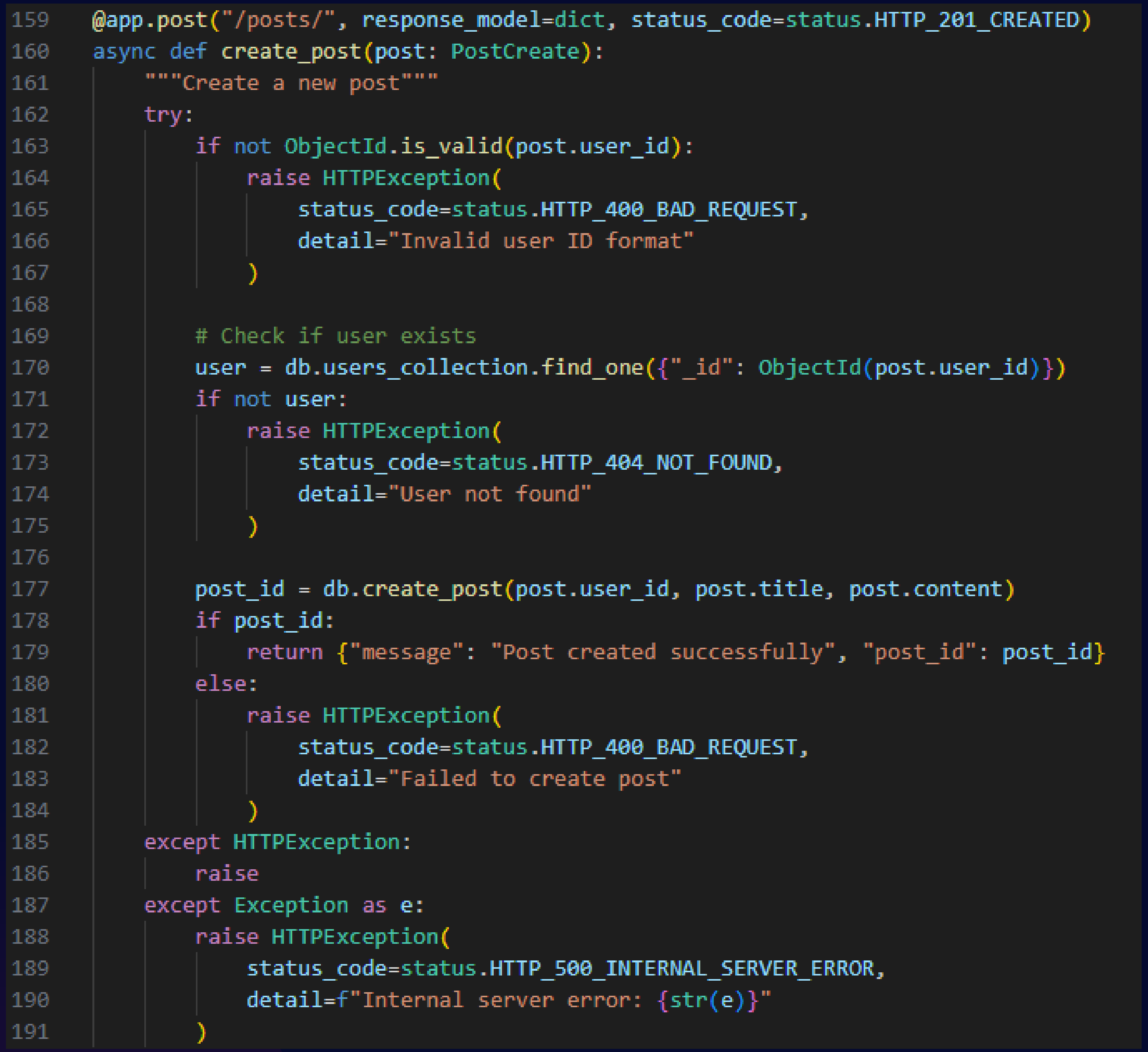
get(READ):



```python
@app.get("/users/{user_id}", response_model=UserResponse)
async def get_user(user_id: str):
    """Get a specific user by ID"""
    try:
        if not ObjectId.is_valid(user_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid user ID format"
            )

        user = db.users_collection.find_one({"_id": ObjectId(user_id)})

        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="User not found"
            )

        return UserResponse(
            id=str(user['_id']),
            name=user['name'],
            email=user['email'],
            age=user['age'],
            created_at=user['created_at']
        )
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

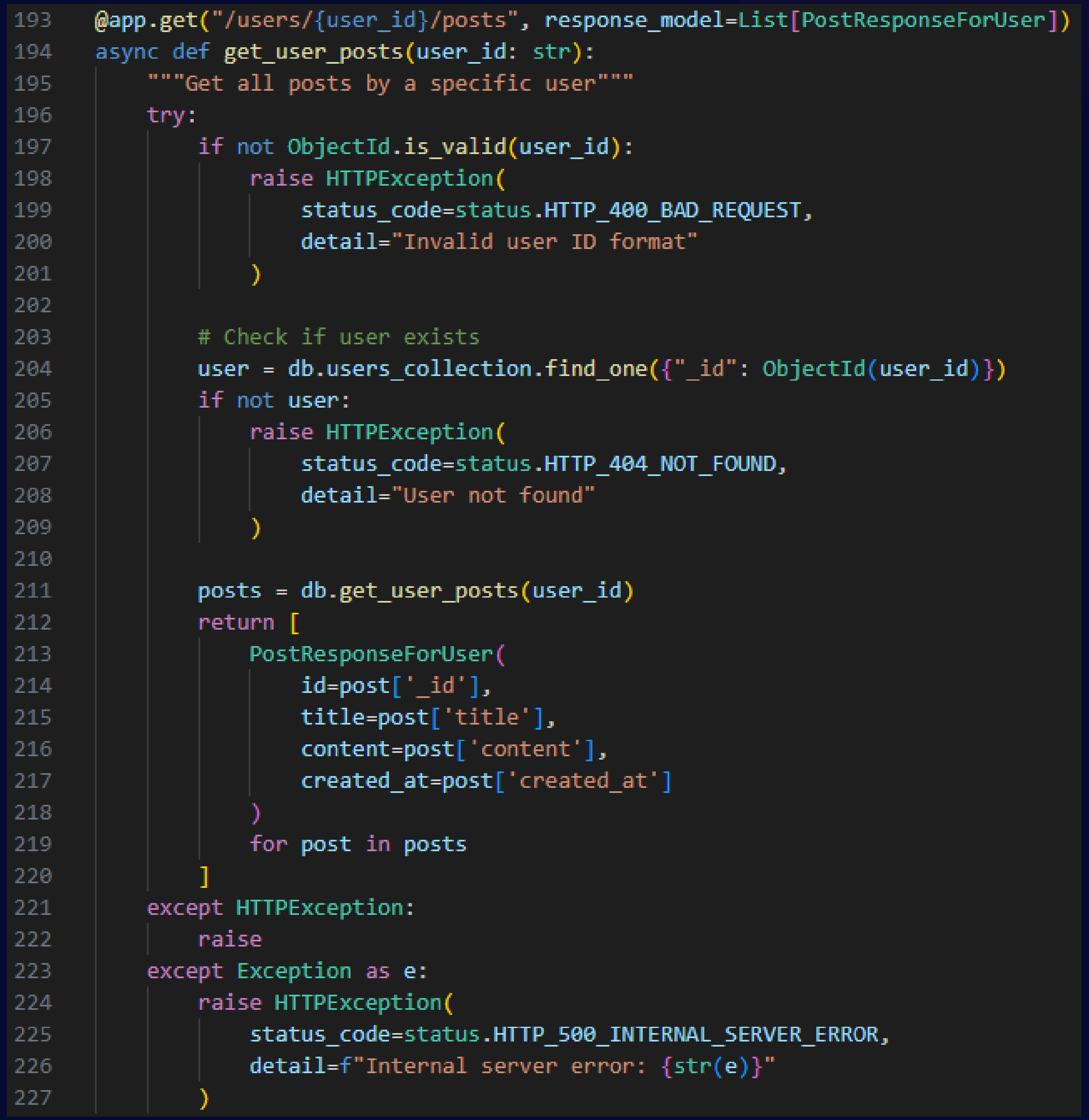
FASTAPI (MONGO)

post(CREATE):



```python
@app.post("/posts/", response_model=dict, status_code=status.HTTP_201_CREATED)
async def create_post(post: PostCreate):
    """Create a new post"""
    try:
        if not ObjectId.is_valid(post.user_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid user ID format"
            )

        # Check if user exists
        user = db.users_collection.find_one({"_id": ObjectId(post.user_id)})
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="User not found"
            )

        post_id = db.create_post(post.user_id, post.title, post.content)
        if post_id:
            return {"message": "Post created successfully", "post_id": post_id}
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Failed to create post"
            )
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

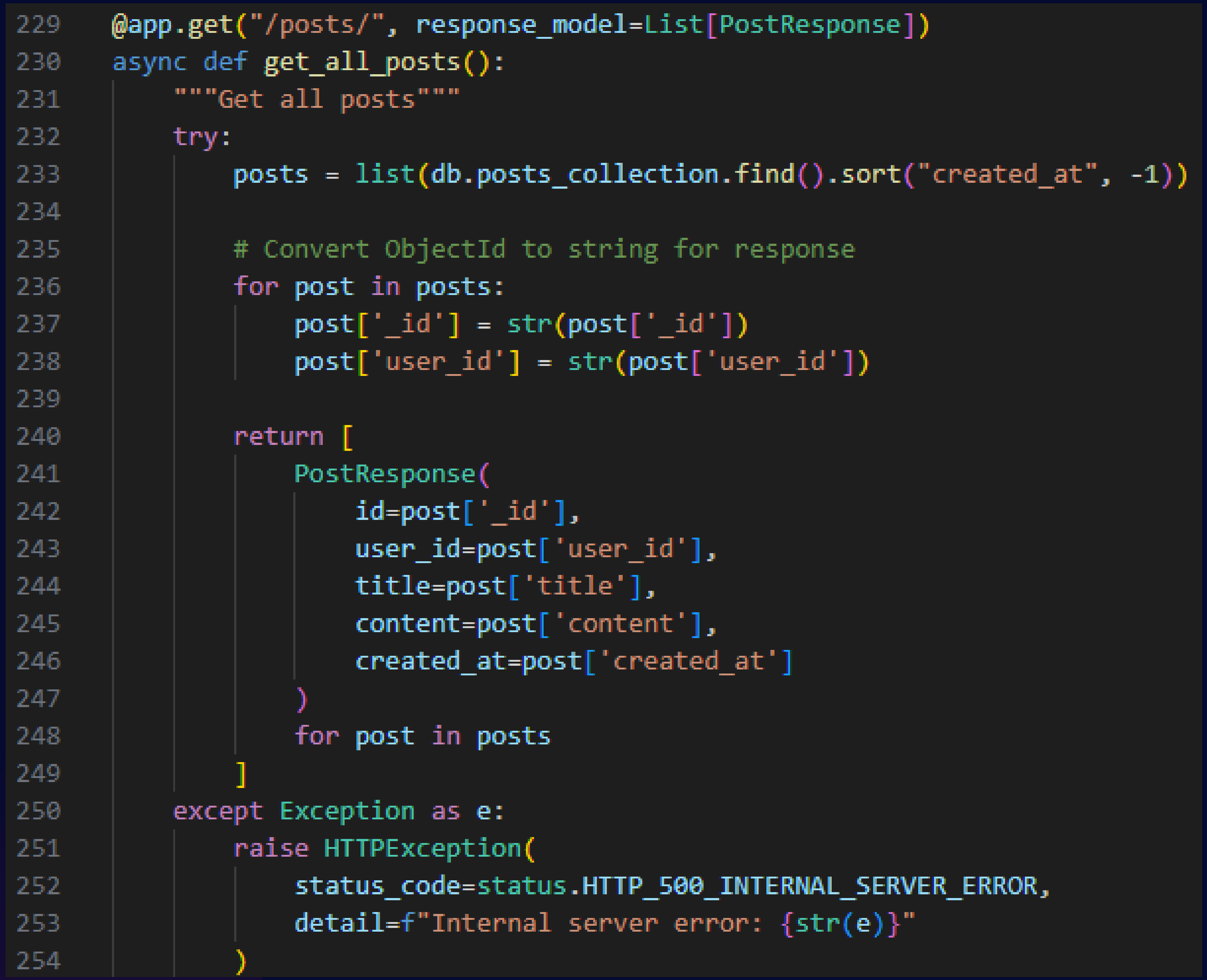
get(READ):



```python
@app.get("/users/{user_id}/posts", response_model=List[PostResponseForUser])
async def get_user_posts(user_id: str):
    """Get all posts by a specific user"""
    try:
        if not ObjectId.is_valid(user_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid user ID format"
            )

        # Check if user exists
        user = db.users_collection.find_one({"_id": ObjectId(user_id)})
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="User not found"
            )

        posts = db.get_user_posts(user_id)
        return [
            PostResponseForUser(
                id=post['_id'],
                title=post['title'],
                content=post['content'],
                created_at=post['created_at']
            )
            for post in posts
        ]
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

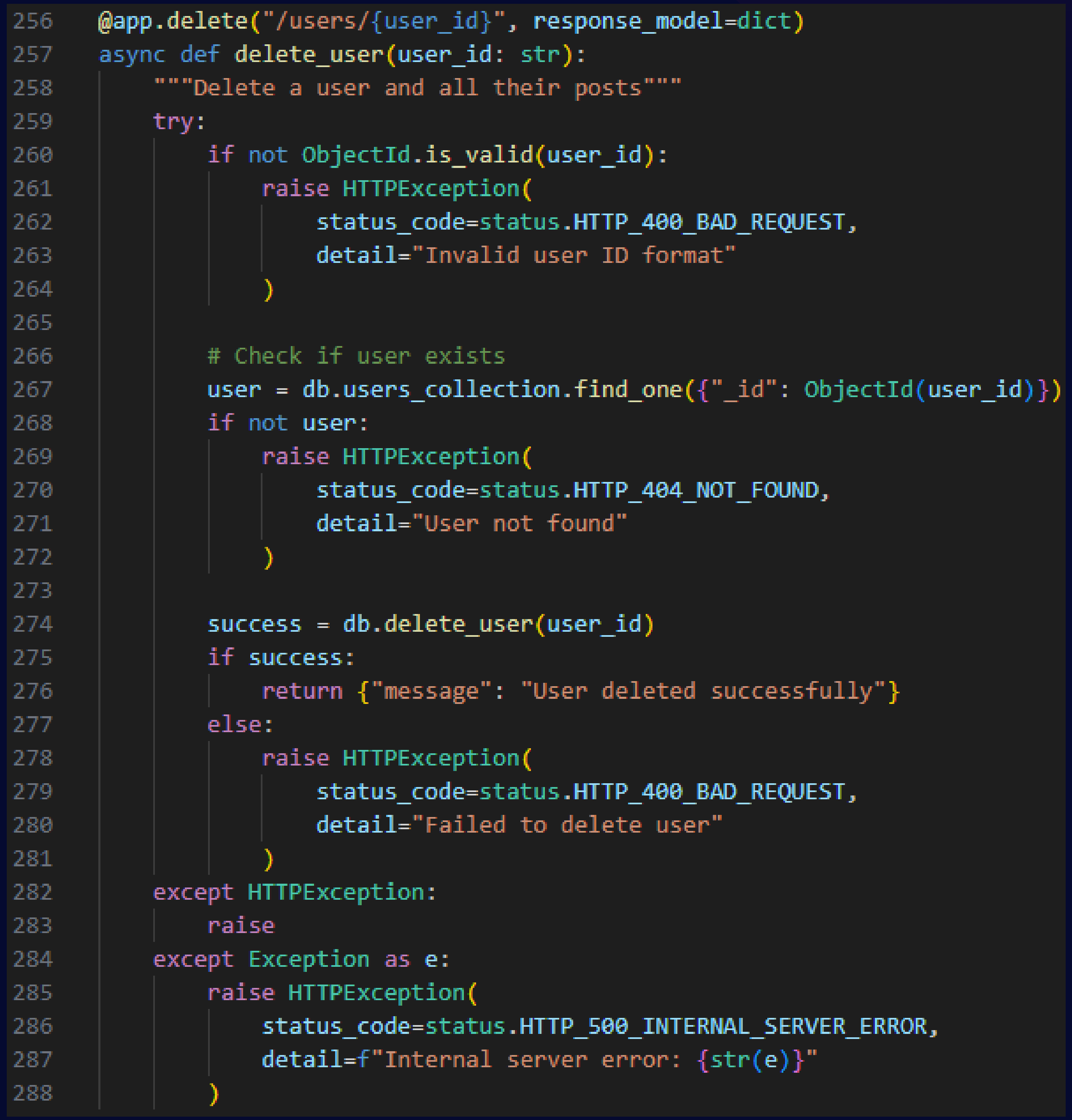
FASTAPI (MONGO)

get(READ):



```python
@app.get("/posts/", response_model=List[PostResponse])
async def get_all_posts():
    """Get all posts"""
    try:
        posts = list(db.posts_collection.find().sort("created_at", -1))

        # Convert ObjectId to string for response
        for post in posts:
            post['_id'] = str(post['_id'])
            post['user_id'] = str(post['user_id'])

        return [
            PostResponse(
                id=post['_id'],
                user_id=post['user_id'],
                title=post['title'],
                content=post['content'],
                created_at=post['created_at']
            )
            for post in posts
        ]
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

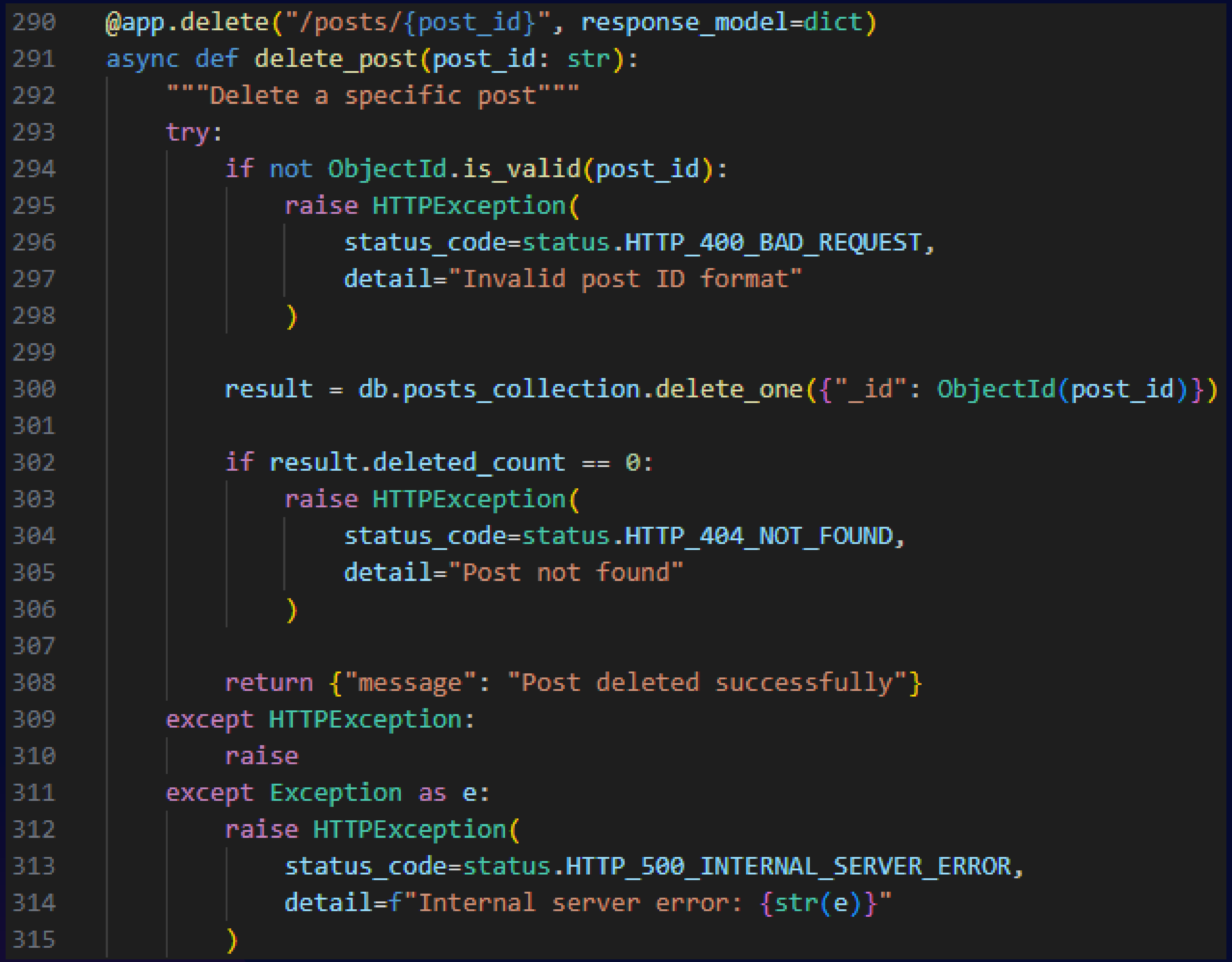
delete(DELETE):



```python
@app.delete("/users/{user_id}", response_model=dict)
async def delete_user(user_id: str):
    """Delete a user and all their posts"""
    try:
        if not ObjectId.is_valid(user_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid user ID format"
            )

        # Check if user exists
        user = db.users_collection.find_one({"_id": ObjectId(user_id)})
        if not user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="User not found"
            )

        success = db.delete_user(user_id)
        if success:
            return {"message": "User deleted successfully"}
        else:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Failed to delete user"
            )
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

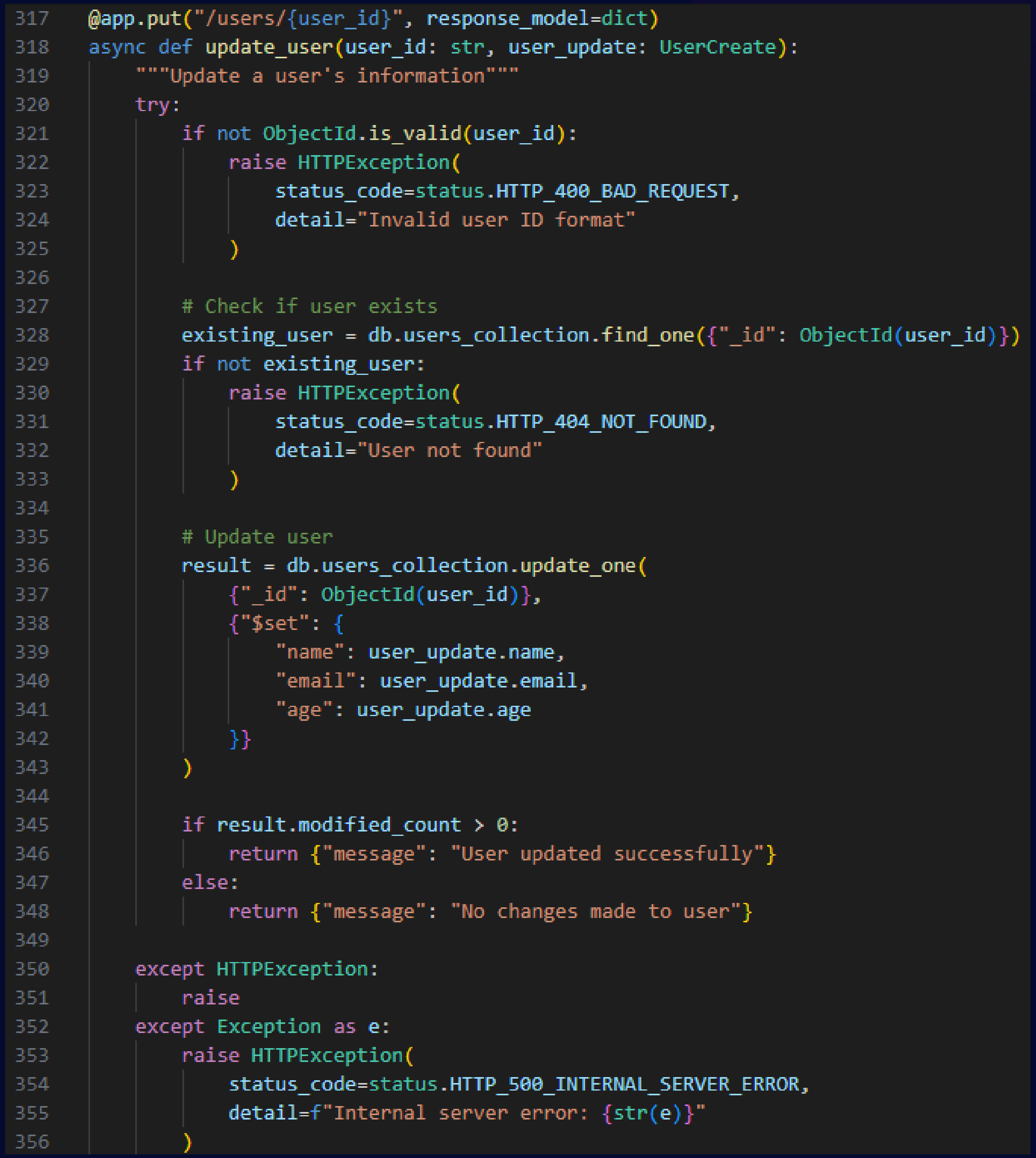
FASTAPI (MONGO)

delete(DELETE):



```python
@app.delete("/posts/{post_id}", response_model=dict)
async def delete_post(post_id: str):
    """Delete a specific post"""
    try:
        if not ObjectId.is_valid(post_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid post ID format"
            )

        result = db.posts_collection.delete_one({"_id": ObjectId(post_id)})

        if result.deleted_count == 0:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Post not found"
            )

        return {"message": "Post deleted successfully"}
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

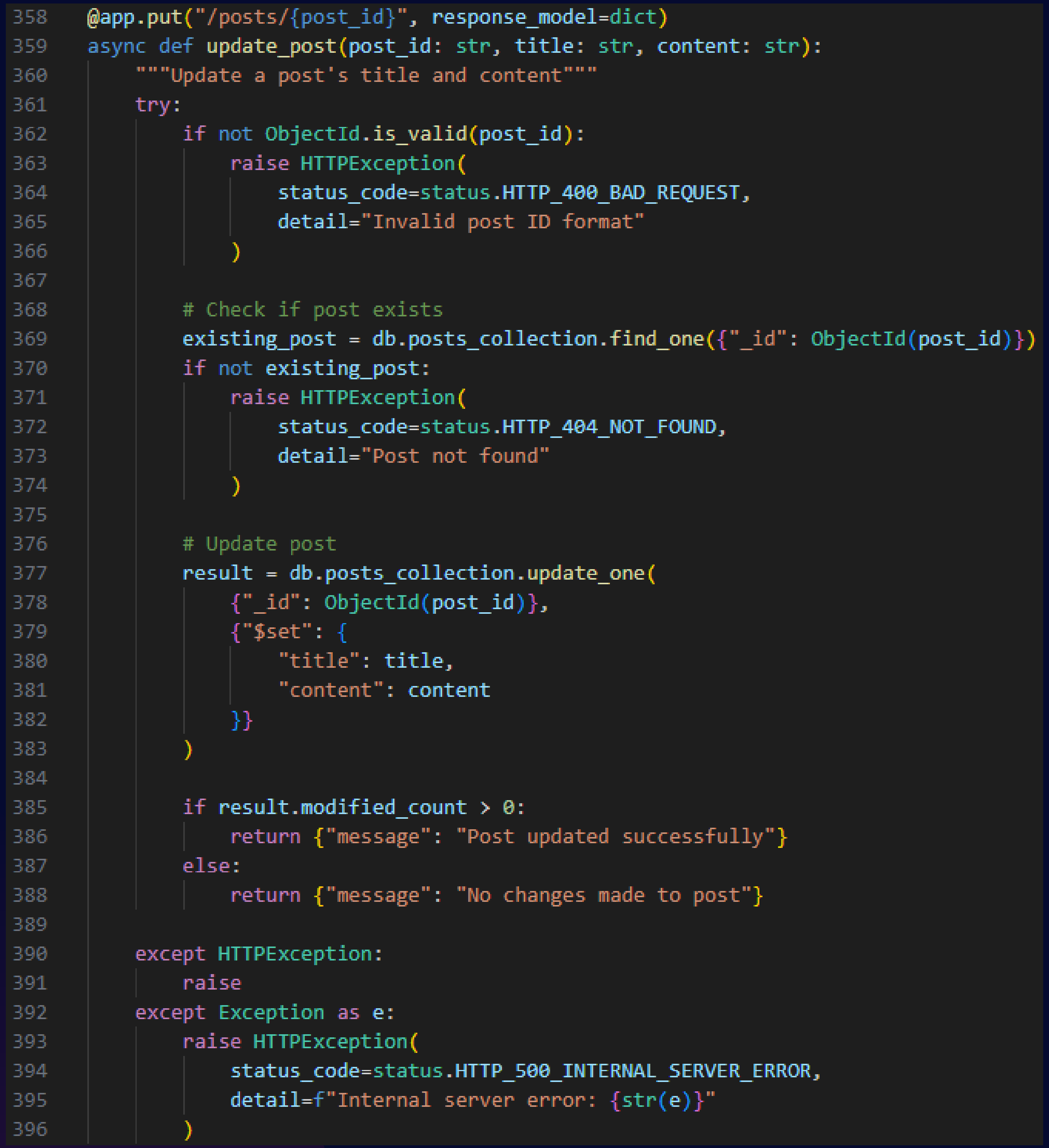
put(UPDATE):



```python
@app.put("/users/{user_id}", response_model=dict)
async def update_user(user_id: str, user_update: UserCreate):
    """Update a user's information"""
    try:
        if not ObjectId.is_valid(user_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid user ID format"
            )

        # Check if user exists
        existing_user = db.users_collection.find_one({"_id": ObjectId(user_id)})
        if not existing_user:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="User not found"
            )

        # Update user
        result = db.users_collection.update_one(
            {"_id": ObjectId(user_id)},
            {"$set": {
                "name": user_update.name,
                "email": user_update.email,
                "age": user_update.age
            }}
        )

        if result.modified_count > 0:
            return {"message": "User updated successfully"}
        else:
            return {"message": "No changes made to user"}

    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

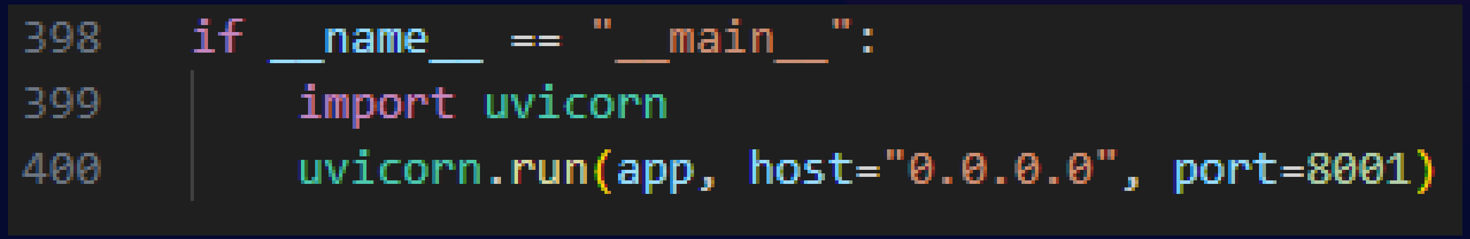
FASTAPI (MONGO)

put(UPDATE):



```python
@app.put("/posts/{post_id}", response_model=dict)
async def update_post(post_id: str, title: str, content: str):
    """Update a post's title and content"""
    try:
        if not ObjectId.is_valid(post_id):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid post ID format"
            )

        # Check if post exists
        existing_post = db.posts_collection.find_one({"_id": ObjectId(post_id)})
        if not existing_post:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail="Post not found"
            )

        # Update post
        result = db.posts_collection.update_one(
            {"_id": ObjectId(post_id)},
            {"$set": {
                "title": title,
                "content": content
            }}
        )

        if result.modified_count > 0:
            return {"message": "Post updated successfully"}
        else:
            return {"message": "No changes made to post"}

    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Internal server error: {str(e)}"
        )
```

Entry point:



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