



A large, semi-transparent dark red circle is positioned in the center of the slide, partially overlapping the text area.

PYTHON BOOTCAMP

www.jomhack.com

FASTAPI (SQLITE)

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 (SQLITE)

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 (SQLITE)

Imports & Initialization:

```
2  from fastapi import FastAPI, HTTPException, status
3  from pydantic import BaseModel, EmailStr
4  from typing import List, Optional
5  from datetime import datetime
6  import sqlite3
7  from sql_database import DatabaseManager
8
9  app = FastAPI(title="SQLite Database API", version="1.0.0")
```

Basemodel:

```
11 # Pydantic models for request/response
12 class UserCreate(BaseModel):
13     name: str
14     email: EmailStr
15     age: int
16
17 class UserResponse(BaseModel):
18     id: int
19     name: str
20     email: str
21     age: int
22     created_at: str
23
24 class PostCreate(BaseModel):
25     user_id: int
26     title: str
27     content: str
28
29 class PostResponse(BaseModel):
30     id: int
31     user_id: int
32     title: str
33     content: str
34     created_at: str
35
36 class PostResponseForUser(BaseModel):
37     id: int
38     title: str
39     content: str
40     created_at: str
41
42 # Initialize database
43 db = DatabaseManager()
```

FASTAPI (SQLITE)

post(CREATE):

```

45 @app.get("/")
46 async def root():
47     return {"message": "SQLite Database API", "version": "1.0.0"}
48
49 @app.post("/users/", response_model=dict, status_code=status.HTTP_201_CREATED)
50 async def create_user(user: UserCreate):
51     """Create a new user"""
52     try:
53         user_id = db.create_user(user.name, user.email, user.age)
54         if user_id:
55             return {"message": "User created successfully", "user_id": user_id}
56         else:
57             raise HTTPException(
58                 status_code=status.HTTP_400_BAD_REQUEST,
59                 detail="Failed to create user. Email might already exist."
60             )
61     except Exception as e:
62         raise HTTPException(
63             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
64             detail=f"Internal server error: {str(e)}"
65         )

```

get(READ):

```

67 @app.get("/users/", response_model=List[UserResponse])
68 async def get_all_users():
69     """Get all users"""
70     try:
71         users = db.get_all_users()
72         return [
73             UserResponse(
74                 id=user[0],
75                 name=user[1],
76                 email=user[2],
77                 age=user[3],
78                 created_at=user[4]
79             )
80             for user in users
81         ]
82     except Exception as e:
83         raise HTTPException(
84             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
85             detail=f"Internal server error: {str(e)}"
86         )

```

FASTAPI (SQLITE)

get(READ):

```

88 @app.get("/users/{user_id}", response_model=UserResponse)
89 async def get_user(user_id: int):
90     """Get a specific user by ID"""
91     try:
92         with sqlite3.connect(db.db_name) as conn:
93             cursor = conn.cursor()
94             cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
95             user = cursor.fetchone()
96
97         if not user:
98             raise HTTPException(
99                 status_code=status.HTTP_404_NOT_FOUND,
100                detail="User not found"
101            )
102
103         return UserResponse(
104             id=user[0],
105             name=user[1],
106             email=user[2],
107             age=user[3],
108             created_at=user[4]
109         )
110     except HTTPException:
111         raise
112     except Exception as e:
113         raise HTTPException(
114             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
115             detail=f"Internal server error: {str(e)}"
116         )

```

post(CREATE):

```

118 @app.post("/", response_model=dict, status_code=status.HTTP_201_CREATED)
119 async def create_post(post: PostCreate):
120     """Create a new post"""
121     try:
122         # Check if user exists
123         with sqlite3.connect(db.db_name) as conn:
124             cursor = conn.cursor()
125             cursor.execute("SELECT id FROM users WHERE id = ?", (post.user_id,))
126             if not cursor.fetchone():
127                 raise HTTPException(
128                     status_code=status.HTTP_404_NOT_FOUND,
129                     detail="User not found"
130                 )
131
132         post_id = db.create_post(post.user_id, post.title, post.content)
133         if post_id:
134             return {"message": "Post created successfully", "post_id": post_id}
135         else:
136             raise HTTPException(
137                 status_code=status.HTTP_400_BAD_REQUEST,
138                 detail="Failed to create post"
139             )
140     except HTTPException:
141         raise
142     except Exception as e:
143         raise HTTPException(
144             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
145             detail=f"Internal server error: {str(e)}"
146         )

```

FASTAPI (SQLITE)

get(READ):

```

148 @app.get("/users/{user_id}/posts", response_model=List[PostResponseForUser])
149 async def get_user_posts(user_id: int):
150     """Get all posts by a specific user"""
151     try:
152         # Check if user exists
153         with sqlite3.connect(db.db_name) as conn:
154             cursor = conn.cursor()
155             cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
156             if not cursor.fetchone():
157                 raise HTTPException(
158                     status_code=status.HTTP_404_NOT_FOUND,
159                     detail="User not found"
160                 )
161
162         posts = db.get_user_posts(user_id)
163         return [
164             PostResponseForUser(
165                 id=post[0],
166                 title=post[1],
167                 content=post[2],
168                 created_at=post[3]
169             )
170             for post in posts
171         ]
172     except HTTPException:
173         raise
174     except Exception as e:
175         raise HTTPException(
176             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
177             detail=f"Internal server error: {str(e)}"
178         )

```

get(READ):

```

180 @app.get("/posts/", response_model=List[PostResponse])
181 async def get_all_posts():
182     """Get all posts"""
183     try:
184         with sqlite3.connect(db.db_name) as conn:
185             cursor = conn.cursor()
186             cursor.execute("SELECT * FROM posts ORDER BY created_at DESC")
187             posts = cursor.fetchall()
188
189         return [
190             PostResponse(
191                 id=post[0],
192                 user_id=post[1],
193                 title=post[2],
194                 content=post[3],
195                 created_at=post[4]
196             )
197             for post in posts
198         ]
199     except Exception as e:
200         raise HTTPException(
201             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
202             detail=f"Internal server error: {str(e)}"
203         )

```

FASTAPI (SQLITE)

delete(DELETE):

```

205 @app.delete("/users/{user_id}", response_model=dict)
206 async def delete_user(user_id: int):
207     """Delete a user and all their posts"""
208     try:
209         # Check if user exists
210         with sqlite3.connect(db.db_name) as conn:
211             cursor = conn.cursor()
212             cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
213             if not cursor.fetchone():
214                 raise HTTPException(
215                     status_code=status.HTTP_404_NOT_FOUND,
216                     detail="User not found"
217                 )
218
219         success = db.delete_user(user_id)
220         if success:
221             return {"message": "User deleted successfully"}
222         else:
223             raise HTTPException(
224                 status_code=status.HTTP_400_BAD_REQUEST,
225                 detail="Failed to delete user"
226             )
227     except HTTPException:
228         raise
229     except Exception as e:
230         raise HTTPException(
231             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
232             detail=f"Internal server error: {str(e)}"
233         )

```

delete(DELETE):

```

235 @app.delete("/posts/{post_id}", response_model=dict)
236 async def delete_post(post_id: int):
237     """Delete a specific post"""
238     try:
239         with sqlite3.connect(db.db_name) as conn:
240             cursor = conn.cursor()
241             cursor.execute("DELETE FROM posts WHERE id = ?", (post_id,))
242
243             if cursor.rowcount == 0:
244                 raise HTTPException(
245                     status_code=status.HTTP_404_NOT_FOUND,
246                     detail="Post not found"
247                 )
248
249         return {"message": "Post deleted successfully"}
250     except HTTPException:
251         raise
252     except Exception as e:
253         raise HTTPException(
254             status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
255             detail=f"Internal server error: {str(e)}"
256         )
257
258     if __name__ == "__main__":
259         import uvicorn
260         uvicorn.run(app, host="0.0.0.0", port=8000)

```