FastAPI

Intro

Intro

Welcome, thanks for being here

Intro

Welcome, thanks for being here

FastAPI

Intro

Welcome, thanks for being here

FastAPI

Easy to use, fast, lightweight

Intro

Welcome, thanks for being here

FastAPI

Easy to use, fast, lightweight

Swagger doc feature is awesome

Intro

Welcome, thanks for being here

FastAPI

Easy to use, fast, lightweight

Swagger doc feature is awesome

You will learn everything you need to know about FastAPI

Intro

Welcome, thanks for being here

FastAPI

Easy to use, fast, lightweight

Swagger doc feature is awesome

You will learn everything you need to know about FastAPI

Implement a real world API project

Intro

Welcome, thanks for being here

FastAPI

Easy to use, fast, lightweight

Swagger doc feature is awesome

You will learn everything you need to know about FastAPI

Implement a real world API project

Add 2 different clients that connect to API

FastAPI

Overview

Getting started overview

Installation

Features

First code

FastAPI

Features

Features

Automatic documentation

Features

Automatic documentation

- Swagger
- ReDoc

Features

Automatic documentation

- Swagger
- ReDoc

Standard python3

Features

Automatic documentation

- Swagger
- ReDoc

Standard python3

Security and authentication

Features

Automatic documentation

- Swagger
- ReDoc

Standard python3

Security and authentication

Dependency injection

Features

Automatic documentation

- Swagger
- ReDoc

Standard python3

Security and authentication

Dependency injection

Testing - 100% coverage

FastAPI

GET method overview

GET method overview

Path parameters

Predefined values

Query parameters

FastAPI

Path parameters

Path parameters

Path parameters

```
@app.get('/blog/{id}')
def index(id):
    return {"message": f"Blog with id {id}"}
```

Path parameters

Path parameters

Type validation

```
@app.get('/blog/{id}')
def index(id: int):
    return {"message": f"Blog with id {id}"}
```

Path parameters

Path parameters

Type validation

```
@app.get('/blog/{id}')
def index(id: int):
    return {"message": f"Blog with id {id}"}
```

Pydantic

Path parameters

Path parameters

Type validation

```
@app.get('/blog/{id}')
def index(id: int):
    return {"message": f"Blog with id {id}"}
```

Pydantic

Order is important

FastAPI

Predefined path

Predefined path

Predefined values with Enum

Predefined path

Predefined values with Enum

```
class BlogType(str, Enum):
    short = 'short'
    story = 'story'
    howto = 'howto'
```

Predefined path

Predefined values with Enum

```
class BlogType(str, Enum):
    short = 'short'
    story = 'story'
    howto = 'howto'
```

```
@app.get('/blog/type/{type}')
def blog_type(type: BlogType):
    return {"message": f"Blog type: {type}"}
```

FastAPI

Query parameters

Query parameters

```
http://localhost:8000/blogs/all?page=2&page_size=10
```

Query parameters: *page* and *page_size*

Query parameters

```
http://localhost:8000/blogs/all?page=2&page_size=10
```

Query parameters: *page* and *page_size*

Any function parameters not part of the path

Query parameters

```
http://localhost:8000/blogs/all?page=2&page_size=10
```

Query parameters: *page* and *page_size*

Any function parameters not part of the path

```
@app.get('/blog/all')
def get_blogs(page, page_size):
    return {'message': f'All {page_size} blogs on page {page}'}
```

Query parameters

Default values

Query parameters

Default values

```
@app.get('/blog/all')
def get_blogs(page = 1, page_size = 10):
   return {'message': f'All {page_size} blogs on page {page}'}
```

Query parameters

Default values

```
@app.get('/blog/all')
def get_blogs(page = 1, page_size = 10):
    return {'message': f'All {page_size} blogs on page {page}'}
```

Optional parameters

Query parameters

Default values

```
@app.get('/blog/all')
def get_blogs(page = 1, page_size = 10):
    return {'message': f'All {page_size} blogs on page {page}'}
```

Optional parameters

```
@app.get('/blog/all')
def get_blogs(page = 1, page_size: Optional[int] = None):
    return {'message': f'All {page_size} blogs on page {page}'}
```

Query parameters

Default values

```
@app.get('/blog/all')
def get_blogs(page = 1, page_size = 10):
    return {'message': f'All {page_size} blogs on page {page}'}
```

Optional parameters

```
@app.get('/blog/all')
def get_blogs(page = 1, page_size: Optional[int] = None):
    return {'message': f'All {page_size} blogs on page {page}'}
```

Query & path parameters

FastAPI

Operation description overview

Operation description overview

Status code

Tags

Summary and description

Response description

FastAPI

Status code

Status code

Indicates the outcome of an operation

Status code

Indicates the outcome of an operation

Change the status code

```
@app.get('/blog/{id}', status_code=404)
def get_blog(id: int):
   if id > 5:
       return {'error': f'Blog {id} not found'}
   else :
       return {'message': f'Blog with id {id}'}
```

Status code

Indicates the outcome of an operation

Change the status code

```
@app.get('/blog/{id}', status_code=status.HTTP_404_NOT_FOUND)
def get_blog(id: int):
   if id > 5:
       return {'error': f'Blog {id} not found'}
   else :
       return {'message': f'Blog with id {id}'}
```

Status code

Change the status code on the response

```
@app.get('/blog/{id}', status_code=status.HTTP_404_NOT_FOUND)
def get_blog(id: int, response: Response):
   if id > 5:
       response.status_code = status.HTTP_404_NOT_FOUND
       return {'error': f'Blog {id} not found'}
   else :
       response.status_code = status.HTTP_200_OK
       return {'message': f'Blog with id {id}'}
```

FastAPI

Tags

Tags

Categorize operations

Tags

Categorize operations

Multiple categories

Tags

Categorize operations

Multiple categories

```
@app.get('/blog/{id}/comments/{comment_id}', tags=['blog', 'comment'])
```

FastAPI

Summary and description

Summary and description

Info regarding operation

Summary and description

Info regarding operation

```
@app.get(
   '/blog/all',
   tags=['blog'],
   summary="Retrieve all blogs",
   description="This api call simulates fetching all blogs"
   )
```

Summary and description

Info regarding operation

```
@app.get(
    '/blog/all',
    tags=['blog'],
    summary="Retrieve all blogs",
    description="This api call simulates fetching all blogs"
    )
```

Description is also taken from function docstring

FastAPI

Routers overview

Routers overview

Routers

Refactoring the app

Adding a second router

FastAPI

Routers

Routers

Separate operations into multiple files

Routers

Separate operations into multiple files

Share a prefix btw multiple operations

Routers

Separate operations into multiple files

Share a prefix btw multiple operations

Share tags

Routers

Separate operations into multiple files

Share a prefix btw multiple operations

Share tags

```
from fastapi import APIRouter
router = APIRouter(prefix='/blog', tags=['blog'])


@router.get('/')
```

Routers

Separate operations into multiple files

Share a prefix btw multiple operations

Share tags

```
from fastapi import APIRouter
router = APIRouter(prefix='/blog', tags=['blog'])

@router.get('/')
```

```
from routers import blog
app = FastAPI()
app.include_router(blog.router)
```

FastAPI

Parameters overview

Parameters overview

Request body

Path and Query parameters

Parameter metadata

Validators

Multiple values

Number validators

Complex subtypes

FastAPI

Request body

Request body

POST method

Request body

POST method

Pydantic *BaseModel*

Request body

POST method

Pydantic *BaseModel*

```
class BlogModel(BaseModel):
    title: str
    content: str
    published: Optional[bool]
```

Request body

POST method

Pydantic *BaseModel*

```
class BlogModel(BaseModel):
    title: str
    content: str
    published: Optional[bool]
```

FastAPI will convert the data

Request body

POST method

Pydantic *BaseModel*

```
class BlogModel(BaseModel):
    title: str
    content: str
    published: Optional[bool]
```

FastAPI will convert the data

```
@router.post('/new')
def create_blog(blog: BlogModel):
    return blog
```

Request body

Read request body as JSON

Request body

Read request body as JSON

Data validation

Request body

Read request body as JSON

Data validation

Data conversion

Request body

Read request body as JSON

Data validation

Data conversion

JSON schema

FastAPI

Request body with parameters

Path and query parameters

Combine all 3 types of data

Path and query parameters

Combine all 3 types of data

```python
@router.post('/new/{id}')
def create_blog(blog: BlogModel, id: int, version: int = 1):
```

FastAPI

Parameter metadata

Parameter metadata

Information displayed in docs

Parameter metadata

Information displayed in docs

Using the Query, Path and Body imports

Parameter metadata

Information displayed in docs

Using the Query, Path and Body imports

Set default value

Parameter metadata

Information displayed in docs

Using the Query, Path and Body imports

Set default value

```
comment_id: int = Query(None)
```

Parameter metadata

Add title and description

```
comment_id: int = Query(None,
        title='Id of the comment',
        description='Some description for comment_id',
```

Parameter metadata

Add title and description

```
comment_id: int = Query(None,
       title='Id of the comment',
       description='Some description for comment_id',
```

Add alias

```
   comment_id: int = Query(None,
       alias='commentId',
```

Parameter metadata

Add title and description

```
comment_id: int = Query(None,
       title='Id of the comment',
       description='Some description for comment_id',
```

Add alias

```
   comment_id: int = Query(None,
       alias='commentId',
```

Add deprecation

```
comment_id: int = Query(None,
       deprecated=True
```

FastAPI

Validators

Validators

Validate data passed to parameters

Validators

Validate data passed to parameters

Provide a default value

```
content: str = Body('hi how are you')
```

Validators

Validate data passed to parameters

Provide a default value

```
content: str = Body('hi how are you')
```

Require a value (non-optional parameters)

```
content: str = Body(...)
```

Validators

Validate data passed to parameters

Provide a default value

```
content: str = Body('hi how are you')
```

Require a value (non-optional parameters)

```
content: str = Body(...)
```

```
content: str = Body(Ellipsis)
```

Validators

Require minimum length

```
content: str = Body(...,
        min_length=10
```

Validators

Require minimum length

```
content: str = Body(...,
        min_length=10
```

Require maximum length

```
content: str = Body(...,
        max_length=50
```

Validators

Require minimum length

```
content: str = Body(...,
         min_length=10
```

Require maximum length

```
content: str = Body(...,
         max_length=50
```

Regex validation

```
content: str = Body(...,
         regex='regex'
```

FastAPI

Multiple values

Multiple values

For query parameters

Multiple values

For query parameters

```
localhost:8001/blog/new/2/comment?commentId=4&v=1.0&v=1.1&v=1.2&v=3
```

Multiple values

For query parameters

```
localhost:8001/blog/new/2/comment?commentId=4&v=1.0&v=1.1&v=1.2&v=3
```

Define an optional query parameter

```
v: Optional[List[str]] = Query(None)
```

Multiple values

For query parameters

```
localhost:8001/blog/new/2/comment?commentId=4&v=1.0&v=1.1&v=1.2&v=3
```

Define an optional query parameter

```
v: Optional[List[str]] = Query(None)
```

Provide default values

```
v: Optional[List[str]] = Query(['1.0', '1.1', '1.2'])
```

FastAPI

Number validators

Number validators

Greater than

```
comment_id: int = Path(None, gt=5)
```

Number validators

Greater than

```
comment_id: int = Path(None, gt=5)
```

Greater than or equal to

```
comment_id: int = Path(None, ge=5)
```

Number validators

Greater than

```
comment_id: int = Path(None, gt=5)
```

Greater than or equal to

```
comment_id: int = Path(None, ge=5)
```

Less than

```
comment_id: int = Path(None, lt=5)
```

Number validators

Greater than

```
comment_id: int = Path(None, gt=5)
```

Greater than or equal to

```
comment_id: int = Path(None, ge=5)
```

Less than

```
comment_id: int = Path(None, lt=5)
```

Less than or equal to

```
comment_id: int = Path(None, le=5)
```

FastAPI

Complex subtypes

Complex subtypes

Pydantic models are not restricted to simple types

Complex subtypes

Pydantic models are not restricted to simple types

```
tags: List[str] = []
```

Complex subtypes

Pydantic models are not restricted to simple types

```
tags: List[str] = []
```

List, set, dict, tuple

Complex subtypes

Pydantic models are not restricted to simple types

```
tags: List[str] = []
```

List, set, dict, tuple

```
metadata: Dict[str, str] = {'key1': 'val1'}
```

Complex subtypes

Pydantic models are not restricted to simple types

```
tags: List[str] = []
```

List, set, dict, tuple

```
metadata: Dict[str, str] = {'key1': 'val1'}
```

Custom model subtypes

Complex subtypes

Pydantic models are not restricted to simple types

```
tags: List[str] = []
```

List, set, dict, tuple

```
metadata: Dict[str, str] = {'key1': 'val1'}
```

Custom model subtypes

```
class Image(BaseModel):
    url: str
    alias: str
```

Complex subtypes

Pydantic models are not restricted to simple types

```
tags: List[str] = []
```

List, set, dict, tuple

```
metadata: Dict[str, str] = {'key1': 'val1'}
```

Custom model subtypes

```
class Image(BaseModel):
    url: str
    alias: str
```

```
image: Optional[Image] = None
```

FastAPI

Files section overview

Files section overview

File

UploadFile

Making files statically available

Downloading files

FastAPI

File

File

Declared similarly to Form fields

File

Declared similarly to Form fields

```
def get_file(file: bytes = File(...)):
```

File

Declared similarly to Form fields

```
def get_file(file: bytes = File(...)):
```

Received as bytes

File

Declared similarly to Form fields

```
def get_file(file: bytes = File(...)):
```

Received as bytes

Stored in memory

FastAPI

UploadFile

UploadFile

Provides more functionality

UploadFile

Provides more functionality

Stored in memory up to a certain size, then on disk

UploadFile

Provides more functionality

Stored in memory up to a certain size, then on disk

Python file like object

Upload File[s]

- https://fastapi.tiangolo.com/tutorial/request-files
- pip install python-multipart

UploadFile

Provides more functionality

Stored in memory up to a certain size, then on disk

Python file like object

```
def get_upload_file(upload_file: UploadFile = File(...)):
```

FastAPI

Download file

Download file

Provide more logic around file access

Download file

Provide more logic around file access

Provide security

Download file

Provide more logic around file access

Provide security

```
@router.get("/download/{name}", response_class=FileResponse)
def download_file(name: str):
  ...
  return path
```

FastAPI

Logger

Logger

```
import logging

logging = logging.getLogger()
```

- Logging levels: DEBUG, INFO, WARNING, ERROR, CRITICAL

- logging.debug('Debug')
- logging.info('Info')
- logging.warning('Warning')
- logging.error('Error')
- logging.critical('Critical')

Config logger

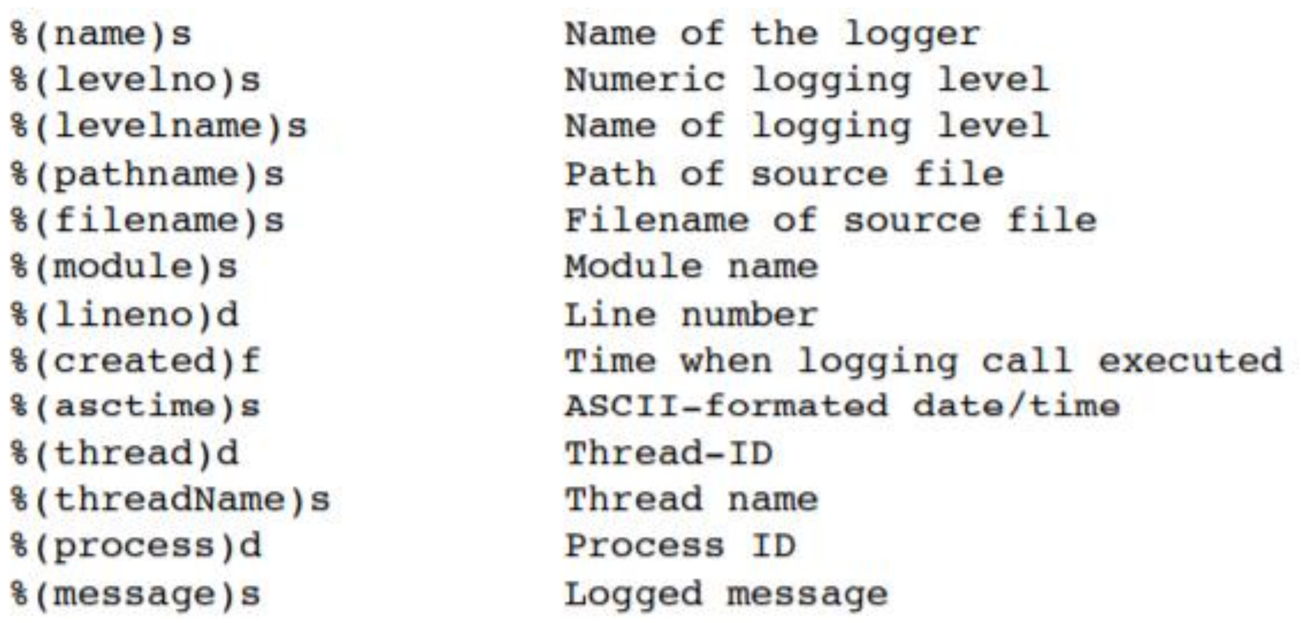
- `logging.basicConfig(level=logging.DEBUG)`

-
  ```
  logging.basicConfig(filename='app.log', filemode='w',
   format='%(name)s - %(levelname)s - %(message)s')
  ```

-
  ```
  logging.basicConfig(filename='applog_{:%Y-%m-
  %d}.log'.format(datetime.now()), filemode='a', format
  ='%(asctime)s - %(name)s -
   %(levelname)s : %(message)s')
  ```

https://realpython.com/python-logging/

Message Format



`%(name)s`	Name of the logger
`%(levelno)s`	Numeric logging level
`%(levelname)s`	Name of logging level
`%(pathname)s`	Path of source file
`%(filename)s`	Filename of source file
`%(module)s`	Module name
`%(lineno)d`	Line number
`%(created)f`	Time when logging call executed
`%(asctime)s`	ASCII-formated date/time
`%(thread)d`	Thread-ID
`%(threadName)s`	Thread name
`%(process)d`	Process ID
`%(message)s`	Logged message

Handling Errors

- https://fastapi.tiangolo.com/tutorial/handling-errors

- `try...except…finally`
- `raise HTTPException(status_code=404, detail="Item not found")`

FastAPI

Databases overview

Databases overview

Databases in FastAPI

Create database and tables

Write data

Create and read

Update and delete

Relationships

FastAPI

Databases

Databases

Any relational database

Databases

Any relational database

ORM library

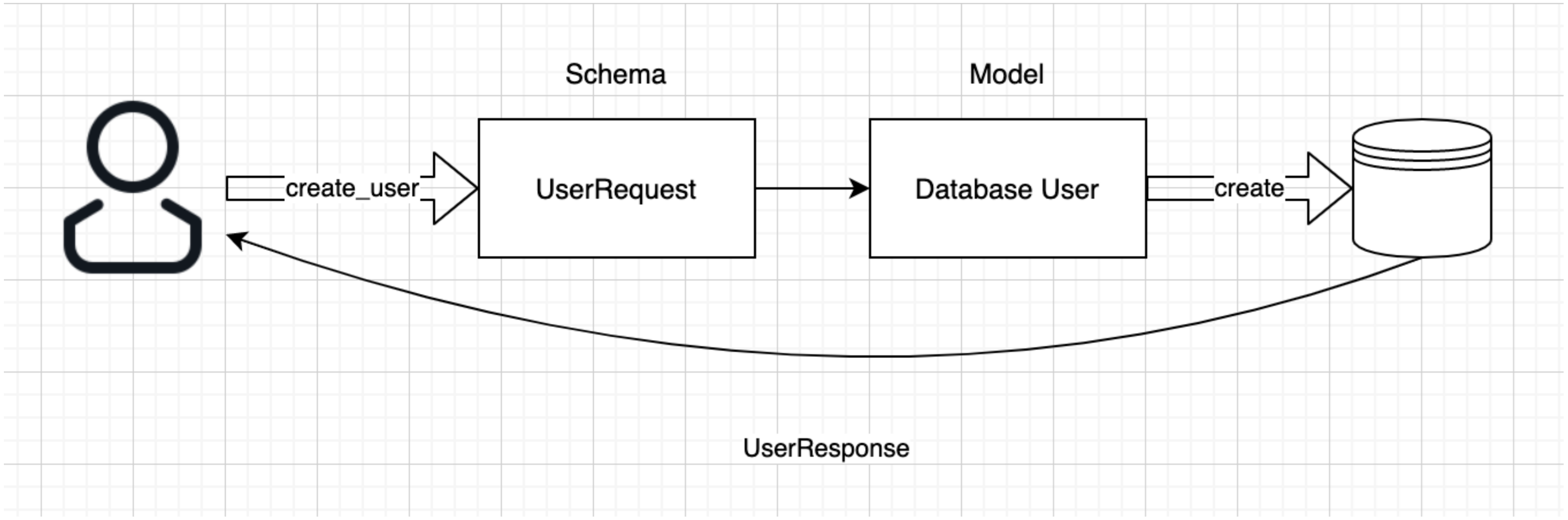
Databases

Any relational database

ORM library

SQLAlchemy

Databases


Schema
Model
create_user
UserRequest
Database User
create
UserResponse

FastAPI

Database and model

Database and model

Database definition

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./fastapi-practice.db"

engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()
```

Database and model

Table definition

```
from .database import Base
from sqlalchemy import Column, Integer, String

class DbUser(Base):
 __tablename__ = 'users'
 id = Column(Integer, primary_key=True, index=True)
 username = Column(String)
 email = Column(String)
 password = Column(String)
```

Database and model

Database creation

```
models.Base.metadata.create_all(engine)
```

FastAPI

Create user data

Create user data

Database checklist

● Database definition	database.py
● Model definition	models.py
● Create database	main.py

Create user data

Database checklist

● Database definition	database.py
● Model definition	models.py
● Create database	main.py
● Schema definition	schemas.py

Create user data

Database checklist

● Database definition	database.py
● Model definition	models.py
● Create database	main.py
● Schema definition	schemas.py
● ORM functionality	db_user.py

Create user data

Database checklist

● Database definition	database.py
● Model definition	models.py
● Create database	main.py
● Schema definition	schemas.py
● ORM functionality	db_user.py
● API functionality	user.py

FastAPI

Process review

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Create database definition and run it in main.py

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Create database definition and run it in main.py

Create database models (tables)

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Create database definition and run it in main.py

Create database models (tables)

Create functionality to write to database

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Create database definition and run it in main.py

Create database models (tables)

Create functionality to write to database

Create schemas

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Create database definition and run it in main.py

Create database models (tables)

Create functionality to write to database

Create schemas

- Data from user: UserBase
- Response to user: UserDisplay

Process review

Import required libraries: sqlalchemy, passlib, bcrypt

Create database definition and run it in main.py

Create database models (tables)

Create functionality to write to database

Create schemas

- Data from user: UserBase
- Response to user: UserDisplay

Create API operation

Next

CRUD operations: Create, Read, Update, Delete

FastAPI

Create and read

Create and read

Create element

```
db.add(new_user)
db.commit()
db.refresh(new_user)
```

Create and read

Create element

```
db.add(new_user)
db.commit()
db.refresh(new_user)
```

Read all elements

```
return db.query(DbUser).all()
```

Create and read

Create element

```
db.add(new_user)
db.commit()
db.refresh(new_user)
```

Read all elements

```
return db.query(DbUser).all()
```

Read one element

```
return db.query(DbUser).filter(DbUser.id == id).first()
```

FastAPI

Update and delete

Update and delete

Update element

```
user = db.query(DbUser).filter(DbUser.id == id)
 user.update({
   DbUser.username: request.username,
   ...
 })
 db.commit()
```

Update and delete

Update element

```
user = db.query(DbUser).filter(DbUser.id == id)
 user.update({
   DbUser.username: request.username,
   ...
 })
 db.commit()
```

Delete element

```
user = db.query(DbUser).filter(DbUser.id == id).first()
 db.delete(user)
 db.commit()
```

FastAPI

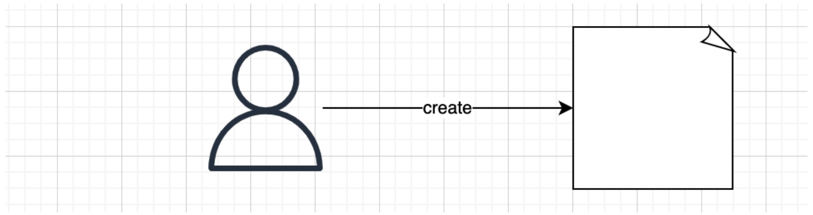
Relationships

Relationships

Retrieve elements from multiple tables in a single request

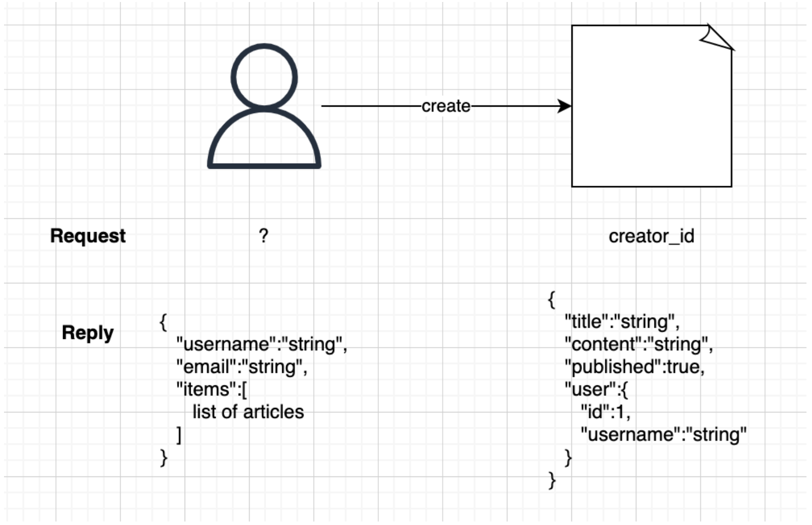
Relationships

Retrieve elements from multiple tables in a single request



Relationships

Retrieve elements from multiple tables in a single request



```
{
  "username":"string",
  "email":"string",
  "items":[
    list of articles
  ]
}
```

```
{
  "title":"string",
  "content":"string",
  "published":true,
  "user":{
    "id":1,
    "username":"string"
  }
}
```

Relationships

Define relationship in models

Relationships

Define relationship in models

```
items = relationship("DbArticle", back_populates="user")
```

Relationships

Define relationship in models

```
items = relationship("DbArticle", back_populates="user")
```

```
user_id = Column(Integer, ForeignKey("users.id"))
user = relationship("DbUser", back_populates="items")
```

Relationships

Define relationship in models

```
items = relationship("DbArticle", back_populates="user")
```

```
user_id = Column(Integer, ForeignKey("users.id"))
user = relationship("DbUser", back_populates="items")
```

Add the elements we want to retrieve in schemas

Relationships

Define relationship in models

```
items = relationship("DbArticle", back_populates="user")
```

```
user_id = Column(Integer, ForeignKey("users.id"))
user = relationship("DbUser", back_populates="items")
```

Add the elements we want to retrieve in schemas

```
class UserDisplay(BaseModel):
...
items: List[Article] = []
```

Relationships

Define relationship in models

```
items = relationship("DbArticle", back_populates="user")
```

```
user_id = Column(Integer, ForeignKey("users.id"))
user = relationship("DbUser", back_populates="items")
```

Add the elements we want to retrieve in schemas

```
class UserDisplay(BaseModel):
...
items: List[Article] = []
```

```
class ArticleDisplay(BaseModel):
...
user: User
```

FastAPI

Authentication overview

Authentication overview

Authentication

Securing an endpoint

Generating access token

User authentication

FastAPI

Authentication

Authentication

Complex topic

Authentication

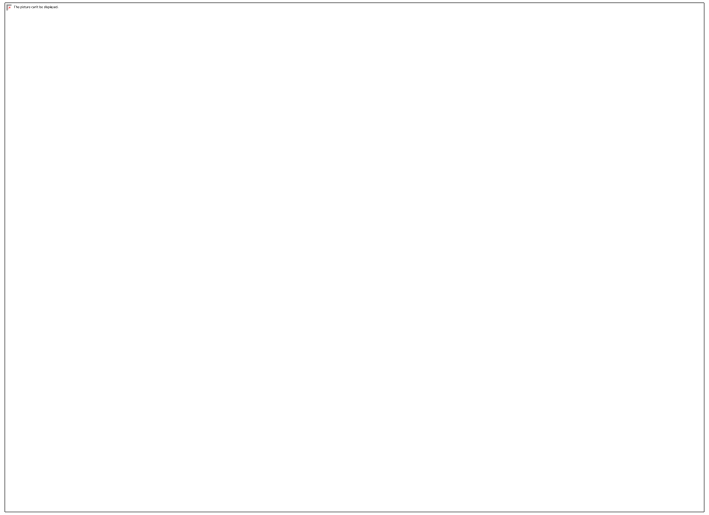
Complex topic

OAuth2 with username and password

Authentication

Complex topic

OAuth2 with username and password



FastAPI

Token

Token

Verify token

Token

Verify token

Retrieve user associated with token

Token

Verify token

Retrieve user associated with token

Secure more endpoints