PERL & PYTHON

Course Code: CS466

No. of Credit Hours: 2 credits (LEC)

Lecturer: Dang, Tran Huu Minh

Mobile/ Zalo: 0918.763.367

Email: tranhminhdang@dtu.edu.vn

SLIDE INFORMATION
INTRODUCTION

Topic:

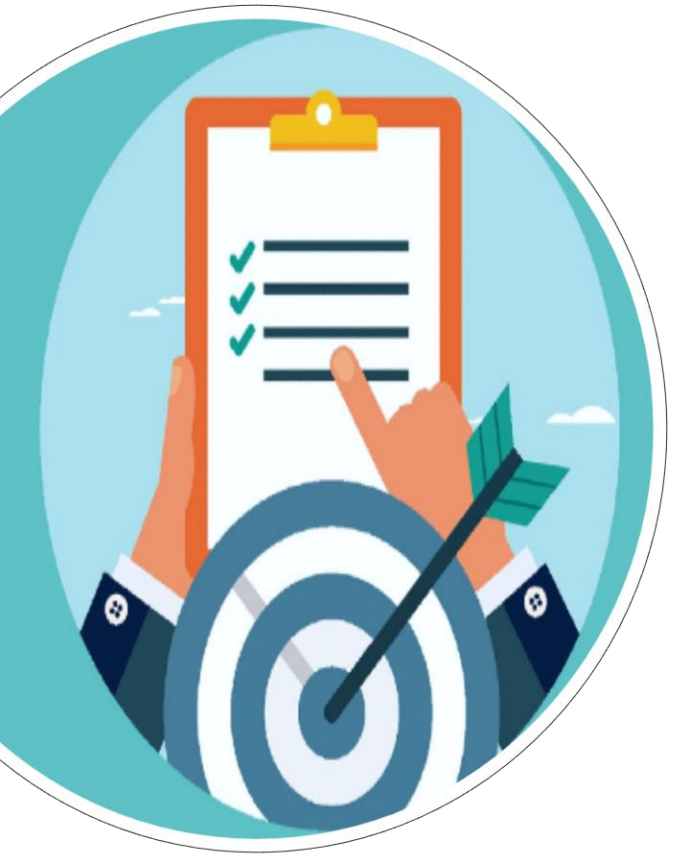
Programming for Web database in Python

Duration: 120 minutes

Lecturer: Dang, Tran Huu Minh
Mobile/ Zalo: 0918.763.367
Email: tranhminhdang@dtu.edu.vn

MỤC TIÊU

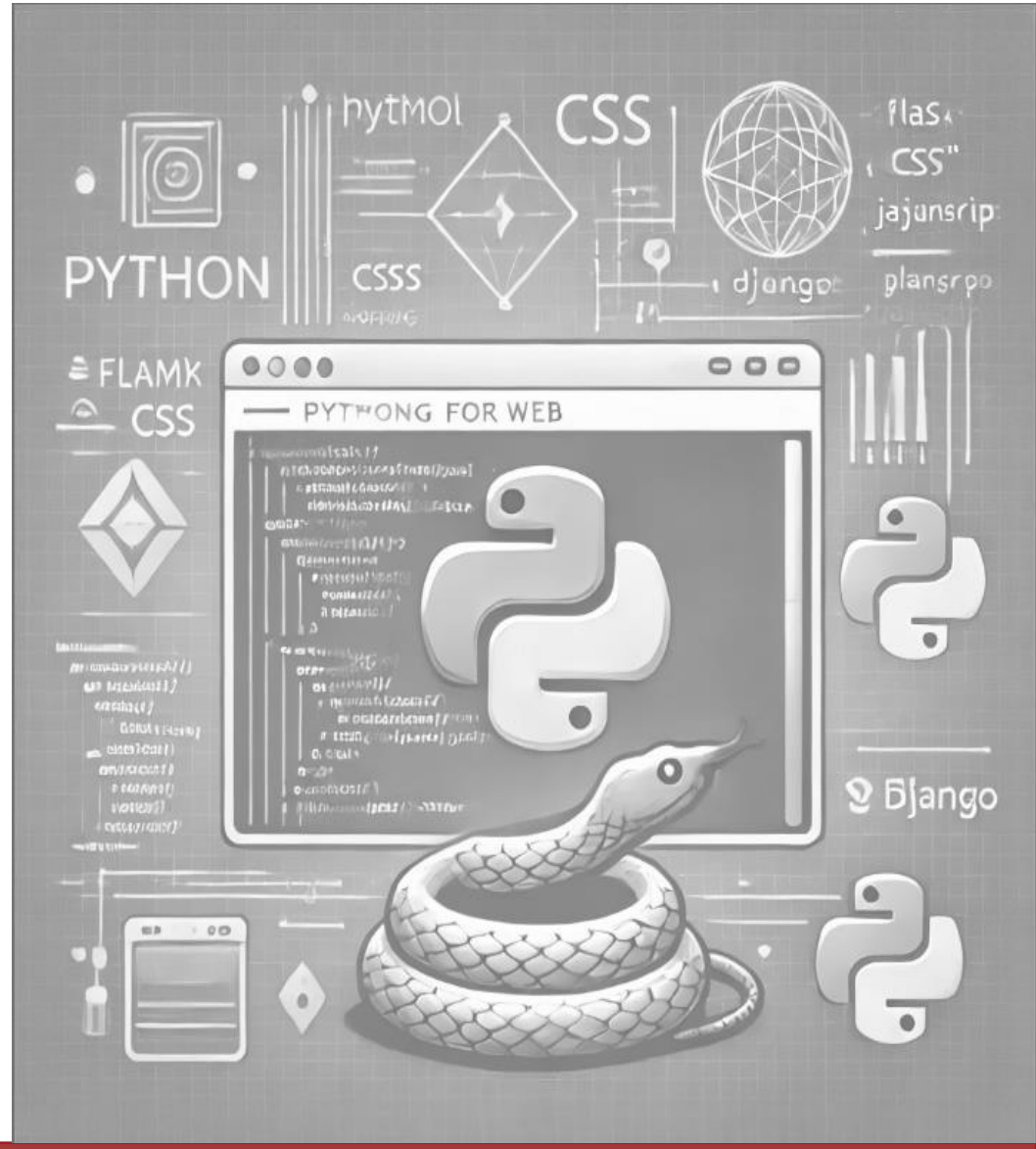
Sau khi học xong, sinh viên sẽ có được những khả năng sau:



- **Hiểu về về cơ bản về** cơ sở dữ liệu và hệ thống quản lý cơ sở dữ liệu (DBMS), các loại cơ sở dữ liệu khác nhau và ngôn ngữ truy vấn cơ sở dữ liệu (SQL) cùng với NoSQL.
- **Sử dụng được cách kết nối với** cơ sở dữ liệu thông qua việc sử dụng các thư viện kết nối phổ biến như MySQL, SQLite và nhiều hệ thống khác.
- ***Vận dụng kiến thức để*** để tạo và quản lý cơ sở dữ liệu sử dụng Django (hoặc Flask), một framework phổ biến cho việc phát triển ứng dụng web bằng Python.

NỘI DUNG

- Python và Web Frameworks
- Cơ sở dữ liệu và ORM
- RESTful API
- Authentication và Authorization
- Bảo mật trong lập trình web
- Tương tác với dịch vụ bên ngoài (API)



Python và Web Frameworks

1 **Django[2]**

Django là một framework web full-stack mạnh mẽ và được sử dụng rộng rãi. Nó cung cấp một loạt các tính năng, bao gồm quản lý cơ sở dữ liệu, định tuyến, mẫu, và các công cụ bảo mật.

2 **Flask[1]**

Flask là một framework web nhẹ và linh hoạt. hoạt. Nó cung cấp một nền tảng vững chắc cho chắc cho việc tạo ra các ứng dụng web nhỏ và nhỏ và đơn giản.

3 FastAPI

FastAPI là một framework web hiện đại, tập tập trung vào tốc độ và hiệu suất, được thiết thiết kế đặc biệt cho việc tạo ra API.

4 Pyramid

Pyramid là một framework web linh hoạt và và mở rộng, cung cấp nhiều tùy chọn cho các các dự án web phức tạp.

Cơ sở dữ liệu và ORM

Cơ sở dữ liệu

Cơ sở dữ liệu đóng vai trò lưu trữ dữ liệu cho ứng dụng web. Python hỗ trợ nhiều loại cơ sở dữ liệu khác nhau, bao gồm:

- MySQL
- PostgreSQL
- **SQLite[3]**
- MongoDB

ORM

ORM (Object-Relational Mapping) là một lớp trừu tượng giúp bạn tương tác với cơ sở dữ liệu một cách dễ dàng hơn bằng cách ánh xạ các đối tượng Python với các bảng trong cơ sở dữ liệu. Một số ORM phổ biến trong Python:

- Django ORM
- **SQLAlchemy[4]**
- Peewee

Các loại cơ sở dữ liệu phổ biến

SQLite

Hệ quản trị cơ sở dữ liệu nhỏ gọn, tích hợp trực tiếp vào Python.

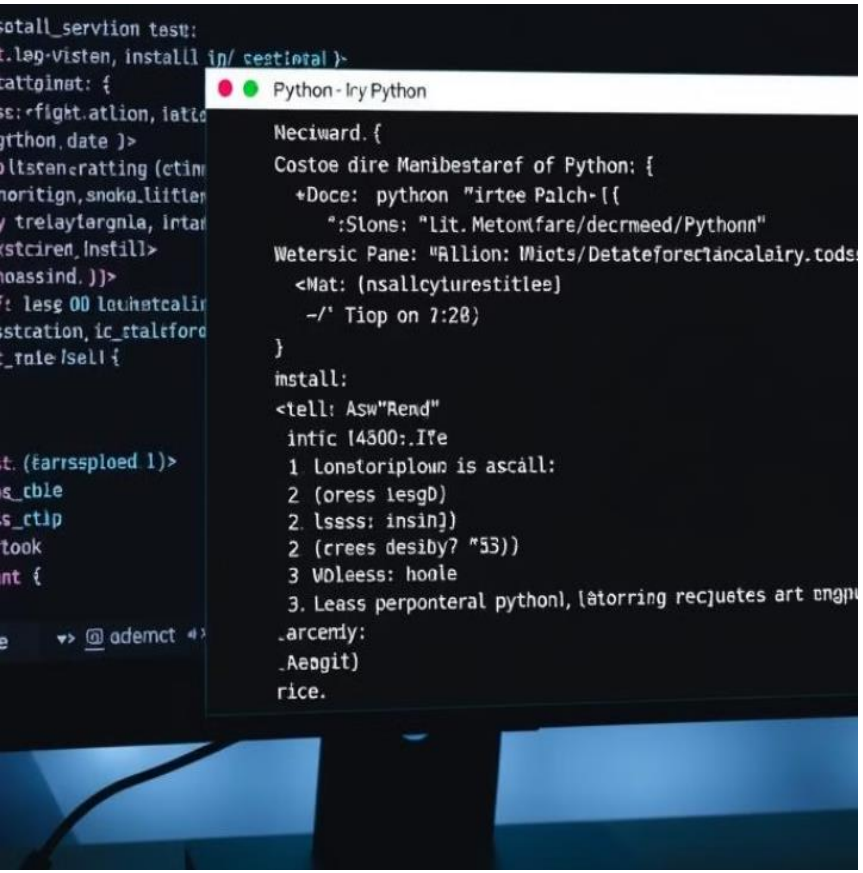
MySQL

Hệ quản trị cơ sở dữ liệu mã nguồn mở phổ biến, yêu cầu máy chủ cơ sở dữ liệu.

SQLAlchemy

Thư viện ORM (Object-Relational Mapping) mạnh mẽ, hỗ trợ quản lý cơ sở dữ liệu một cách đơn giản hơn.

Thiết lập môi trường phát triển



1. **Cài đặt Python**

   Yêu cầu phiên bản Python 3.x trở lên.

2. **Cài đặt thư viện sqlite3**

   Thư viện sqlite3 đã được tích hợp sẵn trong Python.

3. **Cài đặt thư viện MySQL Connector**

   Sử dụng lệnh
   pip install mysql-connector-python.

Kết nối với cơ sở dữ liệu trong Python

Kết nối với SQLite

SQLite là một cơ sở dữ liệu nhỏ gọn, không cần cấu hình nhiều.

```
import sqlite3

# Kết nối tới cơ sở dữ liệu
conn = sqlite3.connect('example.db')

# Tạo đối tượng cursor để thực thi truy vấn
cursor = conn.cursor()
```

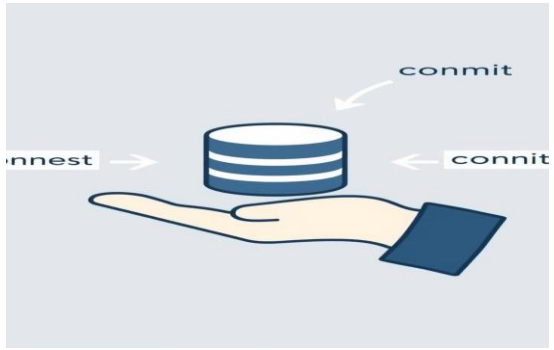
```
import mysql.connector

# Kết nối tới cơ sở dữ liệu MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword",
    database="mydatabase"
)

# Tạo đối tượng cursor
cursor = conn.cursor()
```

Kết nối với cơ sở dữ liệu trong Python

Kết nối với MySQL

MySQL là hệ thống cơ sở dữ liệu mạnh mẽ, thường được sử dụng trong các ứng dụng web quy mô lớn.

Quản lý kết nối cơ sở dữ liệu



1. **conn.commit()**

   Lưu thay đổi sau khi thực thi truy vấn SQL.

2. **conn.close()**

   Đóng kết nối khi không còn sử dụng.

```
# Sử dụng context manager để tự động quản lý kết nối
with sqlite3.connect('mydatabase.db') as conn:
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM users')
```

Ví dụ: SQLAlchemy

```
pip install sqlalchemy

from sqlalchemy import create_engine

engine = create_engine('sqlite:///mydatabase.db')
connection = engine.connect()
result = connection.execute("SELECT * FROM users")
for row in result:
    print(row)
```

Ví dụ:

SQLite và Flask

Sử dụng Flask (một micro-framework cho Python) để phát triển ứng dụng web có kết nối với cơ sở dữ liệu.

```
from flask import Flask, g
import sqlite3
app = Flask(__name__)
def get_db():
    if 'db' not in g:
        g.db = sqlite3.connect('mydatabase.db')
    return g.db

@app.route('/')
def index():
    db = get_db()
    cursor = db.execute('SELECT * FROM users‘)
    users = cursor.fetchall()
    return str(users)
if __name__ == '__main__':
    app.run(debug=True)
```

SQL Queries và Tương tác với Python

Sử dụng Python để thực hiện các truy vấn SQL cơ bản như SELECT, INSERT, UPDATE và DELETE với các cơ sở dữ liệu như SQLite và MySQL. Bạn sẽ học cách kết nối Python với các cơ sở dữ liệu này, viết các truy vấn SQL và xử lý kết quả trả về.

Thực hiện truy vấn SQL cơ bản trong Python

1. **INSERT - Thêm dữ liệu vào bảng**

   Cú pháp SQL: INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);

2. **SELECT - Truy xuất dữ liệu từ bảng**

   Cú pháp SQL: SELECT column1, column2, ... FROM table_name WHERE condition;

3. **UPDATE - Cập nhật dữ liệu trong bảng**

   Cú pháp SQL: UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;

4. **DELETE - Xóa dữ liệu khỏi bảng**

   Cú pháp SQL: DELETE FROM table_name WHERE condition;

Ví dụ Thực thi trong Python

1 SELECT

```
cursor.execute(
"SELECT name, age FROM users
WHERE age > 25“
)

# Lấy tất cả kết quả truy vấn
results = cursor.fetchall()
for row in results:
    print(row)
```

2 INSERT

```
cursor.execute(
"INSERT INTO users (name, age)
VALUES (%s, %s)", ("John", 30)
)

# Xác nhận thay đổi
conn.commit()
```

Ví dụ Thực thi trong Python

3 UPDATE

```
cursor.execute(
"UPDATE users SET age = %s
WHERE name = %s", (35, "John")
)

# Xác nhận thay đổi
conn.commit()
print(cursor.rowcount, "record(s)
updated")
```

2 DELETE

```
cursor.execute(
"DELETE FROM users WHERE
name = %s", ("John",)
)

# Xác nhận thay đổi
conn.commit()
print(cursor.rowcount, "record(s)
updated")
```

Xử lý lỗi và bảo mật

Xử lý lỗi với try-except:

Bảo vệ ứng dụng khỏi các lỗi cơ sở dữ liệu bằng cách sử dụng cấu trúc try-except

```
try:
    cursor.execute("SELECT * FROM non_existing_table")
except sqlite3.Error as e:
    print(f"An error occurred: {e}")
```

Xử lý lỗi và bảo mật

Chống SQL Injection

Sử dụng truy vấn tham số để bảo vệ cơ sở dữ liệu khỏi các tấn công SQL Injection. Luôn sử dụng execute() với dấu %s hoặc ? thay vì kết hợp chuỗi trực tiếp.

```
cursor.execute(
"SELECT * FROM users
WHERE name = %s", ("John",)
)
```

Xây dựng RESTful API để Truy cập Cơ sở dữ liệu

RESTful API là một công cụ mạnh mẽ cho phép các ứng dụng giao tiếp với nhau một cách hiệu quả và linh hoạt. Bài học này sẽ hướng dẫn bạn cách xây dựng RESTful API bằng Python và Flask để tương tác với cơ sở dữ liệu, từ đó mở ra cánh cửa cho việc phát triển các ứng dụng web phức tạp và hiệu quả.

Giới thiệu về RESTful API

RESTful API là một kiến trúc API phổ biến, cho phép các dịch vụ web giao tiếp qua HTTP với các phương thức tiêu chuẩn như GET, POST, PUT, DELETE. RESTful API cung cấp một cách tiếp cận đơn giản và hiệu quả để truy cập và quản lý dữ liệu từ cơ sở dữ liệu web.

1 Mục tiêu

Hiểu khái niệm cơ bản của RESTful API và tầm quan trọng của nó trong việc truy cập dữ liệu từ cơ sở dữ liệu web.

2 API

Một tập hợp các quy tắc cho phép các ứng dụng giao tiếp với nhau.

3 REST

Là một kiến trúc API phổ biến, cho phép các dịch vụ web giao tiếp qua HTTP với các phương thức tiêu chuẩn như GET, POST, PUT, DELETE.

Thiết lập Môi trường cho RESTful API với Flask

Flask là một micro-framework cho Python, phổ biến trong việc phát triển các ứng dụng web và xây dựng RESTful API. Flask cung cấp một nền tảng vững chắc để xây dựng các API đơn giản và hiệu quả.

Cài đặt Flask

pip install Flask

Cấu trúc Dự án

```
project/
├── app.py            # File chính chứa mã nguồn API
├── database.db       # Cơ sở dữ liệu SQLite (hoặc kết nối với MySQL)
└── requirements.txt # Danh sách các thư viện cần cài đặt
```

Xây dựng RESTful API với Flask

Xây dựng RESTful API với Flask bao gồm các bước cơ bản như khởi tạo ứng dụng Flask, kết nối với cơ sở dữ liệu, định nghĩa các route API và xử lý các phương thức HTTP.

1 Khởi tạo Ứng dụng Flask

```
from flask import Flask, jsonify, request
app = Flask(__name__)
@app.route('/')
def index():
    return "Welcome to the API!"
if __name__ == '__main__':
    app.run(debug=True)
```

Xây dựng RESTful API với Flask

Xây dựng RESTful API với Flask bao gồm các bước cơ bản như khởi tạo ứng dụng Flask, kết nối với cơ sở dữ liệu, định nghĩa các route API và xử lý các phương thức HTTP.

2 Kết nối với Cơ sở dữ liệu

```
import sqlite3

def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn
```

```
import mysql.connector
def get_db_connection():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="yourpassword",
        database="mydatabase"
    )
    return conn
```

Xây dựng RESTful API với Flask

Xây dựng RESTful API với Flask bao gồm các bước cơ bản như khởi tạo ứng dụng Flask, kết nối với cơ sở dữ liệu, định nghĩa các route API và xử lý các phương thức HTTP.

3 Các Phương thức HTTP

- **GET:** Truy xuất dữ liệu từ cơ sở dữ liệu.
- **POST:** Thêm mới dữ liệu vào cơ sở dữ liệu.
- **PUT:** Cập nhật dữ liệu hiện có trong cơ sở dữ liệu.
- **DELETE:** Xóa dữ liệu từ cơ sở dữ liệu

Xây dựng các Route API

Các route API xác định các điểm truy cập vào dữ liệu và chức năng của API. Mỗi route tương ứng với một phương thức HTTP cụ thể, cho phép thực hiện các thao tác CRUD (Create, Read, Update, Delete) với cơ sở dữ liệu.

- API để truy xuất dữ liệu (GET)
- API để thêm người dùng mới (POST)
- API để cập nhật người dùng (PUT)
- API để xóa người dùng (DELETE)

API để truy xuất dữ liệu (GET)

```
@app.route('/users', methods=['GET'])
def get_users():
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM users')
    users = cursor.fetchall()
    conn.close()

    # Chuyển kết quả truy vấn thành danh sách
    user_list = [dict(row) for row in users]
    return jsonify(user_list)
```

API để thêm người dùng mới (POST)

```
@app.route('/users', methods=['POST'])
def create_user():
    new_user = request.get_json()
    # Kết nối cơ sở dữ liệu
    conn = get_db_connection()
    conn.execute('INSERT INTO users (name, age) VALUES (?, ?)',
                 (new_user['name'], new_user['age']))
    conn.commit()
    conn.close()
    return jsonify(new_user), 201
```

API để cập nhật người dùng (PUT)

```
@app.route('/users/<int:id>', methods=['PUT'])
def update_user(id):
    update_data = request.get_json()
    conn = get_db_connection()
    conn.execute('UPDATE users SET name = ?, age = ? WHERE id = ?',
                 (update_data['name'], update_data['age'], id))
    conn.commit()
    conn.close()
    return jsonify(update_data)
```

API để xóa người dùng (DELETE)

```
@app.route('/users/<int:id>', methods=['DELETE'])
def delete_user(id):
    conn = get_db_connection()
    conn.execute('DELETE FROM users WHERE id = ?', (id,))
    conn.commit()
    conn.close()

    return jsonify({'message': f'User with id {id} deleted'}), 200
```

Xử lý Lỗi và Bảo mật API

Xử lý lỗi và bảo mật API là những khía cạnh quan trọng để đảm bảo API hoạt động ổn định và an toàn. Xử lý lỗi giúp API cung cấp phản hồi thích hợp khi gặp lỗi, trong khi bảo mật API giúp bảo vệ API khỏi truy cập trái phép.

1

Xử lý Lỗi

2

Bảo mật API

Xử lý Lỗi

Cung cấp phản hồi thích hợp khi gặp lỗi, như trường hợp truy vấn không tìm thấy dữ liệu.

```
@app.errorhandler(404)
def not_found(error):
    return jsonify({'message': 'Resource not found'}), 404
```

Bảo mật API

Sử dụng các phương pháp xác thực như API keys, OAuth để bảo vệ API khỏi truy cập trái phép.

```
from functools import wraps
def require_api_key(f):
    @wraps(f)
    def decorated_function(*args, kwargs):
        if request.headers.get('X-API-KEY') != 'your-api-key':
            return jsonify({'message': 'Unauthorized'}), 403
        return f(*args, kwargs)
    return decorated_function
@app.route('/secure-data')
@require_api_key
def secure_data():
    return jsonify({'message': 'This is secured data'})
```

Tích hợp RESTful API với Ứng dụng Web

RESTful API có thể được sử dụng để giao tiếp giữa frontend và backend. Frontend (HTML, JavaScript) gửi yêu cầu (request) đến API, API sẽ xử lý yêu cầu và trả về dữ liệu (response) để frontend hiển thị.

Frontend

HTML, JavaScript

RESTful API

Xử lý yêu cầu và trả về dữ liệu

Cơ sở dữ liệu

Lưu trữ và quản lý dữ liệu

Triển khai RESTful API

Sau khi hoàn thành việc xây dựng API, bạn có thể triển khai API lên các nền tảng đám mây như Heroku, AWS, hoặc Google Cloud để cho phép truy cập từ bất kỳ đâu.

Bước 1	Đóng gói ứng dụng vào một file requirements.txt để mô tả các thư viện cần thiết.
Bước 2	Triển khai ứng dụng Flask lên các dịch vụ hosting như Heroku, AWS, hoặc Google Cloud.

Bảo mật trong tương tác cơ sở dữ liệu web

Rủi ro bảo mật phổ biến

Tấn công SQL Injection, giả mạo danh tính, truy cập trái phép, đánh cắp dữ liệu.

Tầm quan trọng

Đảm bảo bảo mật cơ sở dữ liệu giúp bảo vệ dữ liệu nhạy cảm của người dùng và ngăn chặn các cuộc tấn công từ hacker.

Xác thực người dùng

1 Authentication (Xác thực)

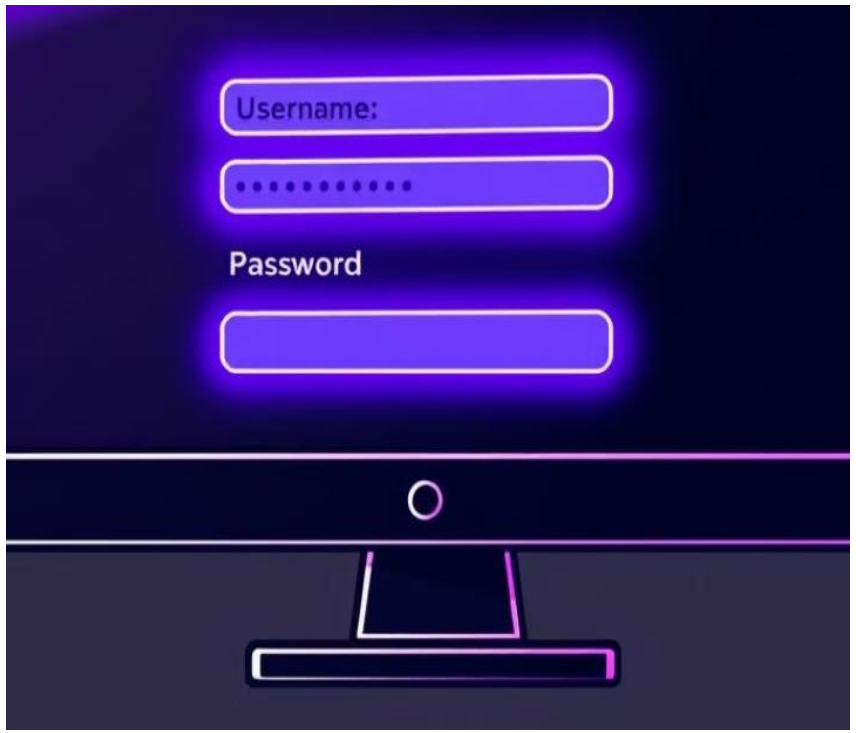
Là quá trình xác nhận danh tính của người dùng, bảo đảm chỉ người dùng hợp lệ mới có thể truy cập tài nguyên.

2 Quy trình xác thực người dùng

Sử dụng token-based authentication như JWT (JSON Web Token). Cơ chế OAuth cho phép các ứng dụng xác thực thông qua các nhà cung cấp bên thứ ba như Google, Facebook.

```
from flask import request, jsonify
users = {
    "admin": "password123"
}
def authenticate(username, password):
    if users.get(username) == password:
        return True
    return False
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    if authenticate(data['username'], data['password']):
        return jsonify({'message': 'Login successful!'}), 200
    else:
        return jsonify({'message': 'Invalid credentials'}), 401
```

Ví dụ xác thực đơn giản



Chống tấn công SQL Injection

SQL Injection

Là kiểu tấn công phổ biến mà kẻ tấn công có thể can thiệp vào truy vấn SQL thông qua input của người dùng, dẫn đến thao tác trái phép trên cơ sở dữ liệu.

Ví dụ về lỗ hổng SQL Injection

```
# Mã không an toàn
@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    conn = get_db_connection()
    user = conn.execute(
            f"SELECT * FROM users
            WHERE id = {id}").fetchone()
    return jsonify(dict(user))
```

Cách ngăn chặn SQL Injection

1

Sử dụng query parameter hóa

```
@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    conn = get_db_connection()
    user = conn.execute(
            "SELECT * FROM users
            WHERE id = ?", (id,)).fetchone()
    return jsonify(dict(user))
```

Cách ngăn chặn SQL Injection

2

Sử dụng ORM

ORM giúp giảm nguy cơ tấn công bằng cách trừu tượng hóa các truy vấn SQL.
Ví dụ ORM phổ biến là SQLAlchemy.

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy(app)
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80))
    age = db.Column(db.Integer)
# Sử dụng ORM để truy xuất dữ liệu
@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    user = User.query.filter_by(id=id).first()
    return jsonify({"name": user.name, "age": user.age})
```

Mã hóa dữ liệu nhạy cảm

1 **Hashing mật khẩu**

Mật khẩu của người dùng không bao giờ nên lưu trực tiếp trong cơ sở dữ liệu. Thay vào đó, sử dụng các thuật toán mã hóa như bcrypt hoặc Argon2 để hash mật khẩu.

2 **Mã hóa dữ liệu**

Khi lưu trữ thông tin nhạy cảm, có thể sử dụng mã hóa symetric hoặc asymetric để bảo vệ dữ liệu.

Ví dụ Hashing mật khẩu

```
from werkzeug.security import generate_password_hash, check_password_hash
# Lưu mật khẩu dưới dạng hash
hashed_password = generate_password_hash('mysecretpassword')
# Kiểm tra mật khẩu khi người dùng đăng nhập
check_password = check_password_hash(hashed_password, 'mysecretpassword')
```

Ví dụ Mã hóa dữ liệu

```
from cryptography.fernet import Fernet
# Tạo khóa mã hóa
key = Fernet.generate_key()
cipher = Fernet(key)
# Mã hóa dữ liệu
encrypted_data = cipher.encrypt(b"My sensitive data")
# Giải mã dữ liệu
decrypted_data = cipher.decrypt(encrypted_data)
```

Sử dụng SSL/TLS

1 **SSL/TLS**

Là các giao thức giúp mã hóa dữ liệu trong quá trình truyền tải giữa client và server, ngăn chặn việc nghe lén hoặc can thiệp vào dữ liệu.

2 **Cách thiết lập SSL/TLS cho Flask app**

Sử dụng thư viện flask-talisman để triển khai HTTPS.

```
from flask import Flask
from flask_talisman import Talisman
app = Flask(__name__)
Talisman(app)
@app.route('/')
def home():
    return 'Secure connection using HTTPS!'
```

Xác thực hai yếu tố (2FA)

```
import pyotp

# Tạo mã 2FA
totp = pyotp.TOTP('base32secret3232')
print(totp.now())  # Mã 2FA hiện tại
```

1 **Two-factor authentication (2FA)**

Là một phương thức bảo mật nâng cao yêu cầu người dùng nhập thêm mã xác thực từ thiết bị thứ hai, như ứng dụng Authenticator hoặc tin nhắn SMS.

2 **Tích hợp 2FA trong ứng dụng Python**

Sử dụng thư viện như pyotp để tạo mã 2FA.

Giới hạn quyền truy cập và phân quyền người dùng

Nguyên tắc phân quyền: Chỉ cho phép người dùng truy cập vào dữ liệu và chức năng mà họ có quyền.
Triển khai phân quyền trong Flask: Sử dụng decorator để kiểm tra vai trò của người dùng trước khi cho phép truy cập

```
from functools import wraps
from flask import abort
def admin_required(f):
    @wraps(f)
    def decorated_function(*args, kwargs):
        if not current_user.is_admin:
            abort(403)
        return f(*args, kwargs)
    return decorated_function
@app.route('/admin')
@admin_required
def admin_dashboard():
    return 'Welcome, Admin!'
```

Tối ưu hóa Cơ sở dữ liệu Web trong Python

Tối ưu hóa hiệu suất cơ sở dữ liệu web trong các ứng dụng Python là một nhiệm vụ phức tạp nhưng cần thiết. Mục tiêu chính là giảm thiểu thời gian phản hồi, tăng khả năng chịu tải và đảm bảo ứng dụng hoạt động ổn định ngay cả khi lượng truy cập tăng đột biến.

Sử dụng Cache

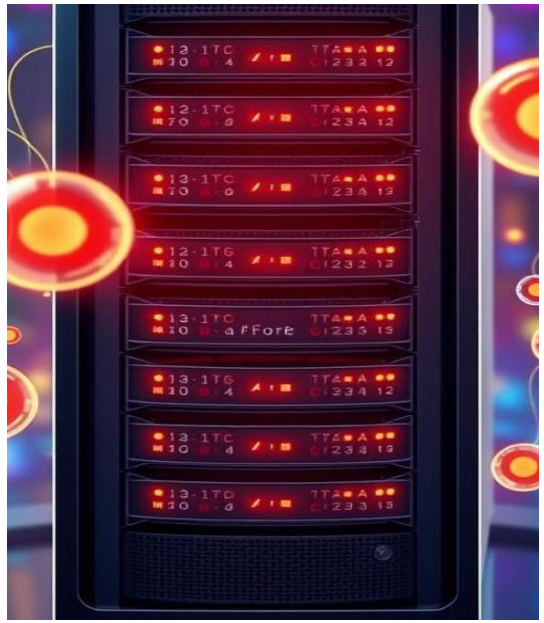
Cache là một kỹ thuật lưu trữ tạm thời các kết quả truy vấn cơ sở dữ liệu, giúp giảm thời gian truy xuất dữ liệu từ cơ sở dữ liệu bằng cách cung cấp dữ liệu đã lưu trữ từ bộ nhớ đệm. Điều này giúp giảm thiểu số lượng truy vấn đến cơ sở dữ liệu chính, từ đó cải thiện tốc độ xử lý và giảm tải cho hệ thống.

1 **Công cụ phổ biến**

Redis, Memcached, Flask-Caching.

2 **Lợi ích**

Giảm số lượng truy vấn cơ sở dữ liệu, cải thiện tốc độ xử lý, giảm tải cho cơ sở dữ liệu chính.



Ví dụ sử dụng Redis cache

```
import redis
import json
# Kết nối với Redis
r = redis.Redis(host='localhost', port=6379, db=0)
def get_user_data(user_id):
    # Kiểm tra xem dữ liệu đã có trong cache chưa
    cache_key = f"user:{user_id}"
    cached_data = r.get(cache_key)
    if cached_data:
        return json.loads(cached_data)
   # Nếu không có trong cache, truy vấn cơ sở dữ liệu
    user_data = query_database_for_user(user_id)
    # Lưu dữ liệu vào cache
    r.set(cache_key, json.dumps(user_data), ex=300)  # Cache dữ liệu trong 5 phút
    return user_data
```

Load Balancing

Load balancing là kỹ thuật phân phối đều các yêu cầu truy vấn cơ sở dữ liệu đến nhiều máy chủ, giảm thiểu tình trạng quá tải và tăng cường khả năng chịu tải của hệ thống. Điều này giúp đảm bảo hệ thống hoạt động liên tục ngay cả khi có một hoặc nhiều máy chủ gặp sự cố.

Công cụ phổ biến

Nginx, HAProxy.

Triển khai cơ bản

Sử dụng Nginx để phân phối yêu cầu giữa các máy chủ cơ sở dữ liệu replica.

Tối ưu hóa Truy vấn SQL

Tối ưu hóa truy vấn SQL là một bước quan trọng để cải thiện hiệu suất của cơ sở dữ liệu. Việc sử dụng chỉ mục, phân tích kế hoạch truy vấn và tối ưu hóa JOINs giúp giảm thiểu thời gian xử lý truy vấn và tăng cường hiệu quả của hệ thống.

1. **Chỉ mục**

   Chỉ mục giúp cải thiện tốc độ truy vấn dữ liệu bằng cách cung cấp cách thức tìm kiếm nhanh hơn trong bảng cơ sở dữ liệu.

2. **Phân tích kế hoạch truy vấn**

   Sử dụng công cụ EXPLAIN để phân tích và tối ưu hóa truy vấn SQL.

3. **Tối ưu hóa JOIN**

   Tránh sử dụng quá nhiều JOINs hoặc sử dụng subquery khi cần thiết để giảm tải cho cơ sở dữ liệu.

Tối ưu hóa Truy vấn với Big Data

Khi dữ liệu lớn và phức tạp, việc xử lý và truy xuất dữ liệu đòi hỏi các công cụ và phương pháp tối ưu hơn. Các công cụ xử lý Big Data như Apache Hadoop, Spark có thể được tích hợp với cơ sở dữ liệu Python để phân tích và truy xuất dữ liệu nhanh chóng.

Sử dụng thư viện PySpark để phân tích và tối ưu hóa truy vấn dữ liệu lớn. PySpark cung cấp một framework mạnh mẽ để xử lý các tập dữ liệu lớn và phức tạp, giúp cải thiện hiệu suất của hệ thống cơ sở dữ liệu.

Tối ưu hóa Kết nối Cơ sở dữ liệu

Connection pooling là kỹ thuật tái sử dụng kết nối cơ sở dữ liệu thay vì tạo kết nối mới cho mỗi truy vấn. Điều này giúp giảm thời gian kết nối và tiết kiệm tài nguyên, từ đó cải thiện hiệu suất của hệ thống.

Ví dụ sử dụng SQLAlchemy với connection pooling

```
from sqlalchemy import create_engine

# Tạo engine với connection pool
engine = create_engine(
"mysql+pymysql://user:password@localhost/
mydb", pool_size=10, max_overflow=20
)

# Kết nối và thực hiện truy vấn
connection = engine.connect()
result = connection.execute("SELECT *
FROM users")
```

Programming for Web database in Python

References

1. Giới thiệu thư viện sqlite3 trong Python: https://docs.python.org/3/library/sqlite3.html
2. Hướng dẫn kết nối Python với MySQL: https://dev.mysql.com/doc/connector-python/en/
3. Các thao tác cơ bản với SQL trong Python: https://www.sqlitetutorial.net/sqlite-python/
4. Hướng dẫn tạo API bằng Flask: https://flask-restful.readthedocs.io/en/latest/
5. Hướng dẫn chống tấn công SQL Injection: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
6. Tối ưu hóa hiệu suất với SQLAlchemy trong Python: https://docs.sqlalchemy.org/en/14/faq/performance.html

References

1. Giới thiệu cơ bản về SQLite và Python:
   https://www.youtube.com/watch?v=byHcYRpMgI4
2. Cách kết nối Python với MySQL:
   https://www.youtube.com/watch?v=Z1RJmh_OqeA
3. Hướng dẫn thực thi các truy vấn SQL trong Python:
   https://www.youtube.com/watch?v=Cu6gLL8gRVs
4. Hướng dẫn tạo REST API và kết nối với cơ sở dữ liệu:
   https://www.youtube.com/watch?v=FsAPt_9Bf3U
5. Hướng dẫn bảo mật ứng dụng web với Flask:
   https://www.youtube.com/watch?v=fEPk6X5jdqI
6. Tối ưu hóa hiệu suất ứng dụng web bằng Python:
   https://www.youtube.com/watch?v=eFMtq-sPYww