Flask Interview Questions

1. What is the difference between before_request and after_request in Flask?

The before_request function is executed before each request, useful for setting up data or checking permissions. The after_request function runs after each request to modify the response object, such as adding headers or logging.

2. How can you create reusable Flask views using MethodView?

MethodView allows you to create class-based views where HTTP methods (GET, POST, etc.) are represented as class methods. This promotes code reuse and better organization for similar route logic.

3. What are class-based views in Flask and when would you use them?

Class-based views organize view logic into classes instead of functions, enabling reusability and cleaner separation of concerns. Use them when multiple routes share common behavior.

4. How does Flask handle thread safety?

Flask uses thread-local storage to keep track of request-specific data (e.g., `request`, `session`). However, care must be taken with global variables, and WSGI servers like Gunicorn should handle concurrency.

5. How do you manage memory leaks or long-running requests in Flask?

Avoid global mutable objects, use proper teardown handlers, and offload long-running tasks to background workers (e.g., Celery). Monitor with memory profilers like tracemalloc.

6. What are Flask CLI commands and how do you add custom commands?

Flask CLI commands allow you to manage and extend the app via terminal. Use @app.cli.command() to define custom commands and run with `flask your_command`.

7. How do you dynamically register routes in Flask?

You can register routes dynamically using app.add_url_rule() at runtime, allowing flexible endpoint creation based on conditions or configuration files.

8. How can you implement role-based access control (RBAC) in Flask?

Use decorators to check user roles before allowing access to routes. Store role data in sessions or database and validate them before request execution.

9. What are teardown_request functions used for in Flask?

These are used to release resources or perform cleanup (like closing database connections) after each request, even if it results in an exception.

10. How does Flask's request dispatching mechanism work?

Flask matches the incoming URL to the route table using Werkzeug's URL matching system, then dispatches it to the corresponding view function.

11. How do you throttle API requests in Flask to prevent abuse?

Use Flask-Limiter to restrict the number of requests a user or IP can make over a period of time. It supports decorators and global limits.

12. What is the difference between request.get_json() and request.json in Flask?

request.get_json() is a method that explicitly parses the JSON body, while request.json is a shortcut property. Use get_json() for better error handling and control.

13. How do you implement soft deletes in Flask with SQLAlchemy?

Add a column like `is_deleted` and filter queries to ignore rows where `is_deleted=True`, allowing data recovery or audit logs.

14. How do you test Flask CLI commands?

Use `app.test_cli_runner()` to run the CLI command and check output or behavior in unit tests.

15. How do you create a custom Jinja2 filter in Flask templates?

Define a function and register it with `@app.template_filter()` to use it in your Jinja2 templates for data formatting.

16. How do you chain multiple decorators on a single Flask route?

You can apply multiple decorators by stacking them above a view function. They execute in order from top to bottom.

17. How do you return streaming responses in Flask (e.g., large CSV export)?

Use Python generators with `Response` and `stream_with_context` to yield chunks of data instead of returning it all at once.

18. How can you protect Flask admin routes with IP whitelisting?

Check the `request.remote_addr` against a list of allowed IPs and return a 403 Forbidden response if it doesn't match.

19. How do you serve multiple Flask apps on the same server (multi-app deployment)?

Use `DispatcherMiddleware` from Werkzeug to mount multiple Flask apps on different subpaths of a single WSGI server.

20. What are Flask Views and how are they different from standard route functions?

Flask Views (e.g., MethodView) encapsulate related route logic in a class instead of writing separate functions for each route.

21. How do you integrate Flask with SQLAlchemy event listeners?

You can attach event listeners to models using SQLAlchemy's @event.listens_for decorator. This allows automatic execution of code during events like before_insert, after_update, etc.

22. How do you integrate Flask with external OAuth providers (e.g., Google, Facebook)?

Use Flask-Dance or Authlib to implement OAuth flows. These libraries handle redirection, token exchange, and user authentication integration with Flask.

23. How do you integrate Flask with Prometheus for metrics?

Use prometheus_flask_exporter to add instrumentation to your Flask app. It tracks metrics like request latency and exposes them at /metrics.

24. How do you use Flask with Apache Kafka or message queues?

Flask can produce messages to Kafka using a library like kafka-python. For background processing, combine Flask with Celery or RQ workers that consume Kafka messages.

25. How do you set up Flask with MongoDB using Flask-PyMongo?

Install Flask-PyMongo, configure the Mongo URI, and use mongo.db.collection_name in your route handlers to interact with MongoDB documents.

26. How do you use Flask with SQLModel or other SQL toolkits?

You can replace SQLAlchemy with SQLModel (based on SQLAlchemy) for Pydantic integration. Define models as data classes and interact with the database using SQLAlchemy sessions.

27. How can you integrate Flask with a legacy database or external API?

Use SQLAlchemy reflection for legacy databases or the requests library to fetch data from an external API and then use it in your Flask routes or templates.

28. How do you use Flask with a Redis message broker for pub/sub?

Use the redis Python package to subscribe and publish to channels. This allows inter-process communication or real-time data handling.

29. How can you use Flask to consume external REST APIs?

Use the requests library within Flask route handlers to make GET/POST requests to external APIs and return the processed response to the client.

30. How do you connect Flask with ElasticSearch?

Use the elasticsearch Python client to query and index documents. You can build search endpoints in Flask that use ES for fast full-text search.

31. How do you implement lazy loading in Flask apps?

Lazy loading can be achieved by loading only what's needed for each view. You can delay database queries, template parts, or resources until necessary.

32. How do you handle memory-intensive views in Flask?

Use pagination for large results, stream responses for file/data downloads, and move heavy processing to background workers (e.g., Celery).

33. How do you benchmark Flask endpoints?

Use tools like ApacheBench, Locust, or JMeter to test endpoint throughput and response time. You can also use time or Flask middleware to log timings.

34. How do you reduce cold-start time in serverless Flask deployments?

Optimize imports, minimize application startup time, and precompile templates. Use services like AWS Lambda with Zappa to handle serverless deployment.

35. How do you optimize response time in a Flask API?

Enable response caching, use efficient queries, avoid unnecessary serialization, compress responses (gzip), and limit large payloads.

36. What are some profiling tools for Flask performance analysis?

Tools like cProfile, Flask-Profiler, and PyInstrument help analyze performance bottlenecks and memory usage in your Flask app.

37. How do you avoid N+1 query problems in Flask-SQLAlchemy?

Use joinedload() or subqueryload() to preload related objects, avoiding repeated SQL queries for related data in loops.

38. How can you cache route outputs in Flask with Flask-Caching?

Install Flask-Caching, configure it with a backend (e.g., Redis, SimpleCache), and decorate routes with @cache.cached(timeout=60) to cache results.

39. What are some thread-related problems with Flask and how to avoid them?

Issues arise when using global variables across threads. Use context-bound objects like g, session, or properly scoped database connections.

40. How do you handle request timeouts gracefully in Flask?

Set timeouts in your web server (like Gunicorn), use signals or background workers for long-running tasks, and catch timeout exceptions.

41. How do you write unit tests for Flask background tasks?

Use mocking to simulate Celery or RQ tasks. Check that the task was enqueued and test task logic independently from Flask.

42. How do you structure integration tests in Flask apps?

Use the test client to simulate requests, set up test databases with fixtures, and use pytest to automate full-stack endpoint testing.

43. How do you use fixtures with pytest in Flask testing?

Fixtures help create reusable setups like test clients or databases. Use @pytest.fixture to define them and pass as arguments in tests.

44. How do you mock database connections in Flask tests?

Use libraries like unittest.mock or pytest-mock to replace database calls with fake responses to test Flask logic without hitting the database.

45. How can you test email sending functionality in Flask?

Use a test SMTP server or mock flask-mail.send() function and assert it was called with the expected subject, recipients, and message.

46. How do you limit login attempts in Flask?

Implement rate-limiting using Flask-Limiter or track failed logins using sessions/IPs and temporarily block repeated attempts.

47. How do you use secure headers (e.g., Content-Security-Policy) in Flask?

Manually set headers in after_request or use extensions like Flask-Talisman to automatically apply security headers to all responses.

48. How do you prevent Clickjacking in Flask?

Set the X-Frame-Options header to DENY or SAMEORIGIN using after_request, or use Flask-Talisman to manage this securely.

49. How do you sanitize HTML inputs in Flask apps?

Use libraries like Bleach to clean user input and remove harmful scripts or tags. You should always validate and escape user-generated content before rendering it in templates.

50. How do you handle session hijacking attacks in Flask?

Enable secure cookies (`SESSION_COOKIE_SECURE=True`), use Flask-Login for managing sessions, rotate session IDs on login, and use HTTPS with proper session timeout policies.

51. How can you build a microservice using Flask?

Expose APIs with Flask using Flask-RESTful or standard route definitions, handle communication via REST or message queues (e.g., RabbitMQ), and containerize the app with Docker.

52. What is the Factory Pattern in Flask and why is it useful?

The Factory Pattern involves defining a create_app() function that initializes and returns a Flask app instance. It promotes modularity, especially useful for testing and configuration flexibility.

53. How do you structure Flask applications for maintainability in teams?

Use a modular folder structure with blueprints, separate concerns (routes, models, forms, services), use config files per environment, and add a requirements.txt or Pipfile.

54. How do you document APIs in Flask (e.g., Swagger or Postman)?

Use Flasgger, Flask-RESTX, or manually document with Postman collections. These tools help generate interactive API docs and define schemas and endpoints.

55. How do you create custom response classes in Flask?

Subclass flask.Response and override methods like __init__ or make_response() to modify how data is formatted or headers are added before returning.

56. What is flask.g used for beyond simple use cases?

g is used for storing data globally during a request (like database connections or user info). It helps avoid repeated lookups and simplifies resource cleanup.

57. How can you inspect the Flask routing table?

Run flask routes in the terminal (with Flask CLI) to list all defined routes, methods, and endpoints in your Flask app.

58. How can you write a custom Flask WSGI middleware?

Create a class that implements __call__(environ, start_response) to intercept and process WSGI requests before they reach Flask.

59. How do you use Flask in asynchronous (async/await) scenarios?

Since Flask 2.0+, route handlers can be async functions. Use await for I/O operations like DB queries. Be sure your server supports async (e.g., Hypercorn, Uvicorn).

60. How do you integrate Flask with a non-relational database like DynamoDB?

Use boto3 to interact with AWS DynamoDB. Store and retrieve structured data via the Flask route handlers using the DynamoDB client.

61. How do you handle environment-specific settings in Flask (dev/test/prod)?

Use separate config files or classes, and set FLASK_ENV or FLASK_CONFIG environment variables. Load them in create_app() based on the environment.

62. How do you monitor and profile a Flask application?

Use tools like New Relic, Sentry, Flask-Profiler, or cProfile to monitor performance, log errors, and analyze slow endpoints.

63. How do you use Redis with Flask?

Use Redis for caching, session storage, or message brokering. Install redis or flask-redis, and configure the Redis URL in your app config.

64. How do you cache data in Flask?

Use Flask-Caching with a backend like Redis or filesystem. Decorate views with @cache.cached() and configure TTLs (time to live).

65. How do you use Flask with Docker?

Create a Dockerfile that installs Flask and dependencies, expose a port, and define CMD to run the app. Use docker-compose for managing multi-container setups.

66. How do you use Flask with AWS Lambda (Zappa)?

Use Zappa to deploy Flask apps as serverless functions on AWS Lambda. It packages your app and handles deployment with API Gateway integration.

67. How do you use Flask with GraphQL?

Install Flask-GraphQL or Graphene, define your GraphQL schemas, and register the GraphQL view as an endpoint (e.g., /graphql).