Flask - Practice Sheet

10 Q&A on WSGI, application factory, routing, and extensions

Q1. What is Flask and how does it handle HTTP requests as a WSGI app?

A: Flask is a micro-framework built on Werkzeug (WSGI toolkit) and Jinja2. When a WSGI server calls the Flask app with an environ dict, Flask parses it into a Request, routes to the matching view function, and returns a WSGI response tuple.

Q2. What is the application factory pattern and why is it recommended?

A: Instead of creating app = Flask(__name__) at module level, a create_app() function instantiates Flask, registers blueprints, and initializes extensions. This enables multiple app instances with different configs (e.g., for testing) and avoids circular imports.

Q3. How does Flask routing work with Blueprints?

A: @app.route('/path') registers a URL rule and view function on the Flask app. Blueprint groups related routes under a prefix: bp = Blueprint('auth', __name__, url_prefix='/auth') and is registered via app.register_blueprint(bp) in create_app.

Q4. How does Flask-SQLAlchemy manage the database session?

A: Flask-SQLAlchemy binds a SQLAlchemy Session to the request context. db.session is available within a request and is automatically committed or rolled back at the end. Use db.session.add(obj) and db.session.commit() to persist changes.

Q5. What is Flask's request context and what does it push?

A: Pushing a request context makes request (current HTTP request), g (per-request scratch namespace), and current_app (the Flask app) available as proxies. Contexts are pushed automatically for HTTP requests; push manually in tests or CLI commands.

Q6. How does Flask-Login manage user sessions?

A: Flask-Login stores the user's ID in a signed cookie session. @login_required checks if current_user.is_authenticated before each request. The user_loader callback receives the ID from the session and returns the User object from your database.

Q7. How do you build a JSON API in Flask?

A: Return jsonify(data) from a view function for a JSON response. For structured APIs, Flask-RESTful or Flask-RESTX provides Resource classes with get/post methods and automatic input parsing with reqparse or marshmallow schemas.

Q8. How do you handle file uploads securely in Flask?

A: Access the file via request.files['field']. Always call secure_filename() on the filename to prevent directory traversal. Validate MIME type and extension. Store outside the web root or in object storage. Limit MAX_CONTENT_LENGTH in the config.

Q9. How does Flask integrate with Celery for background tasks?

A: Create a Celery instance configured with Flask's config (CELERY_BROKER_URL). Define tasks with @celery.task. In the task function, push an app context (with app.app_context()) to access Flask extensions like db. Call task.delay() to enqueue.

Q10. How do you test a Flask view function?

A: Use app.test_client() to send requests without a server. client.get('/path') returns a response with status_code and data. Use app.test_request_context() to test functions that use request context. pytest-flask provides fixtures for test_client and app.