Integration Testing

How would you implement integration tests?

Integration tests verify that different modules or services work together as expected.

Approach:

- **Isolated Environments:**
  - Use a dedicated test environment with separate databases/ services.
  - Containerize with Docker to mirror production environments.
- **Realistic Data & Scenarios:**
  - Use realistic test data to simulate user workflows.
  - Test entire workflows (e.g., product updates, feedback submission).
- **Tooling & Frameworks:**
  - **PHPUnit** for Magento 2 integration testing.
  - **Magento Integration Testing Framework** for testing persistence, service contracts, and module interaction.
  - **Codeception** for broader PHP-based testing scenarios.
  - For GraphQL: **GraphQL Playground**, **Postman**, or **Insomnia** for testing endpoints.
  - In other stacks: **Jest**, **Mocha**, or **pytest**.

Best Practices:

- Automate environment setup and teardown.
- Mock or isolate external dependencies as needed.
- Integrate into CI/CD pipelines.

Caching

Which parts of your application could benefit from caching?

Beneficial Areas:

- **Product Data**: Especially if enriched from external sources.
- **Store Configuration/Metadata**: Rarely changes but often accessed.
- **GraphQL API Responses**: Especially for frequently accessed endpoints.

- **Expensive DB Operations**: Joins, aggregations, computed columns (e.g., "Has Price").

How would you approach implementing caching?

Strategy:

- **Application-Level Caching**:
  - Use Magento’s cache pools or framework-provided cache interfaces.
  - Use Redis or Memcached as the backend.
- **Full-Page Caching**:
  - Leverage **Varnish** for storefronts.
  - Use `full_page` and `block_html` cache types in Magento.
- **Edge/HTTP Caching**:
  - Use CDN providers like Fastly or Cloudflare for static/HTML content.
- **Cache Invalidation**:
  - Use cache tags for targeted clearing.
  - Define TTLs and use observers or plugins to invalidate on data change.

Queue System for Email Sending

How would you manage the load?

Approach:

- **Decouple email sending from user interactions** using queues.
- **Enqueue email tasks** as jobs triggered by actions like order placement.
- **Background workers** consume and process email jobs.

Queue Workflow:

1. Trigger an event (e.g., order completed).
2. Add email job to the queue with relevant data.
3. Worker picks up the job, sends the email using SMTP/API.
4. Retry or log failures as needed.

Have you worked with queue systems like RabbitMQ or Celery?

Yes, experience with:

- **RabbitMQ**: Reliable and scalable for pub/sub patterns.
- **Celery (Python)**: Works well with Django/Flask for background jobs.
- **Magento Message Queue**:
  - Uses `Magento\Framework\MessageQueue`.
  - Can work with RabbitMQ, MySQL, or other adapters.
- **Redis-backed Queues**:
  - Used in Laravel Horizon, Bull (Node.js), etc.

Scalability Tips:

- Scale workers independently from the web app.
- Monitor job queues and worker health.
- Use retry and dead-letter queues for robustness.
