

01	A	B	C	D	E
02	A	B	C	D	E
03	A	B	C	D	E
04	A	B	C	D	E
05	A	B	C	D	E
06	A	B	C	D	E
07	A	B	C	D	E
08	A	B	C	D	E
09	A	B	C	D	E
10	A	B	C	D	E
11	A	B	C	D	E
12	A	B	C	D	E
13	A	B	C	D	E

14	A	B	C	D	E
15	A	B	C	D	E
16	A	B	C	D	E
17	A	B	C	D	E
18	A	B	C	D	E
19	A	B	C	D	E
20	A	B	C	D	E
21	A	B	C	D	E
22	A	B	C	D	E
23	A	B	C	D	E
24	A	B	C	D	E
25	A	B	C	D	E
26	A	B	C	D	E

27	A	B	C	D	E
28	A	B	C	D	E
29	A	B	C	D	E
30	A	B	C	D	E
31	A	B	C	D	E
32	A	B	C	D	E
33	A	B	C	D	E
34	A	B	C	D	E
35	A	B	C	D	E
36	A	B	C	D	E
37	A	B	C	D	E
38	A	B	C	D	E
39	A	B	C	D	E

PRACTICE QUESTIONS

OutSystems

ASODS

Architecture Specialist (ODC) (ES)

EDITION

Demo

QUESTIONS

9

FORMAT

Limited Edition PDF

Before you begin

What this is

9 practice questions for OutSystems **ASODS**, each with its answer key and an explanation. Answer a question before you read its key — the explanation is worth more once you have committed.

If something looks wrong

Send us three things and we will check it:

- the exam code, **ASODS**
- the question number
- the email address on your order

Write to **support@mometrix.net**. We reply within one business day.

QUESTION 1

MULTIPLE CHOICE

An Architecture Specialist is reviewing a high-traffic OutSystems application and needs to address performance and scalability concerns. The team has identified that expensive database aggregations are being recomputed on every request for a heavily viewed dashboard page, and the page is being served from multiple front-end servers behind a load balancer. Which two 4C's levers should the architect prioritize to reduce database load and improve response times? (Choose two.)

- ☒ **A** Introduce an in-memory cache for the aggregated dashboard data with an appropriate TTL
- ☐ **B** Use asynchronous background processing to offload work from the request thread
- ☒ **C** Enable an Output Caching / Page Cache strategy for the dashboard screen
- ☐ **D** Replace the synchronous aggregate query with a stored procedure in the external database
- ☐ **E** Refactor the screen using a more efficient reactive expression

ANSWER A - C

WHY Both in-memory data caching and page/output caching are explicit 4C's levers in the OutSystems Architecture Best Practices. Caching the aggregated data (Cache) reduces repeated expensive database calls, while page/output caching avoids regenerating the screen for identical requests, which is especially effective when front-end servers share or replicate cache. Offloading to async, pushing logic to a stored procedure, or micro-optimizing reactive expressions are not framed as 4C's levers.

QUESTION 2

MULTIPLE CHOICE

A team is implementing an asynchronous process in OutSystems to process customer orders. The architect wants to ensure the design follows OutSystems best practices for asynchronous work. Which two of the following statements are correct best-practice recommendations? (Choose two.)

- ☒ **A** Use Process entities (BPT) when the business process is long-running and human-driven, and use Timers plus lightweight mechanisms for short-running system tasks
- ☐ **B** Always use a single Timer per application to keep the architecture simple
- ☒ **C** Make asynchronous handlers idempotent so that retries do not corrupt data
- ☐ **D** Have asynchronous handlers perform large database reads on every retry to validate state
- ☐ **E** Use the Bulk Processing pattern inside queues when a high volume of small, independent items must be processed efficiently

ANSWER A - C

WHY OutSystems architecture best practices explicitly recommend using Process entities (BPT) for long-running, human-driven flows and Timers / lightweight async mechanisms for short-running system tasks. They also emphasize idempotency of async handlers to make retries safe. Using a single Timer for everything, doing heavy reads on every retry, and using bulk processing for every queue are not correct best practices.

QUESTION 3 SINGLE CHOICE

An enterprise customer wants to deploy an OutSystems application on premises due to strict data-residency and regulatory requirements. Which deployment option best fits this scenario?

- ☐ A OutSystems Cloud (OOC)
- ☒ B Self-managed / on-premises (OutSystems Platform Server) infrastructure
- ☐ C A third-party public SaaS hosting provider
- ☐ D An OutSystems personal environment only

ANSWER B

WHY When customers must keep data within their own data centers or a specific jurisdiction, the recommended OutSystems option is self-managed (on-premises) infrastructure, because OutSystems Cloud is hosted in OutSystems-managed regions. A personal environment is for developer learning, and a third-party public SaaS is not a standard OutSystems deployment model.

QUESTION 4 SINGLE CHOICE

Which of the following is the recommended OutSystems best practice for handling service credentials (for example, an external API key) used inside an OutSystems application?

- ☐ A Hardcode the key directly into the OutSystems logic so it is always available
- ☒ B Store the key in a Site Property and use the built-in encryption for sensitive values
- ☐ C Commit the key into the module source code in Service Studio and rely on the LifeTime deployment process to hide it
- ☐ D Email the key to all developers so they can paste it wherever it is needed

ANSWER B

WHY OutSystems best practice is to store sensitive configuration such as credentials in Site Properties and to mark sensitive values so they are encrypted at rest and not exposed in clear text. Hardcoding keys, putting them in module source, or distributing them by email are explicitly discouraged.

QUESTION 5

SINGLE CHOICE

An OutSystems e-commerce application experiences intermittent timeouts during marketing campaigns. Analysis shows that long-running database operations executed from screen preparations are blocking the request thread. Which 4C's lever should the architect apply first?

- ☒ **A** Coupling - decouple the UI from long operations using asynchronous processing or queues
- ☐ **B** Cache - add an in-memory cache of the catalog data
- ☐ **C** Caching - cache the rendered HTML of the page
- ☐ **D** Concurrency - raise the timeout values on the database connection

ANSWER A

WHY OutSystems architecture guidance frames Coupling as decoupling long-running work from the user request using asynchronous patterns / queues so the request thread is not blocked. Tuning timeouts does not address the root cause, and caching catalog data or HTML pages does not help if the underlying synchronous operation still blocks. Concurrency in the 4C's context refers to designing for concurrent users and parallelism, not increasing timeouts.

QUESTION 6

SINGLE CHOICE

In an OutSystems application, a developer needs to update an Order entity and several related OrderLine entities atomically. Which is the recommended best practice?

- ☒ **A** Wrap the updates in an explicit database transaction using the Transaction action so all changes commit or roll back together
- ☐ **B** Use multiple separate Advanced SQL statements without a transaction, relying on the database to be consistent
- ☐ **C** Disable all database constraints before updating and re-enable them after the update completes
- ☐ **D** Update each entity in a separate Timer for fault tolerance

ANSWER A

WHY OutSystems best practice recommends grouping related writes that must succeed or fail together inside an explicit Transaction, so that the platform manages commit and rollback. Disabling constraints or running unrelated updates in separate Timers breaks atomicity and is not a supported pattern.

QUESTION 7

SINGLE CHOICE

A team uses OutSystems LifeTime to manage application promotion. A developer has just finished a new version of an application in the Development environment and wants QA to start testing. What is the recommended way to move the application forward?

- ☐ A Export the module binary from Service Studio and email it to QA for manual import
- ☒ B Use LifeTime to deploy the new application version from Development to QA, following the standard DEV -> QA -> PROD pipeline
- ☐ C Copy the database directly and ask QA to attach it to a fresh environment
- ☐ D Re-publish the application in the QA environment from the developer's personal area

ANSWER B

WHY LifeTime is the standard OutSystems mechanism for promoting applications across environments along a controlled pipeline (typically DEV -> QA -> PROD). Manual binary exchanges, copying databases, or re-publishing from a personal area bypass governance, traceability and dependency management and are not recommended.

QUESTION 8

SINGLE CHOICE

An OutSystems application exposes a REST API that will be consumed by an external partner system. Which authentication/authorization approach is aligned with OutSystems best practices for exposing APIs?

- ☐ **A** Configure the API to be completely anonymous and rely on network-level IP filtering only
- ☒ **B** Implement an authentication flow that validates the partner's credentials (for example, Basic over TLS, or a token issued by the OutSystems platform) and authorize per endpoint, using HTTPS
- ☐ **C** Hardcode a shared secret in the URL query string of every API call
- ☐ **D** Disable all auditing and logging so partner calls remain hidden

ANSWER B

WHY OutSystems best practices for exposed APIs require proper authentication of the calling system (such as Basic over HTTPS or an issued token), authorization checks per endpoint, and transport security (TLS). Anonymous APIs, secrets in URLs, and disabled auditing all violate security best practices.

QUESTION 9

SINGLE CHOICE

After a production incident in an OutSystems application, the operations team needs to correlate user actions with backend errors and slow requests. Which source of information is the most appropriate first place to look for detailed error and performance diagnostics per request?

- ☐ A The application's Service Studio design-time logs
- ☒ B The platform's built-in logging (Service Center / LifeTime logs) and monitoring of error and request logs
- ☐ C The end users' web browser history
- ☐ D The source code repository commit messages

ANSWER B

WHY OutSystems environments expose error and request logs through Service Center (and aggregate monitoring through LifeTime / ODC monitoring tools), which is the correct operational source for runtime diagnostics. Service Studio logs are design-time only, and browser history or commit messages do not provide per-request error telemetry.