PassQueen
a.k.a dumpsqueen.com

Architecture Specialist (OutSystems 11) Exam

OutSystems Architecture-Specialist-11

Version Demo

Total Demo Questions: 7

Total Premium Questions: 72

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture Fundamentals	15
Topic 2, Application Architecture	51
Topic 3, Architecture Governance	6
Total	72

QUESTION NO: 1

Which of the below matches the most to Core Module Pattern - ECS with Local Replica Pattern...

A. ... Entity is not in Outsystems but in an external ERP system. IS just makes remote call to p external system/database. No data is being kept inside OS. Data retrieval may not be optimized as it needs to traverse two different systems to get the information back. Con: Integration API must support all use cases

B. ... is a pattern with two modules, a connector module that can be used to encapsulate an O external API with the input/output structures and a wrapper module to expose the normalized API to the consumers.p Same as ECS with local replica but synchronization logic is separated. Pro: Codeindependence. Consumers of CS is not affected by Sync. Sync can orchestrate several CS

C. ... a wrapper used to contain the logic, actions and data that will expose code that is inside of external library or to inspect external database and import the data structures so they can be used as entities inside of OS

D. ... caches only summary data that is frequently lister, joined or searched. Full detail for a single entry is fetched directly from external system. Use when whole database too big or costly to synchronize. Details are only required for single entities (not lists)

E. Entity is exposed as read-only and an API is available to centralize business logic for entity creation and update. Same as the Base ECS pattern, but with a local replica that stores data for local use. Pro: Leverage Entity usage and a simpler Integration API. Con: Less impact on the source system.

F. ... is needed if data is coming from MULTIPLE external systems. IS will decide which driver to use depending on the data.

G. Same as ECS with local replica but API module is provided. So any changes to the external system can notify OS, which OS then gets update from the ERP system (subscription system)

ANSWER: E

Explanation:

“Entity is exposed as read-only and API is available to centralize business logic for entity creation/update. Same as Base ECS pattern, but have a local replica. Store data to serve as a local cache. Pro: Leverage Entity Use, Simpler Integration API. Con: Less impact on source system” best describes the ECS with Local Replica pattern. In this pattern, the external core system remains the system of record, while OutSystems maintains a local copy of the external entity data. Applications consume the local entity rather than making a remote request for every read, allowing standard Entity operations, efficient queries, joins, aggregates, and a more stable normalized integration surface for consumers. The replica is typically treated as read-only by consuming modules; changes that affect the authoritative external data are centralized through the Core Service API and synchronized with the external system. This separation is valuable when external data is broadly used by multiple applications or screens and remote access would otherwise create latency, availability, or load concerns. Synchronization must be designed according to the required freshness and consistency of the data, but the defining characteristic is the maintained local replica of data owned externally. OutSystems architecture guidance emphasizes encapsulating external-system access and exposing stable service APIs; see the OutSystems service documentation and Architecture Dashboard documentation.

QUESTION NO: 2

In which of the following scenarios should you choose to clone a built-in Style Guide?

A. When minor customizations to the base theme should be done inside the app theme.

B. It is not possible to customize a built-in Style Guide, since it is part of OutSystems UI.

C. When you want to introduce extensive changes to an existing theme.

D. When it is not possible to benefit from any existing theme and extensive changes are needed.

ANSWER: C

Explanation:

When you want to introduce extensive changes to an existing theme, cloning the built-in Style Guide is the appropriate approach. A Style Guide provides the reusable visual building blocks for an application, including screen patterns, UI elements, typography, colors, and usage examples. Cloning creates an application-owned version that developers can adapt substantially without directly modifying the OutSystems UI version supplied by the platform. This is particularly valuable when the intended design system departs broadly from the standard patterns or requires pervasive changes to styles and components across the application.

The cloned Style Guide can then evolve alongside the application's own theme and design requirements, providing a controlled baseline for implementing the organization's visual language consistently. Because the clone is owned and maintained by the application team, it also avoids relying on changes made to the original built-in asset. OutSystems recommends using themes to customize an application's appearance and using the Style Guide as the reference and starting point for UI patterns; cloning is suited to deeper, broader customization needs. See the OutSystems UI component overview and OutSystems UI documentation.

QUESTION NO: 3

Which of the following options denotes the advantages of defining a Style Guide up front?

A. Security and scalability.

B. Speed up the development phase.

C. Improve performance and maintainability.

D. Allows apps and the Style Guide to be deployed to Production.

ANSWER: B

Explanation:

Defining a Style Guide up front helps **speed up the development phase** by giving teams an agreed, reusable set of visual rules, UI patterns, and component usage guidelines before screens are built. Developers do not need to repeatedly decide how common interface elements should look or behave, and can instead apply established styles and patterns consistently. This reduces design and implementation rework, makes handoffs between team members smoother, and allows new developers to become productive faster. In an OutSystems factory, a shared Style Guide also promotes reuse of UI assets and a consistent user experience across applications, which makes delivery more predictable as the application portfolio grows. OutSystems UI provides reusable interface patterns and a mechanism for applying a common theme, supporting this standardized and accelerated approach to front-end development. See the OutSystems UI documentation and the OutSystems UI Style Guide documentation.

QUESTION NO: 4

Themes and Layouts are an important element of an application. In which of the following would you define these elements?

A. MyApp_CS Module, in the Foundation layer

B. MyApp_BL, in the Foundation Layer.

C. MyApp_Th Module, in the End-User layer

D. MyApp_MTh, in the Foundation layer.

ANSWER: C

Explanation:

MyApp_Th Module, in the End-User layer is correct because themes and layouts define the application's visual presentation and common user-interface structure. A theme centralizes the CSS styling applied throughout the application, while layouts provide reusable page scaffolding, such as headers, navigation, menus, and content areas. These artifacts are

consumed directly by screens and blocks, so they belong in the End-User layer, which contains the UI elements presented to users.

Keeping themes and layouts in a dedicated End-User module creates a clear, reusable UI foundation for all application front ends. It also supports separation of concerns: application logic and integrations remain independent of presentation decisions, while styling and shared page structure can be maintained consistently in one place. This structure makes it easier to evolve the application's look and feel without unnecessarily coupling it to core or business-service modules. OutSystems' architecture guidance organizes applications into layers according to their responsibilities, and reusable UI resources belong with the end-user-facing components. See the OutSystems Architecture Dashboard documentation and the guidance on maintainable CSS with themes.

QUESTION NO: 5

An OutSystems application must display the current account balance held in an external banking system. Regulatory requirements state that every display must use the authoritative balance at the time of the request, and the application does not need to search or join balances locally. Which Core Module pattern is the best fit?

A. ECS with Local Replica

B. ECS Lazy Load variation

C. ECS with Direct Integration

D. ECS with Isolated Synchronization Logic

ANSWER: C

Explanation:

ECS with Direct Integration is the best fit because the balance remains in the external system and is retrieved when required. The requirement for authoritative real-time information makes a local replica or cache unsuitable, and no local list-processing requirement justifies synchronization.

An ECS with Local Replica improves local querying and reduces calls to the source system, but it introduces data-freshness considerations. Lazy Load is useful when selectively caching records, which conflicts with the requirement to obtain the current value from the system of record for every display.

QUESTION NO: 6

What does NOT happen due to a lack of architecture concerns?

A. Inflexible and slow-moving legacy systems : adapting legacy systems to business changes may be difficult. Changes in complex and inflexible systems can take a long time

B. Tech Debt : AI Mentor will raise architectural tech debt such as cyclic dependency and side to side dependency

C. Unmanageable dependencies : System not isolated from each other. Updating or replacing a system has a cascade/snowball effect on other systems

D. Poor service abstraction : Business concepts not correctly isolated, business rules tend to be spread over different systems and little to no code reuse

ANSWER: B

Explanation:

Tech Debt : AI Mentor will raise architectural tech debt such as cyclic dependency and side to side dependency is the correct choice because it describes an AI Mentor governance and detection activity, rather than a direct operational or maintainability consequence that occurs in the application landscape when architecture concerns are neglected. AI Mentor can analyze an OutSystems factory and identify architecture issues, including dependency-related patterns, so that teams

can prioritize remediation. The fact that AI Mentor reports a finding is an outcome of running automated analysis; it is not itself the architectural degradation caused by failing to consider architecture.

Architecture concerns are intended to guide how systems are decomposed, how responsibilities are isolated, and how dependencies are managed as the factory evolves. AI Mentor Architecture Dashboard supports that discipline by providing architectural analysis and actionable findings that help teams monitor and improve the health of their applications and inter-system relationships. In other words, AI Mentor is a supporting control used to surface and manage potential technical debt, not an inherent effect that a lack of architectural attention produces. See OutSystems' AI Mentor Systems documentation and the OutSystems architecture best-practices documentation.

QUESTION NO: 7

A mobile application has a long inspection workflow. Inspectors must be able to complete several steps without connectivity, and only the final submission requires an immediate server confirmation. The current implementation synchronizes whenever the user moves to another screen. Which change is most appropriate?

A. Synchronize each time the user navigates to another screen.

B. Use local storage during the workflow and synchronize at deliberate business points.

C. Call the external inspection system directly from the mobile client for each step.

D. Disable local storage so that each screen always shows server data.

ANSWER: B

Explanation:

Use local storage during the workflow and synchronize at deliberate business points is correct. The inspection data should support the offline use case locally, with synchronization aligned to meaningful events such as starting the process, restoring connectivity when required, or submitting the completed inspection.

Synchronizing on every screen transition creates unnecessary server calls, battery and bandwidth use, and a less reliable experience on intermittent networks. Direct calls from the mobile client to external backend systems would also bypass the controlled OutSystems Core Service boundary.