DUMPSBOSS.

Appian Senior Developer

Appian ACD201

Version Demo

Total Demo Questions: 10

Total Premium Questions: 106

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Application Design	12
Topic 2, Data Management	24
Topic 3, User Experience	10
Topic 4, Process Modeling	9
Topic 5, Integrations	13
Topic 6, Security	12
Topic 7, Performance	25
Topic 8, Mix Questions	1
Total	106

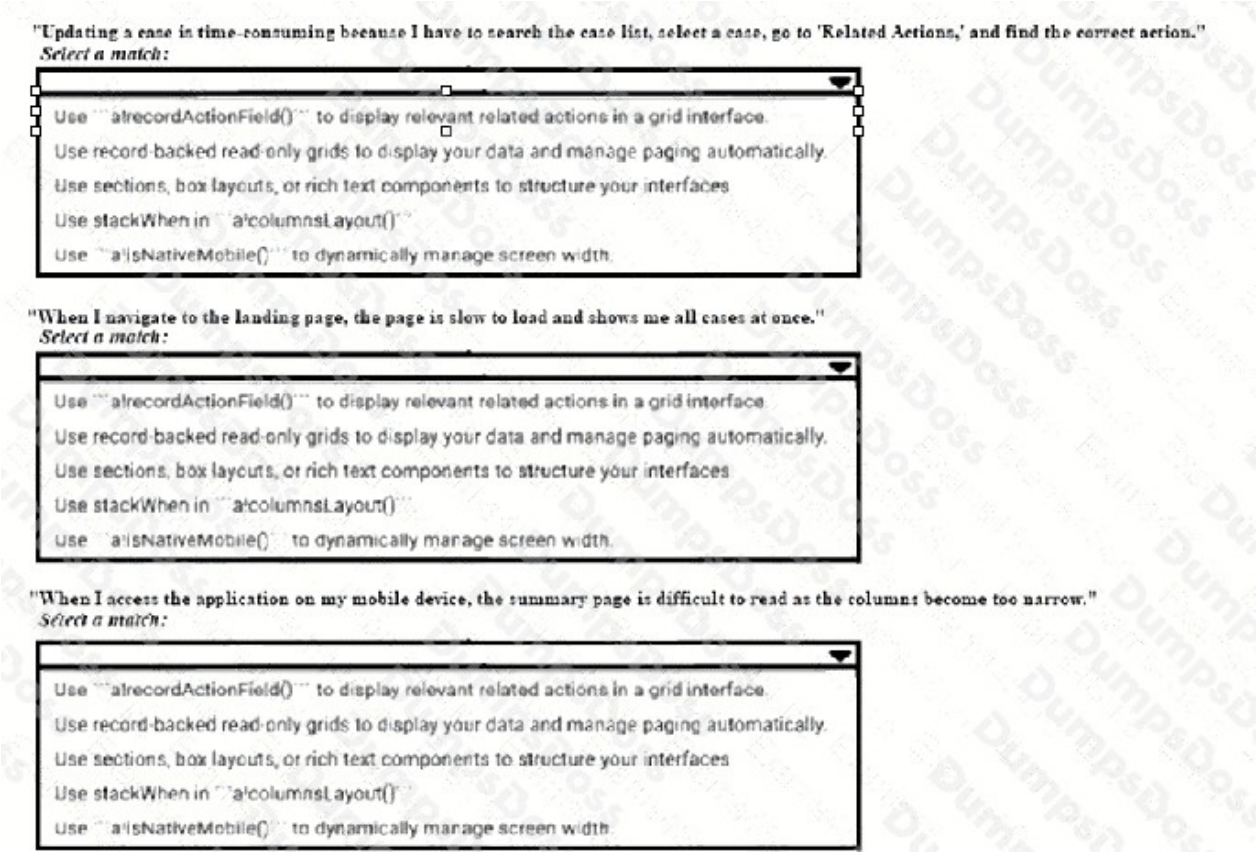
QUESTION NO: 1 - (SIMULATION)

You're developing a case management application. Currently, users can view a list of all cases designed using a a!forEach() loop from a record query. Users can navigate to the case summary page which shows the case details in a two (2) column layout. Users are also able to update the case details.

During the last round of UAT testing, users reported that the tool was *not* intuitive to use.

Match each feedback comment to the suggested UI/UX improvements.

Note: Each UI/UX improvement will be used once, or not at all. To change your responses, you may deselect your response by clicking the blank space at the top of the selection list.



ANSWER: See the explanation for the answer

Explanation:

Match the feedback to these UI/UX improvements:

"Updating a case is time-consuming because I have to search the case list, select a case, go to 'Related Actions,' and find the correct action."
Use `a!recordActionField()` to display relevant related actions in a grid interface.

"When I navigate to the landing page, the page is slow to load and shows me all cases at once."
Use **record-backed read-only grids** to display the data and manage paging automatically.

"When I access the application on my mobile device, the summary page is difficult to read as the columns become too narrow."
Use `stackWhen` in `a!columnsLayout()` so the two columns stack appropriately on narrower screens.

The remaining choices, such as using sections/box layouts/rich text for structure or `a!isNativeMobile()`, do not directly address the reported usability issues.

QUESTION NO: 2 - (SIMULATION)

Match each authentication type to the correct authentication characteristic description.

Note: Each description will be used once. To change your responses, you may deselect your response by clicking the blank space at the top of the selection list.

ANSWER: See the explanation for the answer

Explanation:

The matching options are not included in the supplied simulation JSON. For the standard Appian connected-system authentication types, use the following matches:

Basic Authentication — Sends a *username and password*, typically as a Base64-encoded value in the HTTP `Authorization` header.
API Key Authentication — Sends a static *API key/token*, usually in a request header or query parameter.
OAuth 2.0 — Obtains and uses an *access token* from an authorization server, allowing authorization without sending the user's password to the target API.
Custom Authentication — Uses API-specific authentication logic or request configuration when Basic, API Key, or OAuth 2.0 does not meet the service's requirements.

If the displayed simulation instead lists user-authentication providers, the corresponding standard matches are: **Appian authentication** = credentials maintained in Appian; **LDAP** = credentials validated against a directory server; **SAML** = XML-based federated single sign-on through an identity provider; and **OpenID Connect** = OAuth 2.0-based identity authentication using ID tokens.

QUESTION NO: 3

You are creating an expression rule that determines whether a loan application can proceed to underwriting. The rule will be reused by a start form, a record action, and several process models.

Which two test-case practices best protect the rule from future regressions? (Choose two.)

A. Create test cases for both eligible and ineligible application scenarios.

B. Assert only that the expression rule completes without errors.

C. Use live production records as the expected output for every test case.

D. Use an assertion expression to verify the expected eligibility decision or status in the returned result.

ANSWER: A D

Explanation:

Test cases should cover representative business outcomes, including a valid application that is eligible and an application that fails a required eligibility condition. These tests establish the rule's intended behavioral contract and can be rerun when the rule is changed or when dependent application objects are deployed.

Where the rule returns a complex result, an assertion expression that evaluates a specific business condition is preferable to treating successful evaluation as proof of correctness. A rule can complete without errors while returning an incorrect eligibility decision. Tests should use predictable inputs and expected outcomes rather than relying on changing production data.

QUESTION NO: 4

You need to configure a process model that runs every day at 10:00 AM. The process should start only at the specified time even when a newer version of the process model is published.

A Boolean type constant needs to be configured in the process model. The intended flow of the process should be executed only when the constant is True.

Which three steps should you perform? (Choose three.)

A. Add an expression-based condition in the start node timer event to check for the constant to be True.

B. Add a timer event in the start node.

C. Add a timer event just after the start node to check for the specific time.

D. Add a XOR Gateway just after the start node to check for the constant to be True.

E. Add an expression-based condition in the start node timer event to check for the specific time.

ANSWER: B D E

Explanation:

Adding a timer event in the start node establishes the process model's scheduled start behavior. The timer can be configured with a daily recurrence at 10:00 AM, so the process is launched by the schedule rather than by a user or external system. An expression-based condition in the start node timer event provides an additional guard that evaluates the required time condition before allowing the timer to create a process instance. This is important when publishing a newer process-model version, because the condition ensures that processing remains constrained to the intended scheduled time.

Adding a XOR Gateway just after the start node to check for the constant to be True separates the enabled and disabled paths explicitly. Configure the outgoing flow for the intended process path with a condition that evaluates the Boolean constant. When the constant is true, the gateway routes the instance into the business flow; this makes the constant a controlled feature flag for the scheduled process behavior. Together, the start timer, its time-based condition, and the gateway provide scheduling plus conditional execution. See Appian's Timer Event documentation and Gateway documentation.

QUESTION NO: 5 - (DRAG DROP)

You need to create a new database schema using a connected data source system.

What should you do?

Note: To answer, move all steps from the Options list to the Answer List area and arrange them in the correct order.

ANSWER:

Explanation:

Creating a database schema through a connected data source setup starts by collecting the connection information that identifies and authenticates access to the database. Copying the credentials and connection URL first ensures the later configuration can point to the intended MariaDB environment with the appropriate access details.

Next, create a separate schema for each application, or for each suite of applications that should share a database boundary. Using the `AppianProcess.createNewSchema()` stored procedure provisions those schemas according to Appian's supported database-management process. This establishes the database containers before any Appian integration is configured to use them.

Once each schema exists, create a MariaDB Data Source connected system for each schema using the copied credentials. A connected system centrally stores the connection configuration Appian needs so that data-source objects and integrations can access the corresponding schema consistently. Keeping a connected system aligned to each schema also supports clean separation between applications and their database resources.

Finally, assign users to the Database Editors group when they need to make schema changes through phpMyAdmin, and assign users to the Database Viewers group when they only need read-only visibility. Applying these permissions after the schemas and connections are established provides the appropriate operational access while preserving controlled administration. Appian's product documentation is available through the Appian Documentation portal, and additional platform guidance is available from the Appian Community.

QUESTION NO: 6

You're reviewing the data store performance logs. You notice several items were logged in the generated slow query log file: perf_monitor_rdbms_slow.csv.

What is the default threshold value for a slow query to be included in this log file?

A. 5 sec

B. 7 sec

C. 3 sec

ANSWER: C

Explanation:

3 sec is the default slow-query threshold for the `perf_monitor_rdbms_slow.csv` performance-monitoring log. Appian records database queries in this file when their execution duration reaches or exceeds the configured slow-query threshold; with the standard default configuration, that threshold is three seconds. This log is intended to help administrators and developers identify data-store interactions that may be contributing to slow interfaces, process execution, or overall system performance concerns. Reviewing the entries provides useful context such as the query activity and timing, which can then be correlated with application behavior and investigated for opportunities to improve data access patterns, indexing, query design, or data-store-object usage. The threshold is a monitoring mechanism rather than an indication that every recorded query is necessarily defective; however, queries meeting the threshold merit review because repeated multi-second database activity can materially affect user experience and throughput. Appian's official documentation describes its performance-monitoring and logging capabilities, including the generated performance log files, at Appian Performance Monitoring. Related administration guidance is available in the Appian Logging documentation.

QUESTION NO: 7

You ' re conducting a code review and notice that some accessibility features are missing on the interfaces.

Which two UX best practices should be followed to align with accessibility requirements? (Choose two.)

A. Use a single rich text icon with " POSITIVE " or " NEGATIVE " styling to indicate statuses.

B. Apply a label value for all fields when the label position is " Hidden " .

C. Indicate the hierarchy of sections with a heading tag in addition to label sizes (for example: " H1 ").

D. Use rich text items to act as headers for structuring interfaces.

ANSWER: B C

Explanation:

Applying a label value for all fields when the label position is " Hidden " is an accessibility requirement because hiding a label visually does not remove the need for an accessible name. The label supplies meaningful context to screen readers and other assistive technology, allowing a user to understand the purpose of an input control without relying on surrounding visual layout or placeholder text. This is particularly important when fields are presented compactly but still need to remain understandable to nonvisual users.

Indicating the hierarchy of sections with a heading tag in addition to label sizes creates a semantic structure that assistive technologies can expose to users. Heading levels communicate how content is organized and let screen-reader users navigate efficiently between major sections and subsections. Visual sizing alone conveys hierarchy only through appearance; applying the appropriate heading tag makes that hierarchy programmatically available. Together, accessible field labels and semantic headings support Appian interfaces that are understandable, navigable, and aligned with accessible UX practices. See Appian's Accessibility documentation and the WCAG guidance on headings and labels.

QUESTION NO: 8

You are building a fulfilment process that sends a POST request to an external shipping system after an order is approved.

The shipping request can fail temporarily, and the order must not move to "Shipped" unless the external system returns a successful response. Operations staff must be able to monitor and resolve failed requests.

What is the best implementation approach?

A. Call the integration from a process model by using the Call Integration smart service, then route based on the response.

B. Call the POST integration in the saveInto parameter of the order approval interface.

C. Update the order status to "Shipped" before calling the integration, then log any error for later review.

D. Call the integration from an expression rule and discard the response after the rule evaluates.

ANSWER: A

Explanation:

Call the integration from a process model by using the Call Integration smart service, then route the process according to the response. A process provides a durable execution context for the external write operation, including monitoring, error handling, retries where appropriate, and subsequent routing based on whether the request succeeded.

An interface `saveInto` should not be used to invoke an integration that changes external source data. Calling the integration only from an expression rule does not provide the required operational workflow, and updating the order status before receiving a successful response would create an inaccurate business state.

QUESTION NO: 9

You are designing the data model for an invoice application.

An invoice can contain many products, and a product can appear on many invoices. Each occurrence of a product on an invoice must retain the quantity ordered and the unit price at the time the invoice was issued.

Which data model is most appropriate?

A. Create Invoice and Product tables, with a list of product identifiers stored on each Invoice record.

B. Create Invoice, Product, and Invoice Line tables, with quantity and unit price stored on Invoice Line.

C. Create Invoice and Product tables, with quantity and unit price stored on Product.

D. Create a single Invoice table with one set of product, quantity, and price columns for each possible line item.

ANSWER: B

Explanation:

Create separate Invoice, Product, and Invoice Line tables. The Invoice Line table is the junction entity that resolves the many-to-many relationship between invoices and products. It should contain foreign keys to Invoice and Product, as well as attributes that belong to the specific invoice-product association, such as quantity and unit price.

Storing product identifiers or repeated product details directly on the Invoice table would not support multiple line items cleanly. Storing a current product price only on Product would also fail to preserve the price that applied when a historical invoice was issued.

QUESTION NO: 10

There are two record types, ABC and XYZ, with sync enabled. The XYZ record type is added as a relationship into the ABC record type.

A user has Viewer permission to the ABC record type but does not have access to the XYZ record type.

A site page is presented to the user where the data is sourced from the ABC record type and its related record type the XYZ reference.

What information does the user see on the site page?

A. Page is presented to the user and the data references to the XYZ record type appear as "null".

B. Page is presented to the user with the XYZ record type data and fields references redacted.

C. Page does not load and an error message is presented: "The record type [identifier=XYZ] does not exist, has been deleted, or you do not have sufficient privileges to access its data."

ANSWER: C

Explanation:

Page does not load and an error message is presented: "The record type [identifier=XYZ] does not exist, has been deleted, or you do not have sufficient privileges to access its data." Appian record-type security is enforced whenever record data is retrieved, including when an interface accesses data through a record relationship. Viewer permission on the ABC record type permits the user to view ABC records, but it does not grant any permission to the related XYZ record type. Record-type access must be independently granted for every record type whose data is used by a page.

Because the page's data source includes the XYZ relationship reference, Appian attempts to resolve and retrieve the related XYZ record data. The user has no access to that record type, so the request cannot successfully evaluate. Appian therefore returns its standard record-type access error rather than presenting the related data. This behavior protects record data consistently across record views, sites, interfaces, and relationship-based data retrieval. Security should be configured so that site audiences have the required record-type permissions, or the page logic should avoid evaluating relationships to record types that the audience cannot access. See Appian's Record Type Security documentation and Record Type Relationships documentation.