DUMPSARENA

Associate Reactive Developer (OutSystems 11) Exam

OutSystems Associate-Reactive-Developer

Version Demo

Total Demo Questions: 18

Total Premium Questions: 182

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1

If an Entity Attribute named HouseNumber is created, what needs to be done about its Data Type?

A. Nothing, it will automatically be set to Integer.

B. It should be set to Decimal.

C. It should be set to Integer.

D. Nothing, it will automatically be set to Identifier.

ANSWER: A

Explanation:

Nothing, it will automatically be set to Integer. Service Studio applies naming conventions when it creates entity attributes and can infer an appropriate default data type from a meaningful attribute name. In this case, the `Number` portion of `HouseNumber` is recognized as a numeric value, so the attribute is created with the Integer data type automatically. This behavior helps developers model common data consistently and reduces manual configuration for straightforward attribute names.

After creating the attribute, its properties remain editable in the Data tab. Therefore, a developer can still deliberately change the type if the business meaning requires something else, such as preserving leading zeroes or accepting nonnumeric house-number formats. However, for the question's stated creation scenario, no action is necessary because the initial inferred type is Integer. OutSystems entity attributes represent the fields persisted for each entity record, and selecting suitable types is part of defining the data model. See OutSystems training on entities and attributes and the OutSystems 11 documentation for data-modeling guidance.

QUESTION NO: 2

Ending a Screen Action with an End element or a Destination to the '(Current Screen)' yields the same result.

A. TRUE

B. FALSE

ANSWER: B

Explanation:

FALSE is correct because these elements have distinct behaviors in a Reactive Screen Action. An *End* element simply terminates the action flow. Control returns to the client-side runtime, and the current screen remains displayed without initiating navigation. This is appropriate when the action has completed its work and no screen transition or refresh through navigation is required.

A Destination set to *(Current Screen)*, however, is a navigation outcome. It explicitly navigates to the screen that is already open, causing the screen to be re-entered and its lifecycle behavior to run again. In particular, the screen's data-loading logic can be executed again, so aggregates and other data dependencies can be refreshed according to the screen's normal load process. Consequently, using *(Current Screen)* can be useful after an update when the developer intends to reload the screen state, whereas ending the action does not itself trigger that navigation-based reload.

This difference follows the distinction in OutSystems between ending a client action and using a destination for client-side navigation. See the OutSystems documentation on flow elements and client logic.

QUESTION NO: 3

A Screen contains a Data Action configured to fetch Only on Demand. The Data Action should load notification data when the user opens a Notifications panel. What should the panel-opening Client Action do?

A. Use Refresh Data for the Data Action.

B. Use Refresh Data only for the Notifications panel widget.

C. Use an Assign node to set the Data Action List.

D. Republish the Module when the panel is opened.

ANSWER: A

Explanation:

The Client Action should use **Refresh Data** for the Data Action. A Data Action configured as Only on Demand is not fetched automatically when the Screen is initialized, so it must be explicitly requested when its data is needed.

Refreshing only the Screen or a widget does not execute the Data Action. The Data Action itself must be refreshed so that its server-side logic is invoked and its result becomes available.

QUESTION NO: 4

A Theme defines the look and feel of application Screens and you can directly apply it to ...

A. A Module, a UI Flow, or an individual Screen or Web Block.

B. A Module, a UI Flow, or an individual Screen.

C. A Module or a UI Flow.

D. Only the Module.

ANSWER: B

Explanation:

A Module, a UI Flow, or an individual Screen. is correct because OutSystems supports theme assignment at each of these scopes, allowing developers to control styling inheritance and selectively override the visual design where needed. Applying a theme at the module level establishes the default look and feel for the module's user interface. A UI Flow can receive a theme directly when a group of related screens requires a different visual identity or a controlled variation of the module's default styling. An individual Screen can also be assigned a theme directly, which is useful when that specific screen needs a distinct design while retaining the same functional structure. This hierarchical approach enables reusable, centralized styling while giving developers appropriate flexibility for localized presentation requirements. Themes contain the CSS and design resources used to render Reactive Web application interfaces, and the selected theme determines the styling available to the screens under its scope. OutSystems documents theme management and UI styling in its UI development guidance: OutSystems UI component overview and OutSystems 11 documentation.

QUESTION NO: 5

A Web Block can be used ...

A. Inside Web Screens and Web Blocks, except on itself.

B. Inside Web Screens and Web Blocks, even on itself.

ANSWER: A

Explanation:

Inside Web Screens and Web Blocks, except on itself. is correct. In OutSystems Reactive Web Apps, a Web Block is a reusable UI component that can encapsulate layout, content, client actions, and input parameters. Developers can drag a Web Block from the Interface tab into a Web Screen, allowing common interface elements—such as headers, cards,

navigation sections, or form fragments—to be consistently reused. A Web Block can also be placed inside another Web Block, which supports composition of more complex reusable components from smaller ones.

However, a Web Block cannot include itself. Allowing a block to directly reference itself would create an endless rendering hierarchy: each instance would need to render another instance of the same block without reaching a terminating UI structure. OutSystems prevents this self-reference so that the screen's UI tree remains finite and can be validated, rendered, and maintained safely. This behavior aligns with the platform's component-based development model, where reuse is supported through screens and other blocks while circular component inclusion is avoided. See the OutSystems documentation on Web Blocks and building application user interfaces.

QUESTION NO: 6

Style load order

A. Screen & Email -> Web block -> theme

B. theme -> Web block -> Screen & Email

C. Web block -> theme -> Screen & Email

ANSWER: C

Explanation:

The correct style load order is **Web block -> theme -> Screen & Email**. OutSystems assembles the CSS required by the rendered interface in this sequence so that styling can be progressively specialized. Web Block CSS is loaded first, providing the component-level base styles used by reusable interface elements. Theme CSS is then loaded, enabling the application theme to consistently define or override visual rules across those components. Finally, the CSS defined directly on a Screen or Email is loaded last, allowing page-specific styling to take precedence when selectors have the same specificity. This ordering follows normal CSS cascade behavior: when declarations have equal importance and specificity, the declaration loaded later is applied. It gives developers a practical customization hierarchy, from reusable block styling, through application-wide theming, to targeted screen or email adjustments. For background on CSS's cascade and source-order behavior, see MDN: Handling conflicts. For OutSystems guidance on interface styling and themes, see OutSystems UI: Customizing the look and feel.

QUESTION NO: 7

Inside an Action flow...

A. ••• the Exception Handler flow cannot intersect other flows.

B. ••• only one Exception Handler may exist.

C. ••• it's mandatory to have at least one Exception Handler.

ANSWER: A

Explanation:

The statement "••• the Exception Handler flow cannot intersect other flows." is correct because an Exception Handler defines a separate execution path for handling an exception raised during an Action's execution. In OutSystems, this path starts at an Exception Handler node and must remain structurally independent from the normal flow and from other handler flows. Keeping handler paths separate makes the Action's control flow unambiguous: once an exception is caught, the platform follows the corresponding handler path rather than merging it into another path at an arbitrary point.

This modeling constraint also reflects the purpose of exception handling. A handler can perform recovery logic, log the error, prepare feedback for the caller, or raise an exception again, but its flow is an explicit exception-specific branch. The visual flow editor enforces valid flow structures so that an Exception Handler cannot connect back into, or intersect with, another existing flow. This makes it easier to understand where errors are handled and ensures predictable behavior at runtime. See the OutSystems documentation on Exception Handling and Actions.

QUESTION NO: 8

Select the correct option regarding the Input Widget.

A. All input widgets must be inside a Form widget.

B. You don ' t need to have a variable. The data typed by the user is saved in the Input widget itself.

C. It can only be associated to variables of Text data type.

D. You always need to have a variable to store the value typed by the user.

ANSWER: D

Explanation:

You always need to have a variable to store the value typed by the user. In OutSystems Reactive Web applications, an Input widget is a data-bound control: its *Variable* property identifies the value that the widget displays and updates. This binding can point to an appropriate value in the current screen context, such as a local variable, an input parameter, or an attribute of a record being edited. As the user changes the control's value, OutSystems updates the bound variable, making that value available to client actions, validations, and the logic that persists or otherwise processes the entered data.

The variable also establishes the expected data type for the widget and allows the platform to perform its normal type handling and validation behavior. The widget is therefore not an independent storage location for user-entered data; it is the user-interface element through which a bound value is read and edited. This data-binding model is central to building forms and interactive screens in Reactive Web, because subsequent client-side logic uses the updated bound value when an event occurs, such as a button click or change event. See the OutSystems documentation on Input Widgets and Forms.

QUESTION NO: 9

A Server Action calls an integration that can fail. When it fails, the application must set an output parameter with a friendly error message instead of propagating the exception to the caller. Which implementation is most appropriate?

A. Add an Exception Handler to the Server Action flow.

B. Add an If node after the integration call.

C. Use the Screen On Render event.

D. Add an On After Fetch action to the Server Action.

ANSWER: A

Explanation:

An Exception Handler catches exceptions raised during the Server Action flow. The handler can be configured to handle the relevant exception type, assign the friendly message to the output parameter, and finish the action through an End node. An If node cannot intercept an exception that has already been raised, and an On After Fetch event applies to client data actions rather than server-side exception handling.

QUESTION NO: 10

Considering the Function property in Client Actions, which of the following options is correct?

A. Setting the Function property to Yes restricts the Action to have only one Output Parameter.

B. Setting the Function property to No ensures the Action can only be used in the module where it is defined.

C. Setting the Function property to Yes is not possible, if the Action is exposed to other modules as Public.

D. Setting the Function property to No ensures the Action can only be used in Screen Expressions.

ANSWER: A

Explanation:

Setting the Function property to Yes restricts the Action to have only one Output Parameter. In OutSystems, marking a Client Action as a function makes it usable where an expression is expected, such as in expression editors and property values. To behave as an expression, the action must produce one unambiguous value; therefore, a function Client Action is limited to a single output parameter. That output represents the value returned by the function invocation.

This setting is useful for reusable client-side calculations, formatting helpers, validation predicates, and other logic that needs to return a value directly into an expression. The action can still receive input parameters, allowing the returned value to be calculated from supplied data. The key distinction is that the Function setting defines expression-style use and a single returned result, rather than determining the action's module visibility. Public visibility is controlled separately through the action's exposure settings. See OutSystems documentation on Client Actions and the OutSystems training material covering Client Actions.

QUESTION NO: 11

What types of applications can be created in OutSystems?

A. Web, Mobile and Service

B. Web, Mobile, Service and Extension

C. Module and Extension

D. Only Web

ANSWER: A

Explanation:

Web, Mobile and Service is correct because OutSystems supports creating applications for browser-based experiences, native-like mobile experiences, and service-oriented back ends. A Web application is used to deliver responsive user interfaces through a browser. A Mobile application is used to build mobile apps that can run on supported devices and can use device capabilities through the OutSystems mobile runtime. A Service application is intended for exposing and managing REST APIs and other back-end service functionality without requiring an end-user interface.

These application types reflect the main runtime and delivery models offered when creating an application in OutSystems. They also support a common architecture in which web and mobile front ends consume reusable server-side logic and APIs supplied by service applications. Developers can therefore use the same low-code platform to address user-facing web channels, mobile channels, and integration/API scenarios while organizing functionality according to its purpose. OutSystems documentation describes web and mobile application development and the platform's support for exposing REST APIs; see the OutSystems 11 documentation and the REST APIs documentation.

QUESTION NO: 12

A consumed REST API sometimes returns an HTTP error response with a useful error message in its response body. The developer needs to inspect that response before the normal output mapping is used. Which callback action should be used?

A. OnBeforeRequest

B. OnAfterResponse

C. On Initialize

D. On Parameters Changed

ANSWER: B

Explanation:

OnAfterResponse runs after the external service returns a response and is the suitable callback for inspecting or customizing the received response. It can be used when the application needs to examine response details such as headers, status information, or response content.

OnBeforeRequest is executed before the request is sent and is intended for preparing the outgoing request, such as setting a dynamic authorization header.

QUESTION NO: 13

Considering Aggregates and the SQL Tool, which of the following is the correct option?

A. The SQL Tool allows you to write queries that contain sub-queries.

B. Joins between entities can only be defined in Aggregates.

C. All queries that can be written in an SQL Tool can be defined in an Aggregate.

D. Attribute grouping can only be done with the SQL Tool.

ANSWER: A

Explanation:

The SQL Tool allows you to write queries that contain sub-queries. In OutSystems, the SQL Tool is intended for cases where an Aggregate does not provide the necessary query capability or where a developer needs to express more advanced SQL constructs directly. A sub-query is a query nested within another SQL statement, commonly used to calculate an intermediate result, filter rows according to a related result set, or derive a value that is then consumed by the outer query. This is valid within an Advanced SQL query, provided that the statement follows the database dialect supported by the target environment and that OutSystems entity references use the platform's SQL entity notation, such as `{Entity}` and `{Entity}.[Attribute]`.

The SQL Tool also supports input parameters and produces a typed record list based on the output structure defined for the query. This lets a Reactive application consume the results through normal server-side logic while retaining the flexibility needed for SQL patterns such as nested selects. OutSystems documents Advanced SQL as the mechanism for executing custom SQL statements when needed, including SQL features beyond the visual query composition offered by Aggregates. See OutSystems documentation on Advanced SQL queries and OutSystems documentation on using SQL queries.

QUESTION NO: 14

Regarding Table Records Widget , which of the following options is true?

A. Bound to a Source Record List

B. Bound to a Source List

C. Bound to a Source Record

ANSWER: A

Explanation:

Bound to a Source Record List is correct because the Table Records widget renders repeated rows from a collection of records, rather than from one individual record or an untyped list. Its *Source Record List* property receives a record list, commonly the `List` output of an Aggregate, SQL query, Data Action, or Server Action. The widget uses the structure of that record list to establish the row context, so expressions placed in table cells can access the current record's attributes and display one row for each item in the collection. This data-binding model also supports common table behavior such as headers, sorting integrations, and conditional presentation based on values in each row. For example, assigning an Aggregate result such as `GetEmployees.List` to the Source Record List enables the table to show employee data as

multiple rows. OutSystems documentation describes record lists as collections used to present repeated data and explains how screen data is obtained through Aggregates and actions. See the OutSystems documentation on Aggregates and the OutSystems 11 documentation.

QUESTION NO: 15

The Combo Box widget allows selecting one value out of possible alternatives in a drop-down list. Which of the following is NOT POSSIBLE

A. Use the Source Entity property to get the alternatives from an Entity or a Static Entity.

B. Use the Source Record List property to get the alternatives from a List of Record.

C. Use the Special List section to get the alternatives from an Entity or a Static Entity.

D. Use the Special List section to manually set special alternatives

ANSWER: C

Explanation:

Use the Special List section to get the alternatives from an Entity or a Static Entity is the operation that is not possible. In a Reactive Web Combo Box, entity-based alternatives are configured through the widget's source properties: the Source Entity property supplies records from an Entity or Static Entity, while Source Record List can supply a list already obtained or prepared in the screen logic. These sources define the data set from which the normal Combo Box choices are generated.

The Special List section has a different role. It is intended for explicitly configured special entries that supplement the regular data source, such as an empty or default-style choice. It is not a data-binding mechanism and cannot query, load, or derive alternatives from an Entity or Static Entity. Consequently, selecting an entity as the origin of alternatives belongs in the appropriate source configuration, not in Special List. This distinction lets a developer combine data-driven selections with deliberately defined special values while retaining a predictable value and caption mapping for the widget. See the official Combo Box reference and the Reactive Web widgets documentation for the Combo Box configuration model.

QUESTION NO: 16

In the Web Blocks section of the Theme properties, does changing the Layout web block mean changing the layout of existing screens?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. In a Reactive Web application, the Theme's *Layout* setting identifies the default layout web block Service Studio uses when it creates new screens. It is a creation-time default, rather than a global replacement that automatically changes every screen already in the application. Each existing screen has its own Layout property, which records the layout web block assigned to that specific screen. Therefore, modifying the Theme property does not reassociate existing screens with a different layout; developers must select and update the Layout property on the relevant screens when they intend to change those assignments.

This distinction is useful when an application needs more than one layout, such as a public layout for anonymous users and an authenticated layout with navigation. The Theme establishes the standard choice for newly created UI, while screens can explicitly use a different layout as needed. Changes made inside a layout web block itself can affect screens that already use that same block, because those screens reference it. However, changing which web block is configured as the Theme's default Layout does not alter the layout references of pre-existing screens. See the OutSystems documentation on themes and screens.

QUESTION NO: 17

A Client Action builds a Local Variable named SelectedProducts, with data type List of Product Record. During a For Each over GetProducts.List, the developer must add each selected Product to SelectedProducts. Which operation should be used?

A. ListClear

B. ListAppend

C. ListRemove

D. Assign

ANSWER: B

Explanation:

`ListAppend` is the operation that adds a record to the end of a list. The record passed to it must match the list element type, so `GetProducts.List.Current` can be appended to a List of Product Record while the For Each is iterating.

`ListClear` removes all items from a list. `ListRemove` removes an existing item, and an Assign does not append a record to a list collection.

QUESTION NO: 18

Given 1 Aggregate with data source is 3 entities:

A and B are related to With or Without

A and C are related to Only With. Asked how the result will be?

A. A may not have B but there must be C

B. A may not have B or C

C. A must have B and C

D. A Must have B or C

ANSWER: A

Explanation:

A may not have B but there must be C is correct. In an OutSystems Aggregate, the relationship setting *With or Without* preserves records from the source entity even when no matching record exists in the related entity. Therefore, an A record can be returned without a corresponding B record. In contrast, *Only With* requires a matching related record for the source record to be included in the Aggregate result. Since the relationship between A and C is configured as *Only With*, every returned A record must have an associated C record. Applied together, the Aggregate returns A records that have C, and it includes B data only when a matching B record is available. This behavior is equivalent to combining an optional relationship for B with a required relationship for C, while retaining the filtering effect of the required C relationship. Understanding these relationship types is important when designing Aggregates because they control both the records returned and whether related attributes can be empty. See OutSystems documentation on using data in Reactive Web Apps at OutSystems Documentation and the SQL join behavior underlying required and optional matches in the PostgreSQL JOIN documentation.