DUMPSBOSS.

Sitecore 10 .NET Developer Exam

Sitecore Sitecore-10-NET-Developer

Version Demo

Total Demo Questions: 7

Total Premium Questions: 76

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

QUESTION NO: 1

Which statements are true about shared, unversioned, and versioned Sitecore template fields? (Choose three.)

A. A shared field has one value for all languages and versions of an item.

B. An unversioned field has one value per language across all versions in that language.

C. A versioned field can have a different value for each language version.

D. A shared field stores a separate value for each language.

E. The field type automatically determines whether a field is shared or versioned.

ANSWER: A B C

Explanation:

A field's sharing setting determines how Sitecore stores field values across item languages and versions. A **shared** field has one value shared by all languages and versions of an item. An **unversioned** field has one value per language that is shared by all versions in that language. A **versioned** field can have a separate value for each language version of an item. These settings are configured on the template field and should be selected according to the content's language and versioning requirements. See Sitecore template fields documentation.

QUESTION NO: 2

You need to remove an existing configuration element by using an include-file patch. Which patch instruction should you use?

A. patch:delete

B. patch:remove

C. patch:disable

D. patch:clear

ANSWER: A

Explanation:

Use **patch:delete** to remove a matching configuration element from Sitecore's effective configuration. The instruction is applied in an include-file patch and is useful when a solution must disable or remove an inherited setting, processor, agent, or other configuration node. This approach preserves the original Sitecore configuration file, which supports safer maintenance and upgrades. The patch must identify the appropriate node in the configuration hierarchy so that Sitecore can remove the intended element during configuration merging. See Sitecore patch files documentation.

QUESTION NO: 3

Which statement explains the purpose of dynamic placeholders?

A. Unlike static placeholders, users can create as many dynamic placeholders on a layout as needed.

B. Dynamic placeholders with the same key can be used multiple times on a page while allowing content within the placeholder to be unique.

C. Dynamic placeholders allow users to override the placeholder from a page and directly replace it with a new one.

D. Dynamic placeholders allow the user to move the placeholder's content to any location on the page as necessary.

ANSWER: B

Explanation:

Dynamic placeholders with the same key can be used multiple times on a page while allowing content within the placeholder to be unique. This solves a common presentation-design requirement: a rendering, such as a reusable container, may occur more than once in a layout and each occurrence needs to accept its own independently selected child renderings and data sources. A conventional placeholder name identifies a single placeholder location, so repeated instances can conflict when Experience Editor users add or configure components. Dynamic placeholder support creates a distinct, contextual placeholder identity for each rendering instance while retaining the underlying placeholder key and its configured placeholder settings. Consequently, authors can use the same reusable layout structure several times on one page and manage the components placed inside each instance independently in Experience Editor. This is especially useful for repeatable rows, columns, tabs, cards, or other composite components whose internal regions must remain editable without sharing the same content definition. Sitecore documents the behavior and implementation approach for dynamic placeholders in its Dynamic placeholders documentation. Related presentation and placeholder configuration concepts are covered in the Placeholder Settings documentation.

QUESTION NO: 4

Which of the following information is available within Workbox?

A. Number of editable data source items in all of Sitecore

B. Information about an item's originating template

C. Number of active campaigns divided by marketing team

D. Available workflow commands divided by associated workflow

ANSWER: D

Explanation:

Available workflow commands divided by associated workflow is available in the Workbox. In Sitecore, Workbox is the interface used to manage items that participate in workflows. It organizes workflow-related work by workflow and state, allowing a user to see the items currently awaiting action in each applicable state. When an item is selected, Workbox presents the workflow commands that the current user is allowed to execute, such as submitting an item for approval, approving it, or rejecting it. The commands shown are determined by the item’s current workflow state and the security permissions assigned to the user for those commands. This workflow-centered organization lets content authors, reviewers, and approvers efficiently identify their pending work and move items through the configured approval process without opening each item individually in the Content Editor. Sitecore’s documentation describes Workbox as the application for viewing workflow items and executing available workflow commands, and explains that workflows define the states and commands used to control content progression. See the Sitecore Workbox documentation and the Sitecore workflows and Workbox overview.

QUESTION NO: 5

When creating your containerized Sitecore environment with Docker, for the ease of launching the containers, which two files are recommended? (Choose two.)

A. clean.ps1

B. docker-compose.yml

C. compose.yml

D. .env

E. docker.exe

ANSWER: B D

Explanation:

`docker-compose.yml` and `.env` are the recommended files for launching a containerized Sitecore development environment. The Docker Compose file defines the complete multi-container topology: Sitecore roles, supporting services such as SQL Server and Solr, images, networks, volumes, health checks, dependencies, and container configuration. This lets developers start the coordinated environment through a single Docker Compose command rather than manually creating and configuring each container.

The `.env` file supplies the environment-specific values consumed by Compose variable substitution. Sitecore container configurations commonly use it for values such as image tags, registry settings, hostnames, project names, passwords, and local configuration settings. Keeping these values separate from the Compose topology makes the Compose file reusable while allowing each developer or environment to provide appropriate local values. Together, these files provide the standard, repeatable launch mechanism used by Sitecore's Docker examples and container guidance. See the Sitecore documentation on containers in Sitecore development and Docker's reference for the .env file and Compose variable interpolation.

QUESTION NO: 6

Which statements are true about the Sitecore Event Queue? (Choose three.)

A. It stores events in the EventQueue database table.

B. It can propagate remote events between Sitecore instances.

C. It helps instances respond to events such as cache-clearing events.

D. It copies approved items from the Master database to the Web database.

E. It replaces the need to configure publishing targets.

ANSWER: A B C

Explanation:

The Sitecore Event Queue supports communication between Sitecore instances in a scaled environment. Events raised on one instance can be stored in the EventQueue database table and processed by other instances. This is particularly important for propagating remote events, such as events that cause caches to be cleared after publishing or item changes. The Event Queue does not copy content from Master to Web, which remains the responsibility of the publishing system, and it does not replace a shared database strategy or content serialization. See Sitecore events documentation.

QUESTION NO: 7

What is the purpose of developing field editor buttons for Experience Editor?

A. To allow Content Authors to edit image fields within Experience Editor.

B. To provide additional field-editing functionality for complex fields through a pop-up window.

C. To give Content Authors the ability to change the field type as they work on content.

D. To open the rich text editor for fields of the rich text type in Experience Editor.

ANSWER: B

Explanation:

To provide additional field-editing functionality for complex fields through a pop-up window is correct. In Experience Editor, a field editor button is an extensibility mechanism that places a command next to an editable field. When an author invokes the

button, Sitecore can open an appropriate dialog or custom pop-up editor for that field's value. This is particularly useful when editing requires a richer, purpose-built interface than direct inline text entry can provide, such as selecting items, configuring links, choosing media, or editing structured or otherwise complex field values.

The button is associated with field editing in the page-authoring context and can be implemented to invoke custom functionality while maintaining the Experience Editor workflow. Its purpose is therefore not limited to a single built-in field type or editor; it provides a framework for exposing specialized editing experiences where the field's data and authoring requirements warrant a dialog-based UI. Sitecore's Experience Editor documentation describes the editor as the page-based content-authoring interface and documents its extensibility model for supporting custom authoring behavior. See Sitecore Experience Editor developer documentation and Sitecore guidance for extending Experience Editor.