PassQueen
a.k.a dumpsqueen.com

Flutter Certified Application Developer

Android AFD-200

Version Demo

Total Demo Questions: 7

Total Premium Questions: 75

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

A Flutter developer writes automated widget tests for an application user interface.

Which of the following statements about Flutter widget testing are correct? (Select three)

A. testWidgets() can define a widget test.

B. pumpWidget() can build a widget tree in a test.

C. WidgetTester can simulate user interaction.

D. Widget tests can run only on a physical phone.

E. find.text() is used only for Firebase queries.

ANSWER: A B C

Explanation:

Flutter widget tests are commonly written with the `testWidgets()` function. The test callback receives a `WidgetTester`, which can build a widget tree using `pumpWidget()`, simulate user interactions, and locate widgets using finders such as `find.text()` and `find.byType()`.

Widget tests run in a test environment and do not normally require a physical Android or iOS device. They are useful for verifying widget rendering, user interaction, and changes to the widget tree. See the Flutter testing overview and the Flutter testWidgets API documentation.

QUESTION NO: 2

A Flutter app uses Cloud Firestore to store product information.

Which of the following statements about Cloud Firestore are correct? (Select three)

A. Data is stored in documents and collections.

B. A document can contain fields with application data.

C. Snapshot listeners can receive data updates.

D. All data must be stored in SQL tables.

E. It is used only to store image files.

ANSWER: A B C

Explanation:

Cloud Firestore stores data in documents that are organized into collections. A document contains fields, which can hold values such as strings, numbers, Boolean values, lists, maps, timestamps, and references. Flutter applications can also use snapshot listeners to receive updated data when matching Firestore documents change.

Cloud Firestore is a NoSQL document database. It does not use SQL tables as its primary data model, and Firebase Storage is a separate service intended for files such as images, videos, and documents. See the Cloud Firestore documentation and the Cloud Firestore data model documentation.

QUESTION NO: 3

A Flutter app uses an `AppBar` at the top of a `Scaffold` screen.

Which of the following properties can be used to configure an `AppBar`? (Select three)

A. title

B. leading

C. actions

D. body

E. floatingActionButton

ANSWER: A B C

Explanation:

An `AppBar` provides a Material Design application bar. Its `title` property displays the primary title widget, `leading` can display a widget at the start of the app bar, and `actions` accepts a list of widgets displayed after the title, often icon buttons for common screen actions.

The `body` property belongs to `Scaffold`, where the main content of a screen is placed. A `floatingActionButton` is also configured on a `Scaffold`, rather than directly on an `AppBar`. See the Flutter AppBar API documentation and the Flutter Scaffold API documentation.

QUESTION NO: 4

. Check the image of the app interface in this question.

Which type of Flutter widgets is used in designing this app interface ?

A. Switch Widget

B. CupertinoAlertDialog Widget

C. Expansion Panel Widget

D. Bottom Sheet Widget

ANSWER: C

Explanation:

The **Expansion Panel Widget** is the appropriate choice for an interface that presents content in collapsible sections. In Flutter, expansion panels let an app display a concise header for each section and reveal or hide its associated body when the user interacts with that header. This pattern is particularly useful for grouped settings, frequently asked questions, categorized details, or forms where showing every section at once would make the screen crowded.

Flutter provides `ExpansionPanel` objects as the individual panel definitions, typically displayed through an `ExpansionPanelList` or `ExpansionPanelList.radio`. Each panel includes a header, a body, and an expansion state. The parent list handles the layout and exposes callbacks that allow the application to update whether a panel is expanded. This produces the familiar accordion-style interaction represented by the interface.

For implementation details, see the Flutter API documentation for ExpansionPanel and ExpansionPanelList. These components follow Material Design conventions and are intended specifically for expandable content sections.

QUESTION NO: 5

You can add or import a new font to your Flutter by pasting this font file in a font folder in your Flutter project without needing to declare this font file or the font folder in the pubspec.ymal. Is this correct ?

A. True

B. False

ANSWER: B

Explanation:

False is correct because Flutter does not automatically discover arbitrary font files merely because they were copied into a project directory. To make a bundled custom font available to the application, the project must declare a font family and its font asset files under the `flutter:` section of `pubspec.yaml`. The declaration maps the family name used in Dart code, such as through `TextStyle(fontFamily: 'MyFont')`, to the actual file path or paths. It can also define the weight and style associated with each font file, allowing Flutter to select the appropriate regular, bold, italic, or bold-italic variant when rendering text. After editing `pubspec.yaml`, developers should run `flutter pub get` so the asset configuration is processed. The font files may be stored in a conventional directory such as `assets/fonts/`, but the folder location itself has no automatic effect; the files must be referenced accurately in the manifest. Flutter's font documentation provides the required YAML structure and usage details: Use a custom font. The complete manifest reference for font assets is also available in the Flutter assets documentation.

QUESTION NO: 6

A Flutter function performs an asynchronous operation and must wait for a `Future` to complete before using its returned value.

Which Dart keyword can be used inside an `async` function for this purpose?

A. await

B. yield

C. return

D. final

ANSWER: A

Explanation:

`await` is correct because it pauses execution within an `async` function until the specified `Future` completes. When the future completes successfully, the expression evaluates to the result returned by that future. For example, `final data = await loadData();` waits for `loadData()` before assigning its result to `data`.

Using `await` helps asynchronous code read in a sequential manner while allowing the Dart event loop to continue handling other work. A function that uses `await` must normally be marked with `async`. See the Dart asynchronous programming documentation.

QUESTION NO: 7

If you install the Flutter SDK on your computer and configure it as a plug-in for Android Studio or another IDE software, Android Studio will be able to create Flutter apps.

A. True

B. False

ANSWER: A

Explanation:

True is correct. Flutter development requires the Flutter SDK to be installed locally and made available to the development environment. In Android Studio, the Flutter plugin provides IDE integration, including the ability to create a new Flutter project from the New Project workflow, edit Dart code with language assistance, run and debug Flutter apps, and access Flutter-specific tooling. The plugin also installs or enables the Dart plugin when necessary, because Dart is the language used for Flutter application code.

After installing the SDK, Android Studio must be able to locate its installation directory. Once the Flutter plugin is installed and the SDK path is configured, the IDE recognizes Flutter as a supported project type and presents templates for creating Flutter applications. This setup supports targeting Android as well as other Flutter-supported platforms from the same project. Flutter's official installation documentation describes installing the Flutter SDK and adding the Flutter plugin to Android Studio, while the Android Studio documentation covers Flutter project creation and IDE support. See Flutter installation documentation and Flutter support in Android Studio.