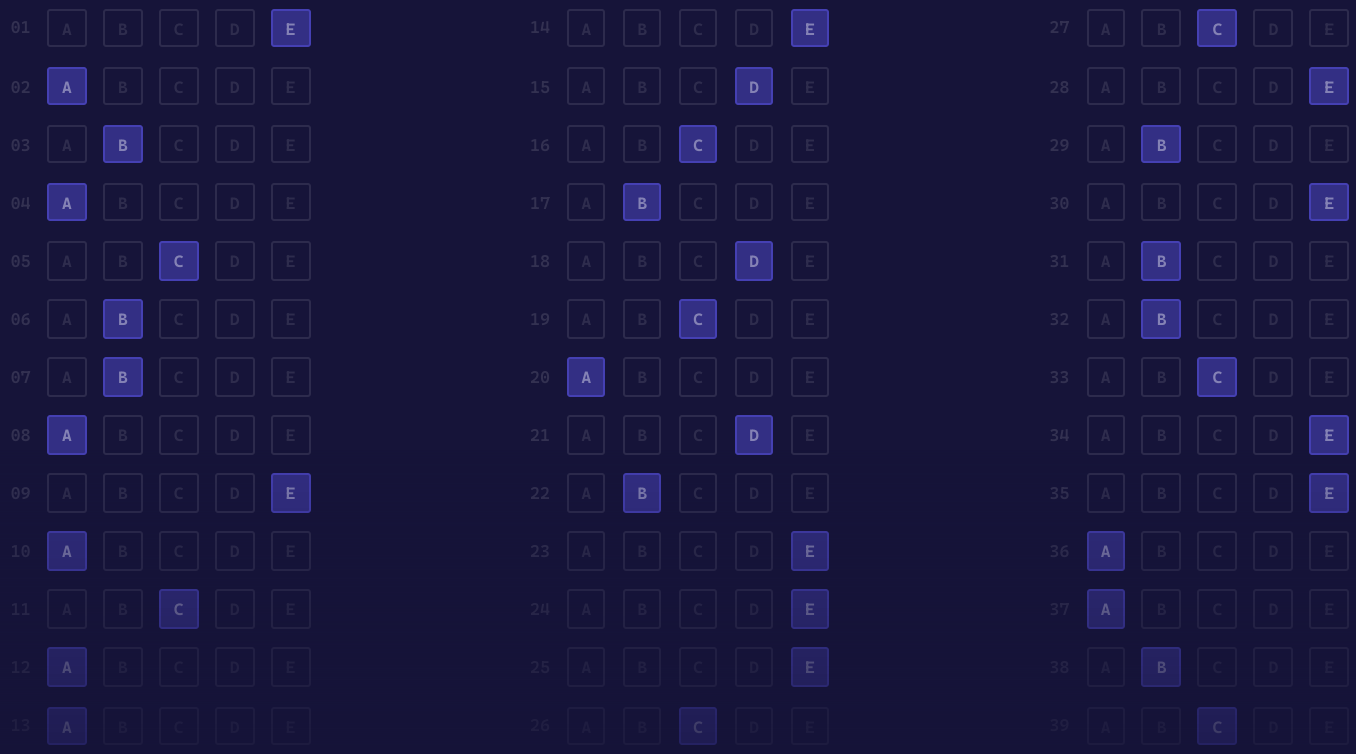
MOMETRIX
SMARTER PREP. BETTER RESULTS.




PRACTICE QUESTIONS

## OutSystems

# FE11P

Front-end Specialization (PT)

| EDITION | QUESTIONS | ISSUED |
|---|---|---|
| Demo | 9 | September 3, 2026 |

WEB www.mometrix.net

SUPPORT support@mometrix.net

# Before you begin

## What this is

9 practice questions for OutSystems `FE11P`, each with its answer key and an explanation. Answer a question before you read its key — the explanation is worth more once you have committed.

## If something looks wrong

Send us three things and we will check it:

- the exam code, `FE11P`
- the question number
- the email address on your order

Write to **support@mometrix.net**. We reply within one business day.

## Licence

This file is licensed to one person. Sharing or republishing it, in whole or in part, ends that licence.

Mometrix

**QUESTION 1** SINGLE CHOICE

In an OutSystems Reactive Web screen, a developer wants to validate that an email field is not empty and matches a valid email pattern before submitting the form. What is the recommended approach?

**A** **Add an 'Is Mandatory' valid property and a 'Valid Email' pattern to the input widget so the framework enforces validation client-side before submit**

B Validate only on the server action and ignore client-side validation

C Disable form submit and rely solely on JavaScript custom code

D Use a database trigger to enforce the constraint

**ANSWER** **A**

**WHY** OutSystems Reactive Web supports built-in validation through the widget's valid properties such as 'Is Mandatory' and 'Valid Email', which the framework enforces before executing the submit action. Server validation alone is insufficient as a recommended pattern, custom JavaScript is not idiomatic, and database triggers are not part of the validation flow.

**QUESTION 2** SINGLE CHOICE

Which OutSystems development practice most directly improves accessibility (a11y) for screen readers in a Reactive Web app?

**A** **Setting meaningful 'Name' (and where applicable 'Label', 'Tooltip') properties on interactive widgets such as buttons and inputs**

B Embedding all UI inside iframes to isolate styling

C Hiding all decorative text using display:none via inline styles

D Disabling the browser's default focus indicators globally

**ANSWER** **A**

**WHY** OutSystems UI widgets expose accessibility-related properties (Name, Label, Tooltip) that map to ARIA attributes and accessible names. Iframes, hiding text, and disabling focus indicators are all anti-patterns that hurt accessibility.

**QUESTION 3** SINGLE CHOICE

A developer needs to display data from an external REST API in a Reactive Web app. What is the correct first step in OutSystems Service Studio?

**A** **Right-click the application node, choose 'Consume REST API', and provide the API's Swagger/OpenAPI or base URL to generate the methods**

B Create a Database Entity that mirrors the JSON payload

C Write a server action that hardcodes a JSON string and parse it manually

D Use a client-side Fetch call from JavaScript embedded in the screen

**ANSWER** **A**

**WHY** OutSystems integrates with REST APIs via the 'Consume REST API' dialog, which introspects the Swagger/OpenAPI spec or the base URL and generates typed server actions. The other options describe patterns not recommended or not first-class in the platform.

**QUESTION 4** SINGLE CHOICE

Which OutSystems feature is the recommended way to persist small pieces of client-side state (for example, a logged-in user's display name) across screen navigations in a Reactive Web app without repeatedly calling the server?

**A** **A Client Variable scoped to the user session**

B A Server Entity stored in the database

C An aggregate computed on every screen load

D A static text on a hidden web block

**ANSWER** **A**

**WHY** Client Variables in Reactive Web are designed to keep values on the client between navigations and are persisted across sessions when configured as such. Server Entities and aggregates require server round-trips, and hidden blocks are not a state mechanism.

mometrix

**QUESTION 5** SINGLE CHOICE

In an OutSystems Reactive Web app, which UI pattern is the standard way to create a responsive layout that adapts to phone, tablet, and desktop breakpoints?

**A** **Use the 'Columns' container pattern from OutSystems UI, placing widgets inside its columns and configuring the device breakpoints**

B Use absolute pixel coordinates on every widget

C Embed three separate iframes, one per device, and show/hide them via JavaScript

D Write custom media queries inside an inline style attribute on the body tag

**ANSWER** **A**

**WHY** OutSystems UI provides a 'Columns' container with responsive breakpoints (phone, tablet, desktop) for building responsive layouts declaratively. The other approaches are not recommended and are inconsistent with the OutSystems UI framework.

**QUESTION 6** SINGLE CHOICE

An OutSystems developer wants to ensure that only users belonging to the 'Manager' role can see and click the 'Approve' button on a screen. Which is the recommended approach?

**A** **Wrap the button in an 'If' widget whose condition checks IsManager from the GetUserRoles server action, and also protect the underlying server action with the Manager role**

B Hide the button using display:none and rely on no server-side check

C Disable the button for all users and never call the action

D Add the role check only in the database trigger

**ANSWER** **A**

**WHY** OutSystems front-end best practice is defense in depth: expose UI conditionally based on roles using an If widget bound to a role check, AND enforce the same role on the server action that mutates data. Hiding alone, disabling, or relying on DB triggers are insufficient or not recommended.

Mometrix

**QUESTION 7** SINGLE CHOICE

In a Reactive Web app, a developer needs to navigate from a 'List Customers' screen to a 'Customer Detail' screen, passing the selected CustomerId. Which OutSystems approach is correct?

**A** **Use a Link/Button with a 'Navigate' destination pointing to 'Customer Detail' and map the CustomerId input parameter to the selected row's Id attribute**

B Store the CustomerId in a global JavaScript variable and read it on the next screen

C Use window.location.href with a query string and manually parse it on the next screen

D Pass the value through a hidden form field submitted via GET

**ANSWER** **A**

**WHY** OutSystems provides declarative navigation with input parameters via the 'Navigate' destination on Link/Button widgets, where source values are bound to destination inputs. Global JS variables, manual query strings, and hidden form fields are not idiomatic OutSystems navigation patterns.

**QUESTION 8** SINGLE CHOICE

In an OutSystems mobile app built with OutSystems UI Mobile, which pattern provides a native-feeling bottom navigation that switches between top-level sections?

**A** **The 'Bottom Navigation' UI pattern, with each item pointing to a different screen via Navigate**

B A sidebar menu shown only in portrait orientation

C A modal popup displayed on every screen load

D A Floating Action Button that triggers navigation

**ANSWER** **A**

**WHY** OutSystems UI Mobile includes a 'Bottom Navigation' pattern purpose-built for top-level mobile navigation. Sidebar menus, modal popups, and FABs are different UI affordances and not substitutes for primary navigation.

**QUESTION 9** SINGLE CHOICE

A developer wants to customize the primary color of an OutSystems Reactive Web app so that all OutSystems UI components automatically adopt the brand color without overriding individual component CSS classes. Which approach is recommended?

A Edit the base CSS file and add an !important rule for each component class

**B Override the CSS custom properties (variables) defined in the OutSystems UI theme**

C Create a new module that injects inline styles into every screen

D Modify the generated HTML of each component in the screen tree

**ANSWER B**

**WHY** OutSystems UI exposes its look-and-feel through CSS custom properties (variables) on the theme. Overriding these variables (for example, --color-primary) in the theme's CSS is the supported, maintainable way to change the primary color across all components without touching individual component classes or generated HTML.

Mometrix