PassQueen
a.k.a dumpsqueen.com

Salesforce Certified MuleSoft Developer 2

Salesforce Salesforce-MuleSoft-Developer-II

Version Demo

Total Demo Questions: 9

Total Premium Questions: 90

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, API Design	13
Topic 2, Application Design	17
Topic 3, DataWeave	1
Topic 4, Flow Design	9
Topic 5, Error Handling	10
Topic 6, Deployment and Management	18
Topic 7, Testing	5
Topic 8, Security	17
Total	90

QUESTION NO: 1

A scatter-gather router is configured with four routes:Route A, B, C and D.

Route C false.

A. Error,errorMesage.payload.results ['2']

B. Payload failures['2']

C. error.errorMessage.payload.failures[2]

D. Payload ['2']

ANSWER: C

Explanation:

`error.errorMessage.payload.failures[2]` is the correct expression for retrieving the failure associated with Route C. When a Scatter-Gather route fails, Mule raises a composite routing error after the router has processed its routes. The error's `errorMessage.payload` contains the Scatter-Gather aggregated output, including a `failures` collection. That collection holds the error information produced by failed route executions. Route C is the third configured route, and collections are zero-indexed in DataWeave expressions, so its position is index `2`. Using the `error` object is essential because the aggregated failure payload is supplied as the error message payload when the Scatter-Gather router raises its composite routing error. This lets an application's error handler inspect the specific route failure and use its details for logging, transformation, or an appropriate recovery action. MuleSoft documents the aggregated results and failures returned by Scatter-Gather in the Scatter-Gather Router documentation. The structure and use of Mule error objects, including `error.errorMessage`, are described in the Mule Errors documentation.

QUESTION NO: 2

An API has been developed and deployed to CloudHub Among the policies applied to this API is an allowlist of IP addresses. A developer wants to run a test in Anypoint Studio and does not want any policies applied because their workstation is not included in the allowlist.

What must the developer do in order to run this test locally without the policies applied?

A. Create a properties file specifically for local development and set the API instance ID to a value that is not used in API Manager

B. Pass in the runtime parameter `-Danypoint.platform.gatekeeper=disabled`

C. Deactivate the API in API Manager so the Autodiscovery element will not find the application when it runs in Studio

D. Run the test as-s, with no changes because the Studio runtime will not attempt to connect to API Manager

ANSWER: B

Explanation:

Pass in the runtime parameter `-Danypoint.platform.gatekeeper=disabled`. This Mule runtime system property disables the embedded Anypoint API Gateway (Gatekeeper) for the locally running application. API Manager policies, including an IP allowlist policy, are enforced through this gateway when the application is associated with its API instance through API autodiscovery. Disabling Gatekeeper for a Studio test run prevents the local runtime from enforcing those managed API policies, so requests from the developer's workstation can reach the application flows without being rejected by the allowlist.

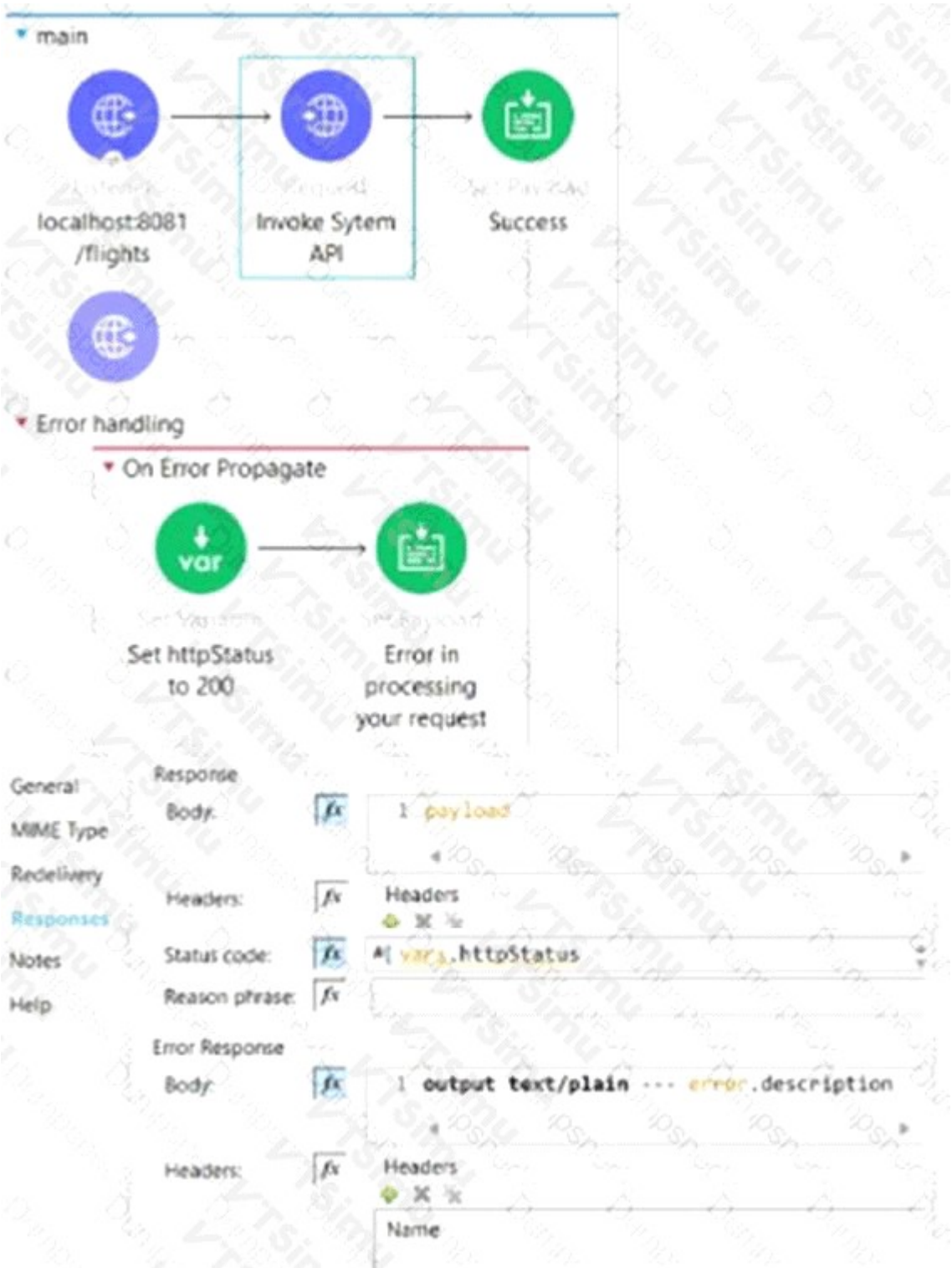
This is specifically useful for local development and testing because it affects the runtime process started by Studio, rather than requiring any modification to the deployed CloudHub application or its centrally managed API configuration. The property should be supplied as a JVM/system-property argument in the local run configuration. It should only be used in a controlled development or test context: the gateway provides the policy-enforcement layer that is expected to remain enabled in deployed environments. MuleSoft describes API autodiscovery and its connection between an API

implementation and API Manager at API Autodiscovery. For Mule runtime API gateway configuration and policy enforcement concepts, see API Gateway on Mule Runtime.

QUESTION NO: 3

The HTTP Request operation raises an HTTP CONNECTIVITY error.

Which HTTP status code and body are returned to the web client?



A. HTTP Status Code: 200.Body 'Error in processing your request'

B. HTTP Status Code: 500.Body 'The HTTP CONNECTIVITY Error description'

C. HTTP Status Code: 500.Body 'Error in processing your request'

D. HTTP Status Code: 500.Body 'Error in processing your request'

ANSWER: A

Explanation:

HTTP Status Code: 200. Body 'Error in processing your request' is correct because the HTTP CONNECTIVITY error is handled by an `on-error-continue` scope. An `on-error-continue` handler marks the error as handled and allows the flow to complete successfully from the perspective of the flow's caller. The handler's Set Payload operation replaces the message payload with `Error in processing your request`, so that value is returned in the HTTP response body.

Because the error is continued rather than propagated, the HTTP Listener produces its normal successful response. When no response status code is explicitly configured in the listener response, Mule uses the default successful HTTP status, `200 OK`. The listener therefore returns a 200 response containing the payload set in the error handler. Mule documentation describes that `on-error-continue` handles the error and treats the owner flow as successfully completed; the HTTP Listener documentation describes configuring response status codes and the normal response behavior. See On-Error Scope and HTTP Listener Configuration Reference.

QUESTION NO: 4

An API has been built to enable scheduling email provider. The front-end system does very little data entry validation, and problems have started to appear in the email that go to patients. A validate-customer'' flow is added validate the data.

What is he expected behavior of the 'validate-customer'' flow?

A. If only the email address Is invalid a VALIDATION.INVALID_EMAIL error is raised

B. If the email address is invalid, processing continues to see if the appointment data and customer name are also invalid

C. If the appointment date and customer name are invalid, a SCHEDULE:INVALID_APPOINTMENT_DATE error is raised

D. If all of the values are invalid the last validation error is raised:SCHEDULE:INVALID_CUSTOMER_NAME

ANSWER: A

Explanation:

If only the email address Is invalid a VALIDATION.INVALID_EMAIL error is raised is the expected behavior. Mule flows execute message processors in sequence. When the email validation processor detects an invalid email value, it raises its validation error immediately. That error interrupts normal processing at that point, so subsequent validation processors for the appointment date and customer name are not reached during that execution path. The error type retains meaningful business context—here, `VALIDATION.INVALID_EMAIL`—which lets the API's error handling return an appropriate response to the calling front end and identify the specific invalid patient-data field.

The `until-successful` scope retries the processors it contains when an error occurs, up to its configured retry limit. It does not allow the flow to continue onward through later processors after a failing validation operation. Therefore, an invalid email is the first validation failure encountered and is the relevant validation error for this scenario. Mule's error-handling model propagates an error unless it is handled by an applicable error handler, preserving controlled failure behavior for validation logic. See MuleSoft's Until Successful Scope documentation and Error Handling documentation.

QUESTION NO: 5

A Mule application must create a PGP signature for outbound files. The receiving partner already has the Mule application public key and must be able to verify that the file was produced by the Mule application.

Which key is required by the Mule application to create the signature?

A. The Mule application private key

B. The Mule application public key

C. The partner private key

D. The partner public key

ANSWER: A

Explanation:

The Mule application must use its own private PGP key to create the signature. The receiving partner uses the corresponding Mule application public key to verify both the origin and integrity of the signed file.

The partner public key is used when Mule encrypts content so that only the partner can decrypt it. It is not used to create a signature that the partner can verify as originating from Mule.

QUESTION NO: 6

A developer has created the first version of an API designed for business partners to work commodity prices.

What should developer do to allow more than one major version ofthe same API to be exposed by the implementation?

A. In Design Center, open the RAML and modify each operation to include the major version number

B. In Anypoint Studio, generate scaffolding from the RAML, and the modify the in the generated flows to include a parameter to replace the version number

C. In Design Center, open the RAML and modify baseUri to include a variable that indicates the version number

D. In Anypoint Studio, generate scaffolding from the RAML, and then modify the flownames generated by APIKit to include a variable with the major version number

ANSWER: C

Explanation:

In Design Center, open the RAML and modify baseUri to include a variable that indicates the version number is correct because API versioning should be expressed as part of the API's base URL rather than repeated in every individual operation. RAML supports URI parameters in the `baseUri`, allowing the API contract to define a versioned path such as `https://api.example.com/commodities/{version}`. The version can then be supplied as a value—for example, `v1` or `v2`—when the API is implemented and exposed. This produces distinct, clear endpoints for each major version while keeping the resource paths and operations organized under one API specification approach. Major versions commonly represent potentially breaking contract changes, so placing the version in the base URI lets partner applications explicitly select the contract version they depend on. In an APIkit implementation, the RAML base URI and its parameters establish the API's routing structure; the implementation can therefore expose the required versioned endpoint without encoding version details into every flow name or operation definition. MuleSoft also recommends managing API versions deliberately so consumers can continue using an existing version while a newer major version is introduced. See MuleSoft's API specification design documentation and its API version management documentation.

QUESTION NO: 7

Refer to the exhibit.

```
traits:
  client-id-required:
      headers:
        client_id:
          type: string
        client_secret:
          type: string
protocols:
  - HTTPS
  - HTTP
types:
  Customers: !include datatypes/Customers-request.raml

/customers:
  is: [client-id-required]
  put:
    body:
      application/json:
        type: Customers
        example: !include examples/Customers-request.json
    responses:
        200:
            description: Successfull
            body:
              application/json:
                example: !include examples/Customers-response.json
        400:
          description: Bad request
          body:
            application/json:
              example: !include examples/postErrorCode403.json
        404:
            description: Resource not found
            body:
              application/json:
                example:
                  !include examples/postErrorCode404.json
        500:
            description: Internal server error
            body:
              application/json:
                example:
                  !include examples/postErrorCode500.json
        501:
          description: Not implemented
          body:
            application/json:
              example:
                !include examples/postErrorCode501.json
```

A developer generates the base scaffolding for an API in Anypoint Studio.

Which HTTP status code is returned while testing using the API Kit console if no values are entered in client-secret?

A. HTTP status code:200

B. HTTP status code:403

C. HTTP status code:400

D. HTTP status code:500

ANSWER: C

Explanation:

HTTP status code:400 is returned because the generated APIkit application validates incoming requests against the API specification before routing the request to an implementation flow. The `client-secret` input is defined as a required HTTP header in the API contract. Leaving that field empty in the APIkit Console means the required header is not supplied

with a valid value in the request. This is a request-validation failure: the request does not conform to the contract that the API exposes.

APIkit identifies this condition as a bad request and raises its bad-request validation error. In the base scaffold generated by Anypoint Studio, the APIkit error handling maps that error to the HTTP 400 response status. This behavior occurs at the router/contract-validation layer, before any endpoint business logic needs to run. Required headers are therefore an effective way to ensure that clients provide mandatory request metadata consistently during both console testing and runtime invocation. See MuleSoft's APIkit header validation documentation and the APIkit error-handling documentation for the router validation and status-code mapping behavior.

QUESTION NO: 8

A new Mule project has been created in Anypoint Studio with the default settings.

Which file inside the Mule project must be modified before using Maven to successfully deploy the application?

A. Settings.xml

B. Config.yaml

C. Pom.xml

D. Mule.artificact.json

ANSWER: C

Explanation:

Pom.xml is the Maven Project Object Model file and is the project-level file that Maven reads to build and deploy a Mule application. A Mule project generated by Anypoint Studio includes a basic POM containing the application coordinates, Mule runtime configuration, dependencies, and Mule Maven Plugin declaration. Before a Maven-based deployment, the project's POM must be configured with the deployment target and the settings required by that target. For example, a CloudHub deployment commonly requires Mule Maven Plugin configuration for the business group, environment, application name, region, worker details, and deployment properties. Maven then uses this configuration when the relevant deployment goal is executed. The POM also provides the dependency and packaging information needed to produce the deployable Mule application artifact consistently outside Studio, which is why it is the central file for Maven-driven deployment automation. Credentials should generally be supplied securely through Maven settings, environment variables, or CI/CD secret management rather than stored directly in the project POM. MuleSoft documents the Mule Maven Plugin's deployment configuration in its Mule Maven Plugin documentation and provides deployment-specific guidance in the CloudHub deployment documentation.

QUESTION NO: 9

Which statement is true about using mutual TLS to secure an application?

A. Mutual TLS requires a hardware security module to be used

B. Mutual TLS authenticates the identity of the server before the identity of the client

C. Mutual TLS ensures only authorized end users are allowed to access an endpoint

D. Mutual TLS increases the encryption strength versus server-side TLS alone

ANSWER: B

Explanation:

Mutual TLS authenticates the identity of the server before the identity of the client. In a standard TLS handshake, the server first presents its certificate to the client. The client validates that certificate, including its trust chain and, where applicable, the hostname, to establish that it is communicating with the intended server. With mutual TLS enabled, the server subsequently requests a certificate from the client. The client presents its certificate and proves possession of the associated private key,

allowing the server to validate the client's identity as well. This sequence provides two-way certificate-based authentication while TLS protects data in transit using the session security negotiated during the handshake. In Mule applications, mutual authentication is configured through the TLS context so that the application can supply its own key material and trust the certificates used by its peer. The client certificate is therefore an additional authentication step after the server has identified itself to the client. MuleSoft documents this server-certificate-then-client-certificate flow for mutual authentication in its TLS Configuration documentation. The underlying behavior is defined by the TLS protocol, including the server's CertificateRequest message described in RFC 8446.