1. **How would you identify and prioritise the test cases needed across all components?**

To identify and prioritise test cases, I would follow a **risk-based and component-driven approach**:

1. **Analyse the change impact:**  
   The new mandatory field UniqCltID affects the GUI, API, message protocol, and data flow across all three components. So, I would trace how this field is created, validated, transmitted, stripped, stored, and reported.
2. **Identify key test areas per component:**
   * **GUI/API:** Validate that UniqCltID is mandatory and follows the correct format (3 letters + 9 digits).
   * **Component 1:** Verify that client-specific fields including UniqCltID are removed before sending data to Component 2.
   * **Component 2:** Ensure no UniqCltID or other client details appear in broadcast messages.
   * **Component 3:** Confirm UniqCltID is stored correctly in the database and displayed in CSV reports.
3. **Design test cases:**  
   Create both **positive** (valid UniqCltID flow) and **negative** (missing, invalid format, boundary cases) tests, plus **integration** tests to verify correct data flow and sanitisation between components.
4. **Prioritise based on risk and impact:**
   * **High priority (P1):** Validation of UniqCltID format and mandatory rule, Component 1 sanitisation, Component 3 data storage.
   * **Medium priority (P2):** End-to-end message flow, CSV reporting, backward compatibility.
   * **Low priority (P3):** Performance, logging, and UI presentation checks.
5. **Ensure coverage and traceability:**  
   Map each requirement to test cases in a traceability matrix and tag automated tests by priority (P1, P2, P3) to focus regression runs on the highest-risk areas.
6. **What types of automated tests would you implement at each layer (GUI, API, messaging, database, reporting), and why?**
7. **GUI Layer:**

* Tests: UI field validation (mandatory check, format validation, error messages).
* Why: To ensure users can only enter valid UniqCltID values (3 letters + 9 digits) and receive correct feedback for invalid input.
* Tools: Selenium or Playwright with Pytest for input and form submission automation.

1. **API Layer:**

* Tests: Positive and negative API requests, schema validation, mandatory field checks, and response code verification.
* Why: To confirm the API correctly enforces the new field rules and handles errors gracefully.
* Tools: Pytest with Requests or Postman/Newman.

1. **Messaging Layer (between components):**

* Tests: Integration and contract tests verifying that UniqCltID is removed before sending to Component 2 and retained for Component 3.
* Why: To ensure correct data sanitisation and prevent leakage of client information.
* Tools: Mocking or stubbing message queues, contract testing frameworks like Pact.

1. **Database Layer:**

* Tests: Data persistence and integrity checks — confirm UniqCltID is stored accurately in the database.
* Why: To ensure no data loss, truncation, or corruption occurs during storage.
* Tools: Pytest with SQL queries or ORM verification scripts.

1. **Reporting Layer (CSV generation):**

* Tests: Verify that CSV reports include the correct UniqCltID and match database records.
* Why: To ensure reporting accuracy and end-to-end data consistency.
* Tools: Automated comparison of CSV output against DB query results using Python scripts.

1. How would you ensure both backwards compatibility and data integrity throughout the system after this change?

**Ensuring Backward Compatibility and Data Integrity**

1. **Backward Compatibility:**
   * **Versioning:** Introduce version control in the message schema or API (e.g., v1 for old format, v2 including UniqCltID) so older clients can continue using the existing structure temporarily.
   * **Optional Handling During Transition:** Allow the system to accept messages without UniqCltID during migration, logging warnings instead of hard failures.
   * **Contract Tests:** Implement automated contract tests between components to ensure older message formats are still correctly parsed and do not break downstream systems.
   * **Regression Testing:** Run full regression suites to confirm existing functionalities and data flows are unaffected by the new field.
2. **Data Integrity:**
   * **Validation Rules:** Enforce strict validation (format and length) of UniqCltID at input (GUI/API) to prevent bad data entering the system.
   * **Sanitisation Verification:** Test that Component 1 correctly removes client-specific data (including UniqCltID) before forwarding to Component 2.
   * **Database Consistency Checks:** Automate database verification to ensure UniqCltID values are stored accurately and not modified or truncated.
   * **End-to-End Data Comparison:** Compare source input, internal message, and final report (CSV) to confirm that all data is correctly preserved through each component.
   * **Monitoring & Alerts:** Add runtime checks to detect schema mismatches or missing UniqCltID values in production data.
3. Can you provide an example of a test case you would automate for Component 1?

**Example Automated Test Case for Component 1**

**Objective:**  
Verify that Component 1 correctly removes client-specific information, including the new UniqCltID, before forwarding the message to Component 2.

**Test Case ID:** TC\_C1\_001  
**Component:** Component 1 (Message Processing)  
**Test Type:** Integration / Functional Automation

**Precondition:**  
Component 1 is running and configured to receive input messages and forward sanitized messages to Component 2 (or a mock endpoint).

**Test Steps:**

1. Send a valid input message to Component 1 via API:

ini

Client=Bob Smith|Company=ABC|Product=ISX|Quantity=100|Price=17|UniqCltID=JPM000023789

1. Capture the message output that Component 1 sends to Component 2.
2. Inspect the outgoing message payload.

**Expected Result:**

* Client, Company, and UniqCltID fields are **removed**.
* The sanitized message forwarded to Component 2 should look like:

ini

Product=ISX|Quantity=100|Price=17

* No personal or client-specific data should appear in logs or the broadcast output.
* Component 3 (if configured) still receives the **full** message, including UniqCltID.

**Automation Approach:**  
Use **pytest** with a **mock receiver** (e.g., a local HTTP server or message queue listener) to automatically send the input, capture the output, and assert that restricted fields are stripped.

1. What challenges do you anticipate in automating validation for the new UniqCltID field, and how would you address them?

**Challenges in Automating Validation for UniqCltID and How to Address Them**

1. **Challenge 1 – Complex Validation Rules:**  
   The UniqCltID must strictly follow the format of 3 letters followed by 9 digits.  
   **Solution:** Use regex-based validation (^[A-Z]{3}\d{9}$) and automate both **positive** and **negative** test data generation to cover all boundary and invalid cases.
2. **Challenge 2 – Consistent Handling Across Layers (GUI, API, Messaging):**  
   Each component (input, processing, storage) must interpret and validate the field consistently.  
   **Solution:** Define a **common validation utility or schema** used by all components and test it with **contract and integration tests** to ensure uniform enforcement.
3. **Challenge 3 – Data Privacy in Tests:**  
   The field contains client-specific information, so test data must not expose real client IDs.  
   **Solution:** Use **synthetic test data generators** (e.g., random prefixes + digits) and ensure sensitive data is masked in logs and reports.
4. **Challenge 4 – Backward Compatibility:**  
   Older messages without UniqCltID may still be processed during migration.  
   **Solution:** Automate **dual-path testing**—one suite for new messages (with UniqCltID) and another to validate the system’s behaviour with old message formats.
5. **Challenge 5 – Multi-layer Verification:**  
   Validation must be confirmed at input, during sanitisation, and after storage.  
   **Solution:** Build **end-to-end automated tests** that trace UniqCltID through all components and assert expected presence or removal at each stage.