Problem Solving: Reference number

1. Reference number changed after loan updates, risking reconciliation and audit failures.
   → Identified regeneration logic in update APIs and enforced immutability checks.

2. Retries created duplicate references during timeouts or page refreshes.
   → Simulated retry scenarios and pushed for idempotent handling with uniqueness constraints.

3. Mismatch of reference numbers across systems (Loan Engine, CRM, Accounting).
   → Introduced cross-system validation and reconciliation checks in regression testing.

4. QA coverage was limited to UI validation, missing backend testing.
   → Expanded testing to API and data integrity validations.

5. Admin users could manually edit reference numbers, violating compliance rules.
   → Flagged it as audit risk and ensured reference fields were locked at UI and API levels.

6. Partial failures caused inconsistent data states.
   → Tested failure and recovery scenarios to validate safe retry behavior.

7. Lack of clear state ownership for reference generation.
   → Collaborated with developers to restrict reference creation strictly to loan creation flow.

Problem Solving: DOB

1. DOB updated in UI but reverted after refresh
   → Added save–refresh–relogin validation and verified DB commit instead of trusting UI success messages.

2. DOB mismatch between CRM and KYC systems
   → Traced data flow end-to-end and enforced a single source of truth (KYC) with

re-verification on DOB change.

3. Earlier, Backened Data was not validating
 → Validated API responses and database state to detect partial or failed transactions.

4. Leap-year (29 Feb) DOB causing age calculation errors
 → Introduced boundary test data and verified consistent date handling across services.

<u>Problem Solving: Loan Rejection</u>

1. Duplicate rejection of terminal-state((`rejected and` failed)) loans
 → Identified missing backend terminal-state validation and enforced strict state machine rules.

2. Loan rejection reappearing after page refresh
 → Ensured UI reloads state from API and blocks actions for REJECTED / FAILED loans.

3. API retries creating multiple rejection records
 → Designed idempotency test cases and validated duplicate requests return 409 status code.

4. Parallel user actions causing race conditions
 → Simulated concurrent rejections and verified backend state check before processing.

5. UI allowing invalid actions on terminal states
 → Added terminal-state-based UI locking and read-only behavior.

6. Audit log corruption due to duplicate workflows
 → Once an audit entry is written, it cannot be changed or duplicated. No matter how many times the system receives a reject request.
7. Lack of retry and timeout testing in QA scope
 → Expanded regression to include network failure, refresh, and retry scenarios.
8. Inconsistent data across systems (UI, DB)
 → Performed cross-system validation to ensure state consistency everywhere.