MASFIQUR RAHMAN

QA Work Demo · Sample Artifacts

Prepared for: **QA Tester — Web Application**
Focus: Data migration / ETL workflows · Data governance · Web QA
14+ years Full-Stack QA · Playwright · Manual & Exploratory · API

14+	Playwright	40 hrs	US overlap
Years QA experience	Web UI automation	Weekly availability	Business hours ready

About this demo

This PDF contains **illustrative sample artifacts** that show how I plan tests, document defects, approach Playwright automation, and validate data-centric workflows. They are anonymized demos — not client-confidential materials — built to match the needs of a web-based data operations / ETL / governance platform.

What I bring to this role

- Risk-based test plans for functional, regression, integration, and exploratory testing
- Data integrity focus: mappings, transforms, row counts, nulls, duplicates, partial failures
- Clear, reproducible bug reports with severity, evidence, and expected vs. actual
- Hands-on **Playwright** automation + SQL/dataset checks beyond UI-only testing
- Collaboration with engineers & PMs · 40 hrs/week · US business-hours overlap

Domains & tools

Domains: SaaS · Fintech · Healthcare · eCommerce · Education · Marketplace
Testing: Manual · Exploratory · Regression · E2E · Usability · API (Postman)
Automation: Playwright (selectors, fixtures, CI smoke, API + UI assertions)
Tracking: Jira · Linear · GitHub · ClickUp · Trello · Loom · BrowserStack

Contents

Tip for reviewers: skim page 3 for bug-report quality, page 4 for Playwright style, and page 5 for how I prove data correctness beyond the UI. These samples reflect my documentation style and can be adapted quickly to your stack after a short walkthrough.

1. Profile Snapshot & Playwright Experience

Full-Stack QA Engineer with 14+ years testing web and mobile applications across eCommerce, SaaS, healthcare, education, fintech, and marketplace platforms. I combine meticulous manual/exploratory testing with practical automation so teams catch regressions early without slowing delivery.

Core strengths aligned to this role

- Functional, regression, integration, exploratory, E2E, usability, and API testing
- Test plans & cases; defect lifecycle; clear repro steps with evidence (screenshots, Loom video)
- Data-aware validation: outputs, dataset compare, transformation spot-checks
- **Playwright** for web UI automation — stable selectors, E2E flows, regression suites
- Tools: Jira, Linear, GitHub, ClickUp, Trello, Postman, BrowserStack, TestFlight, Firebase, Loom
- Real-device coverage: iPhone, iPad, Android, MacBook Pro, Windows, Linux
- Availability: 40 hrs/week · US business-hours overlap · independent / startup-ready

Playwright experience (highlight)

I use Playwright to automate critical web paths: auth, multi-step wizards, job status polling, table/grid assertions, file upload flows, and cross-browser smoke. I prefer role/label-based locators, fixtures for auth state, and assertions tied to user-visible outcomes — plus API or DB checks when UI alone cannot prove data integrity.

2. Sample Test Plan — Data Migration / ETL Web App

Product context (demo): A web platform where operators configure source → transform → destination jobs, monitor runs, and review governance/audit metadata.

Objectives

- Verify functional correctness of job setup, execution, monitoring, and governance views
- Protect data integrity: mapping, transforms, counts, nulls, duplicates, partial failure
- Reduce release risk via regression coverage (manual checklist + Playwright smoke)
- Clarify ambiguous requirements early with engineering & product

Scope (in)

- Auth & roles (admin / operator / read-only)
- Source connection & schema preview
- Field mapping & transformation rules
- Job run lifecycle (queued → running → success / failed / partial)
- Error surfacing, retries, and audit/governance logs
- Export / download of run reports

Out of scope (demo)

Performance/load at scale, security penetration testing, and third-party vendor SLAs (would be separate workstreams).

Risk-based test areas

Area	Risk	Focus
Field mapping	High	Wrong type coercion, dropped columns, silent nulls
Transforms	High	Edge values, empty strings vs null, timezone/date formats
Partial failure	High	Row-level errors, resume/retry, audit trail accuracy
Permissions	Med	Read-only cannot mutate; PII fields masked per role
UI status	Med	Stale status, polling gaps, misleading success banners
Governance	Med	Who changed what / when; incomplete audit events

Sample test cases (excerpt)

ID	Title	Type	Priority
TC-01	Create job with valid CSV source and complete mapping	Functional	P0
TC-02	Reject job when required destination field unmapped	Negative	P0
TC-03	Transform: trim + lowercase email; verify output sample	Data	P0
TC-04	Duplicate primary keys → flagged / quarantined per rules	Data	P0
TC-05	Job fails mid-run; retry processes remaining rows only	Integration	P1
TC-06	Read-only user cannot edit mapping or trigger run	Security/UX	P1
TC-07	Audit log shows actor, timestamp, before/after mapping	Governance	P1
TC-08	Regression smoke: login → open last job → download report	Regression	P0

Entry criteria: build deployed to staging, seed datasets available, known-good baseline counts documented. Exit criteria: P0/P1 defects resolved or waived with owner; smoke suite green; release notes reviewed.

3. Sample Bug Reports

Style I use in Jira/Linear/GitHub: short title, environment, severity, clear steps, expected vs actual, evidence, and data impact when relevant.

BUG-1041 — Silent data loss when optional source column is missing

Severity	**Critical**
Environment	Staging · Chrome 126 · macOS · Role: Operator
Build	web-app@1.48.2-rc.3
Component	ETL Mapping / Job Runner

Summary

When a mapped source column is absent from an uploaded file, the job completes as **Success** and writes destination rows with nulls for that field — without warning. Operators believe the migration completed correctly.

Steps to reproduce

- Create a job mapping `customer.email` → destination `email` (required in business rules, optional in UI).
- Upload CSV that omits the `email` column entirely.
- Run job and wait until status = Success.
- Open run detail → sample output / export.

Expected

Job should fail validation (or complete as Partial/Failed) with a clear message: mapped column missing. No silent null fill for required business fields.

Actual

Job Success. Destination rows written with null emails. No warning in UI or audit log.

Data impact

Integrity risk: downstream systems receive incomplete customer records; governance trail does not flag the omission.

Evidence

Screenshots of mapping + success banner; exported CSV sample; Loom walkthrough (2m).

BUG-1048 — Governance audit omits actor on mapping edits via bulk paste

Severity	**High**
Environment	Staging · Firefox 127 · Windows 11 · Role: Admin
Component	Data Governance / Audit Log

Summary

Bulk-pasting field mappings updates the job config, but the audit log shows **system** as actor instead of the signed-in admin — breaking accountability.

Steps to reproduce

- Open an existing job → Mapping tab → Bulk paste 5 field pairs → Save.
- Navigate to Governance → Audit for that job.
- Inspect the latest “mapping updated” event.

Expected

Actor = signed-in admin email/user id; before/after diff retained.

Actual

Actor = system; before/after present but attribution wrong.

4. Playwright Automation Approach (Examples)

I keep automation focused on high-value, stable flows. Data assertions are paired with UI checks so a green suite means more than “page loaded.”

What I typically automate first

- Login + role-gated navigation smoke
- Create job → map required fields → start run → assert terminal status
- Negative: missing required mapping blocks Run
- Download/export report appears after success
- Cross-browser smoke (Chromium + Firefox) on CI for P0 paths

Example: job success smoke (TypeScript / Playwright)

```
test('operator can run a mapped job to success', async ({ page }) => {
  await page.goto('/jobs/new');
  await page.getByLabel('Job name').fill('QA Demo Migration');
  await page.getByRole('button', { name: 'Upload source' }).click();
  await page.setInputFiles('input[type=file]', 'fixtures/customers_ok.csv');
  await page.getByRole('button', { name: 'Auto-map' }).click();
  await expect(page.getByText('All required fields mapped')).toBeVisible();
  await page.getByRole('button', { name: 'Run job' }).click();
  await expect(page.getByTestId('job-status')).toHaveText('Success', { timeout: 120_000 });
  await expect(page.getByText(/rows written:\s*100/i)).toBeVisible();
});
```

Example: data integrity guard (API + UI)

```
// After UI success, confirm counts via API (or SQL in staging)
const run = await request.get(`/api/jobs/${jobId}/runs/latest`);
expect(run.ok()).toBeTruthy();
const body = await run.json();
expect(body.status).toBe('success');
expect(body.metrics.rows_read).toBe(body.metrics.rows_written);
expect(body.metrics.rows_failed).toBe(0);
```

Practices I follow

- Prefer getByRole / getByLabel over brittle CSS; testids only when needed
- Fixtures for auth storageState to keep suites fast
- Seed known CSVs in repo; assert on metrics, not only screenshots
- Keep P0 smoke under ~10 minutes; expand regression separately
- Fail loudly on silent data mismatches (counts, required fields)

5. Data Validation Checklist & SQL Spot-Checks

For data migration / ETL / governance features, I treat the database (or warehouse) as part of the product under test — not an afterthought.

Pre-run checklist

- Document source row count, distinct keys, null rates for critical columns
- Confirm mapping version / transform rules under test
- Note timezone and locale assumptions for dates/currency
- Define success criteria: counts, checksums, sample row equality

Post-run validation

Check	Method	Pass criteria
Row counts	UI metrics + SQL COUNT(*)	read = written (+ explained rejects)
Key uniqueness	SQL GROUP BY HAVING COUNT > 1	No unexpected duplicates
Required fields	SQL WHERE col IS NULL	Zero nulls for required columns
Transform sample	Compare 20–50 keyed rows	Expected casing/format/rules
Rejects	Quarantine table / error file	Each reject has reason code
Audit	Governance UI + audit table	Actor, time, action present

Example SQL spot-checks (staging)

```
-- Count parity
SELECT 'src' AS side, COUNT(*) FROM staging.source_customers
UNION ALL
SELECT 'dst', COUNT(*) FROM staging.dest_customers WHERE job_run_id = :run_id;

-- Required email present
SELECT COUNT(*) AS null_emails
FROM staging.dest_customers
WHERE job_run_id = :run_id AND (email IS NULL OR email = '');

-- Duplicate business keys
SELECT external_id, COUNT(*)
FROM staging.dest_customers
WHERE job_run_id = :run_id
GROUP BY external_id HAVING COUNT(*) > 1;
```

Closing note

I enjoy finding problems before customers do — especially the quiet data issues that UI-only testing misses. Happy to walk through these samples on a call and adapt a test strategy to your stack, environments, and release cadence.

Masfiqur Rahman · Full-Stack QA Engineer · 14+ years
Playwright · Manual / Exploratory / API · Data validation · 40 hrs/week · US hours overlap