### 

PayFuture Financial Services Ltd.  
QA / Automation Engineer Task

### 

# **Title:**

Full-Stack Test Automation for Web, API, and Performance with CI/CD

# **Objective:**

Assess your ability to create a comprehensive, automated test suite that integrates with modern DevOps pipelines, covering functional, API, and performance testing. The goal is to demonstrate how you approach automated quality assurance in real-world CI/CD environments with diverse tech stacks.  
  
**Technical Requirements**

## **Functional Testing (Frontend UI)**

· Use Cypress, Playwright, or TestCafe to write end-to-end (E2E) tests for a simple frontend.

· Test cases should include:

· - User navigation flow (login, form submission, etc.)

· - Field validation and error handling

· - Assertions on DOM elements and results

· Test logic should be structured using best practices (Page Object Model, reusable functions, etc.)

## **API Testing (Backend Integration)**

· Use Postman, REST Assured (Java), or Supertest (Node.js) to test APIs.

· Use reqres.in or JSONPlaceholder as the public API target.

· Create 4–5 test cases, including:

· - Basic GET and POST requests

· - Validation of response status codes and payloads

· - Handling negative tests (e.g., 404/500)

· - Data-driven testing with input variations

· Extra credit for automated schema validation or mocking with tools like WireMock.

## **Performance Testing**

· Use JMeter, Locust, or k6 to simulate load on the API.

· Simulate 100 concurrent users over 60 seconds.

· Include graphs or reports showing latency, error rate, and throughput.

· Bonus for using environment variables or config files to parameterize the load profile.

## **CI/CD Integration**

· Integrate all tests into a GitLab CI pipeline using a `.gitlab-ci.yml` file.

· Pipeline should include:

· - Test execution stages (e.g., `e2e`, `api`, `performance`)

· - HTML test reports or logs as CI artifacts

· - Clear pipeline exit codes for success/failure visibility

## **Dockerization**

· Containerize your test suite (or parts of it) using Docker.

· Provide a `Dockerfile` and `docker-compose.yml` for local orchestration.

## **Documentation**

· Include a `README.md` with:

· - Setup instructions (local and CI)

· - Test coverage summary

· - Tool selection rationale

· - Example screenshots or report links

# **Bonus (Optional but Valuable)**

· Link test cases to a Test Management Tool (e.g., TestRail or XRay).

· Include browser matrix testing via BrowserStack or Playwright Cloud Grid.

· Add test coverage badges, code linting, or pre-commit checks.

· Explore mock server setup (e.g., MSW, WireMock, Prism).

# **Deliverables**

· A Git repository (GitLab or GitHub) with:

· - Source code, test scripts, Docker setup, and `.gitlab-ci.yml`.

· An optional Loom video walkthrough (5–8 min) explaining structure, test design decisions, and pipeline demo.

# **Evaluation Criteria**

| Area | Weight | What We Look For |
| --- | --- | --- |
| Functional Test Quality | 25% | Clean, modular, and realistic UI tests using proper structure and assertions |
| API Test Coverage | 20% | Well-formed API requests, negative cases, automation potential |
| Performance Testing | 15% | Usable load simulation, graph output, parameterization |
| CI/CD Pipeline | 20% | Working GitLab pipeline, test reports, clear stages and fail/pass indicators |
| Dockerization & Infra | 10% | Proper containerization and reproducibility of test runs |
| Documentation & UX | 10% | Clear instructions, context, and demonstration of quality mindset |

# **Tech Stack You May Encounter at PayFuture**

· Frontends: React, Vue, Angular

· Backends: Node.js, Java Spring Boot, Python (Flask, FastAPI), .NET

· APIs: REST, GraphQL, Webhooks

· Infrastructure: Kubernetes, GCP, GitLab CI, Docker, Vault

· Monitoring: Prometheus, Grafana, DataDog