Assignment – QA Engineer

(Automation & Manual)

Confidentiality Notice

This document is a strictly confidential communication intended solely for the use of the recipient. It may not be copied, reproduced, shared, or distributed in any format—electronic, physical, or otherwise—without prior written authorization from **Oppizi** or its affiliated entities.

If you do not agree to the above terms, you are required to discontinue reading, delete any copies in your possession, and notify Oppizi's recruitment or human resources team accordingly.

Part 1 – System Journeys and Architecture

Background

Oppizi is a logistics and tracking platform supporting offline marketing campaigns. It enables brands to manage flyer distribution, sampling efforts, and DOOH logistics through dashboards, field apps, and reporting systems.

System Components

- **Campaign Dashboard** – for campaign creation and route assignments
- **Field Agent Mobile App** – used by agents to access routes, scan flyers, and check in/out
- **Reporting Engine** – aggregates performance and delivery metrics
- **APIs and Microservices** – for internal operations and external integrations
- **Data Services** – for geolocation, QR validation, route optimization, and audit logs

User Journey 1 – Campaign Creation

1. Admin user creates a campaign via the dashboard UI.
2. System sends API requests to validate:
    - Serviceable geolocation areas
    - QR code pool availability
    - Flyer stock levels
3. Campaign metadata is stored in the campaign database.
4. Campaign status transitions to `Draft`.

User Journey 2 – Agent Flyer Scan

1. Agent opens the assigned campaign and route via the mobile app.
2. Agent scans a flyer's QR code.
3. The app submits the scan data (QR code, location, timestamp) via the agent API.
4. The backend:
    - Verifies geofencing compliance
    - Validates flyer against the assigned campaign
    - Logs a delivery entry with scan details

Task

Design one or more diagrams that clearly show the integration flow for these two journeys. Use whichever format you find most suitable (e.g., sequence diagram, swimlane flowchart, or component interaction diagram).

The goal is to highlight the integration points between services and systems for purposes of system integration testing.

Part 2 – API Testing

The Open Charge Map API simulates access to a comprehensive database of location-based infrastructure—such as electric charging stations—allowing Oppizi to test and validate geolocation features, mapping accuracy, and campaign planning logic in environments similar to real-world delivery points.
Web URL: https://map.openchargemap.io/#/search

Here's a clear documentation of the main endpoints:

Base URL
`https://api.openchargemap.io/v3/`

Authentication
An API key is required for all requests. Include it as a query parameter:
`?key=YOUR_API_KEY`
To get a valid API key, visit the web URL and inspect the API calls triggered.

Endpoints

1. **Get POIs (Points of Interest)**
   Endpoint: `/poi/`
   Retrieves charging station data.
   Key parameters:
   - `latitude` and `longitude`
   - `distance`
   - `countrycode`
   - `maxresults`
2. **Get Reference Data**
   Endpoint: `/referencedata/`
   Provides static reference values for charging stations.

Exercise Objective

Create a set of API tests for the Open Charge Map API using a testing tool of your choice.
Allowed tools: Postman or Cypress.

Requirements:

1. Create tests for:
   - `GET /poi/`
   - `GET /referencedata/`
2. For each endpoint, validate:
   - Response time (under 1000ms)
   - Status code (200)
   - Response schema

    - Business logic
3. Document how to import/run your tests.
4. Provide a brief test report and any observations.

Deliverables:

- Test scripts or collection
- `README.md` with setup instructions
- Optional: test reports or screenshots

Part 3 – Manual Testing Assignment

Scenario: Route Conflict Detection and Reassignment Auditing

Oppizi has launched a feature in the admin dashboard that allows managers to **reassign routes across multiple campaigns**. However, newly introduced constraints must be validated thoroughly:

Feature Description:

- Routes can be reassigned only:
  - Before the campaign start date
  - If the receiving agent has no overlapping route at the same time
  - If the location types match (e.g., indoor vs. outdoor)
- Each reassignment must:
  - Be logged in the audit system
  - Trigger a confirmation email to both agents
  - Lock the reassigned route from being changed again for 24 hours

Task:

Write a **comprehensive test suite** to manually validate this feature.

Include the following:

1. **Test Design**
   - Use-case-based test cases (not just field validation)
   - Define functional, negative, edge, and permission-related scenarios
   - Example: “Attempt to reassign a route to an agent with an overlapping indoor route in another campaign”
2. **Test Data Table**
   - Include campaign names, agent names, route IDs, and assignment schedules
   - Map each test case to data sets (e.g., Campaign A → Agent X → Route 5)
3. **Audit and Email Verification**
   - Design test steps to check:
     - Audit log entries (via logs or API call)
     - Email sent events (mock or real inbox, test environment)
4. **Bug Reporting Sample**
   - Write a sample bug report with all necessary details
5. **Assumptions & Risks**
   - Mention if there are unclear or assumed behaviors in your test scope
   - Identify key risks if edge cases aren’t covered
6. **Checklist for Regression Impact**
   - List modules or flows that could be indirectly affected by this feature

Submission Instructions

Please submit your assignment either as:

- A GitHub repository (preferred), or
- A ZIP archive containing:
    - Diagrams (PDF, PNG, or draw.io)
    - API test files
    - Manual test documentation (`.docx`, `.xlsx`, or `.md`)
    - README file with overview and how to run/import