QualGent Backend Coding Challenge

Objective:

Build a CLI tool and GitHub Actions integration to queue, group, and deploy AppWright tests across local devices, emulators, and BrowserStack.

Scenario:

You are building an internal test infrastructure for a QA automation platform powered by AppWright. Organizations want to run end-to-end tests for various app versions across multiple device targets. To minimize install/setup overhead, tests targeting the same `app_version_id` should be grouped and scheduled together on the same device.

What You Need to Build:

CLI Tool

Build a CLI tool named `qgjob` that allows users to:

- Submit a test job:

Shell
```
qgjob submit --org-id=qualgent --app-version-id=xyz123
--test=tests/onboarding.spec.js
```

- Check status:

Shell
```
qgjob status --job-id=abc456
```

Requirements:

- Written in Go, Rust, Python, or Node.js.
- Should talk to a backend server using REST or gRPC.
- Must include a job payload schema with:
    - `org_id`, `app_version_id`, `test_path`, `priority`, `target` (emulator, device, browserstack).

Backend Service (Job Orchestrator)

Build a backend service (`job-server`) to:

- Receive and queue test jobs.
- Group jobs by `app_version_id` (to avoid reinstalling the app multiple times).
- Assign jobs to available agents (workers) based on device availability and `target`.
- Track job and run statuses.

Bonus for:

- Retry and failure handling
- Prioritization within orgs
- Horizontal scalability

Core Design Goals:

- **Modular:** separate concerns for queueing, scheduling, test execution.
- **Reliable:** crash recovery, fault-tolerance, job deduplication.
- **Scalable:** handle parallel orgs, devices, tests.

- **Efficient:** batch jobs by app version per device.

GitHub Actions Integration

Build a GitHub Actions workflow that:

- Runs the `qgjob` CLI during CI to submit tests.
- Polls for completion.
- Fails the build if any test fails.

Workflow file example:

```
None
name: AppWright Test
on: [push]

jobs:
  run-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: |
          pip install qgjob
          qgjob submit --org-id=qualgent --app-version-id=xyz123
--test=tests/onboarding.spec.js
```

Evaluation Criteria

Area	What We're Looking For
Architecture	Clear separation of concerns, modular service layers
Scalability	Can handle multiple orgs/app versions/test runners

Maintainability	Clean code structure, comments, environment setup
Efficiency	Grouping logic for `app_version_id`, batching to minimize installs
CLI UX	Intuitive commands, useful help, clear errors
GitHub Actions	End-to-end integration, good developer experience
Bonus	Horizontal scaling, monitoring endpoints, test retries

Deliverables

1. GitHub repo with:
    - CLI tool code
    - Backend service code
    - Example GitHub Actions workflow
2. `README.md` with:
    - Setup instructions (Docker if needed)
    - Architecture diagram (simple is fine)
    - How the grouping/scheduling works
    - How to run an end-to-end test submission
3. (Optional) Sample output logs of job processing

Tech Requirements

- Use **any language you're comfortable with** for backend and CLI.
- Use **Redis, PostgreSQL, or in-memory store** for queueing.

- AppWright tests can be example test scripts for this challenge.
- GitHub Actions should work end-to-end with your CLI.