techrar

Techrar Notification Service

1. Overview

Design and implement a multi-channel notification platform that enables merchants to send large volumes of push, SMS, e-mail, and WhatsApp messages. The service must expose a clean API for external systems, include a self-service web UI, and sustain high throughput without degradation in latency or reliability. You will also add scheduling capabilities and support bulk receiver uploads.

---

2. Functional Requirements

1. Channels & Providers:
    - Integrate at least one provider per channel (push notification, SMS, email, WhatsApp).
    - Support dynamic provider selection.
2. Merchant API:
    - REST endpoints to:
        - Send immediate notifications.
        - Create/cancel scheduled notifications.
        - Query delivery status.
    - Idempotency key on send endpoints.
3. Web UI:
    - Merchant sign-in.
    - Compose message templates.
    - Upload CSV of recipients (small sample).
    - List campaigns with status.
4. Scheduling & Queuing:
    - Schedule notifications.
    - Use a reliable job queue.
    - At-least-once delivery with retries.
5. Performance & Scalability:
    - Sustain reasonable API request rate and notification volume.
    - Acceptable latency for "send now."
    - Scalable service architecture.
6. Persistence:
    - Store merchant data, templates, schedules, delivery receipts.
7. Security:
    - HTTPS only.
    - Protect sensitive data.
    - Basic rate limiting.

techrar

3. Deliverables

Item	Details
Source Code	Well-structured repository; clearly separated backend, frontend, infra.
Documentation	README.md with setup, run, and load-test instructions.Design document covering architecture, data models, and scaling strategy (1–2 pages).OpenAPI / GraphQL schema.
Tests	Unit tests for core logic, integration tests for one full outbound flow.
Deployment Assets	Dockerfile(s) and docker-compose with sample values.
Sample Data	Minimal script or fixtures to create demo merchants and send sample notifications.

4. Assessment Criteria

1. **Architecture & Design**
    - Clear separation of concerns, appropriate patterns, and justified technology choices.
2. **Code Quality**
    - Readability, modularity, adherence to best practices, effective error handling.
3. **Performance & Reliability**
    - Evidence of meeting throughput and latency targets (load-test results or metrics screenshots).
4. **Completeness & Correctness**
    - Functional parity with stated requirements, graceful handling of edge cases.
5. **Tests & Tooling**
    - Coverage across critical paths; simple “one-command” setup for reviewers.
6. **Documentation & Communication**
    - Clarity, conciseness, and ability to articulate trade-offs.
7. **UI/UX (Weight lower than backend)**
    - Clean, intuitive interface demonstrating thoughtfulness rather than polish.

techrar

5. Extra (Stretch Goals)

Pick any you like; mention them explicitly in your design doc. They will earn bonus points but are **not** mandatory.

- Pluggable rule engine for audience segmentation (e.g., send to users with last-login > X days).
- Real-time WebSocket dashboard streaming live delivery updates.
- A/B testing framework for message variants.

---

6. Notes

- We believe that system design is a craft, so we expect that you would follow best practices in designing your system.
- This task is designed to assess a candidate's ability to build a robust and scalable backend system.
- You can use whatever language, framework, and datastore you want, but be careful.
    - You should follow best practices of that specific language/framework.
    - You should be able to justify your choices

Please allocate no more than **15–20 hours** over one week. Focus on depth in core areas rather than completing every stretch goal. We value well-reasoned design and clean execution over breadth. If you have any questions concerning the project or the implementation, do not hesitate to contact us.

We look forward to reviewing your solution. Good luck!