Real-Time Notification & Event Management Platform

Software Requirements Document (SRD)

Version: 1.0
Date: December 22, 2025
Status: MVP Definition
Document Type: Software Requirements & Technical Specification

---

Table of Contents

Executive Summary

This document defines the requirements for **Real-Time Notification & Event Management Platform**, an MVP-phase notification system designed to route, queue, and deliver notifications across multiple channels (email, SMS, push, in-app) to end users. The platform ingests events from external applications, processes them through a microservices architecture, and exposes real-time dashboards for monitoring, analytics, and user preference management.

Purpose: Enable other applications to reliably send notifications without building their own notification infrastructure.

Primary Users:

- End users: receive notifications and manage preferences
- Administrators: view system health, delivery rates, and error logs
- Developers: integrate their applications via REST/GraphQL/gRPC APIs

Learning Objective: Build a production-grade system that demonstrates full-stack development (frontend, backend, databases, queues, monitoring, containerization, authentication, cloud deployment patterns).

Project Description

What is this project?

A **notification platform** is a centralized service that manages the full lifecycle of notifications:

1. **Ingestion**: Accept events from other applications (e.g., "order placed", "password reset", "meeting reminder")
2. **Routing**: Determine which users should be notified and through which channels based on preferences
3. **Queuing**: Decouple notification requests from delivery to ensure reliability and scalability
4. **Processing**: Transform events into user-friendly messages using templates
5. **Delivery**: Send notifications through multiple channels (email initially, SMS/push later)
6. **Tracking**: Record delivery status, failures, and user engagement
7. **Observability**: Provide dashboards and metrics so operators can monitor system health

Why build this?

Real-world products (e-commerce, banking, SaaS, social platforms) send millions of notifications daily:

- Transactional: "Your order shipped", "Password changed", "2FA code"
- Marketing: "New offer", "Sale ending soon"
- Alerts: "Server down", "Unusual login attempt", "Payment failed"

This platform consolidates notification logic into a reusable service instead of duplicating it across applications. For you as a developer, it's a **complete learning project** that touches:

- API design (REST, GraphQL, gRPC, WebSockets)
- Event-driven architecture (asynchronous processing, message queues)
- Distributed systems (microservices, inter-service communication)
- Database design (relational & NoSQL)
- Observability (metrics, logging, tracing, alerting)
- DevOps (containerization, orchestration, CI/CD)
- Security (authentication, encryption, rate limiting)
- AI integration (smart scheduling, content personalization)

Objectives & Goals

Primary Objectives (MVP)

- **OBJ-1**: Accept events via REST API with minimal latency
- **OBJ-2**: Process and queue notifications reliably with zero message loss
- **OBJ-3**: Deliver email notifications within 5 seconds for 99% of cases
- **OBJ-4**: Provide end-user dashboard to view and manage notifications
- **OBJ-5**: Expose metrics and health dashboards for system operators
- **OBJ-6**: Support containerized deployment via Docker and docker-compose

- **OBJ-7**: Demonstrate authentication (JWT/OAuth), authorization, and role-based access control

Secondary Objectives (Post-MVP)

- **OBJ-8**: Support multiple notification channels (SMS, push, in-app, Slack, Teams)
- **OBJ-9**: Implement AI-powered send-time optimization and personalization
- **OBJ-10**: Deploy to Kubernetes with auto-scaling and self-healing
- **OBJ-11**: Implement advanced observability (distributed tracing, custom alerts)
- **OBJ-12**: Support webhook integrations for real-time event notifications

Success Metrics (MVP)

Metric	Target
API Response Time (p95)	< 200ms
Notification Delivery Latency (p99)	< 5 seconds
System Availability	99.5%
Failed Notification Recovery (retry success)	95%
Code Coverage	> 70%
Documentation Completeness	100% (README, API docs, deployment guide)

Scope

In Scope (MVP)

Features:

- User registration and login via email/password
- API endpoint to accept notification events
- Email template management and rendering
- Notification history and status tracking
- User preference management (email enable/disable)
- Role-based access control (Admin, User)
- Health check endpoints
- Prometheus metrics exposure
- Grafana dashboard for system monitoring
- Docker Compose for local development
- Basic logging to console/stdout

Supported Channels: Email only (SMTP simulation or real provider)

Data Persistence: PostgreSQL (relational), Redis (caching/sessions), Redpanda (queue)

Integrations: Keycloak for authentication, Prometheus for metrics

Out of Scope (Post-MVP)

- Multiple notification channels (SMS, push, in-app, webhook notifications)
- MongoDB analytics layer
- ELK Stack (Elasticsearch, Logstash, Kibana)
- Kubernetes deployment
- CI/CD pipeline (Concourse)
- GraphQL API (REST only for MVP)
- gRPC inter-service communication
- WebSocket real-time updates (polling suffices)
- Advanced AI features (send-time optimization, content generation)
- Multi-region deployment
- Internationalization/localization
- Advanced encryption features beyond TLS
- SonarQube code quality gates
- CDN integration

Exclusions & Limitations (MVP)

- **Rate Limiting**: Simple in-memory counter (not distributed Redis-based)
- **Message Retention**: Notifications older than 30 days are archived (not deleted immediately)
- **Scalability**: Designed to scale horizontally later; MVP runs on single nodes
- **Load Balancing**: Not included; nginx/load-balancer setup is out of scope
- **Disaster Recovery**: No backup/restore procedures defined for MVP
- **Multi-tenancy**: Single-tenant system; future versions can support multiple organizations

---

System Architecture Overview

High-Level Architecture Diagram

```
┌──────────────────────────────────────────────────────────────────────
───────────────┐
│ EXTERNAL APPS │
│ (e.g., E-commerce, Banking) │
└──────────────────────────────────────────────────────────────────────
───────────────┘
│
│ Events
▼
┌──────────────────────────────────────────────────────────────────────
───────────────┐
│ API GATEWAY (.NET) │
│
┌──────────────────────────────────────────────────────────────────────
─┐ │
```

```
│ │ Keycloak Auth │ REST Endpoints │ Rate Limiting │ │
│ │ /register │ /notifications │ Request Validation │ │
│ │ /login │ /preferences │ Error Handling │ │
│ │ /events │ /history │ │ │
│
└──────────────────────────────────────────────────────────────────
──┘ │
│ │ │
│ ▼ │
│ Publish to Queue │
└──────────────────────────────────────────────────────────────────
──────────┘
│
│ Notification Events
▼
┌──────────────────────────────────────────────────────────────────
──────────┐
│ REDPANDA (Message Queue) │
│ Topics: notifications, deadletter, events │
└──────────────────────────────────────────────────────────────────
──────────┘
│
│ Consume
▼
┌──────────────────────────────────────────────────────────────────
──────────┐
│ PYTHON WORKER SERVICE │
│
┌──────────────────────────────────────────────────────────────────
──┐ │
│ │ Consume from Queue │ Update Status │ Send Email │ │
│ │ Render Template │ Retry Logic │ Error Handling │ │
│ │ Provider Integration │ Metrics Emit │ Dead Letter Routing│ │
│
└──────────────────────────────────────────────────────────────────
──┘ │
└──────────────────────────────────────────────────────────────────
──────────┘
│
▼
┌─────────────────────────────────────┬────────────────────────────
──┐
│ │ │
▼ ▼ ▼
┌──────────────────────┐ ┌──────────────────────┐
┌──────────────────────┐
│ POSTGRESQL │ │ REDIS CACHE │ │ EMAIL PROVIDER │
│ │ │ │ │ (SMTP/SendGrid) │
│ Users │ │ Sessions │ │ │
│ Notifications │ │ Rate Limits │ │ (External Service) │
│ Templates │ │ User Prefs Cache │ │ │
│ Preferences │ │ │ │ │
└────────────────────────────────┘ └────────────────────────────────┘
└────────────────────────────────┘
▲ │
│ │
└──────────────────────────────────────────────────────────────────────────────────────────┘
Status Updates & Email Send

┌────────────────────────────────────────────────────────────────────────────────────────────
──────────────┐
│ MONITORING & OBSERVABILITY │
│ ┌─────────────────────────────┐ ┌────────────────────────────────┐ ┌──────────────────────┐ │
│ │ PROMETHEUS │ │ GRAFANA DASHBOARDS │ │ LOGS (STDOUT) │ │
│ │ /metrics │ │ Throughput │ │ Structured │ │
│ │ Counters │ │ Error Rates │ │ JSON │ │
│ │ Gauges │ │ Latency │ │ │ │
│ │ Histograms │ │ Queue Depth │ │ │ │
│ └─────────────────────────────┘ └────────────────────────────────┘ └──────────────────────┘ │
└────────────────────────────────────────────────────────────────────────────────────────────
──────────────┘

┌────────────────────────────────────────────────────────────────────────────────────────────
──────────────┐
│ USER INTERFACES │
│ ┌───────────────────────────────────┐ ┌───────────────────────────────────┐ │
│ │ User Dashboard │ │ Admin Dashboard │ │
│ │ (React) │ │ (React) │ │
│ │ ├─ Notifications │ │ ├─ System Health │ │
│ │ ├─ Preferences │ │ ├─ Delivery Stats │ │
│ │ ├─ History │ │ ├─ Error Logs │ │
│ │ └─ Profile │ │ ├─ Recent Failures │ │
│ │ │ │ └─ Configuration │ │
│ └───────────────────────────────────┘ └───────────────────────────────────┘ │
│ │
│ Connected via Keycloak Auth (JWT) │
└────────────────────────────────────────────────────────────────────────────────────────────
──────────────┘

┌────────────────────────────────────────────────────────────────────────────────────────────
──────────────┐
│ CONTAINERIZATION │
│ Docker Compose Orchestrates: │
│ ├─ api (port 5000) │
│ ├─ worker (background process) │
│ ├─ frontend (port 3000) │
│ ├─ postgres (port 5432) │
│ ├─ redis (port 6379) │
│ ├─ redpanda (port 9092) │
│ ├─ prometheus (port 9090) │
│ └─ grafana (port 3000) │
│ │
│ Single command: docker-compose up │
└──────────────────────────────────────────────────────────────────────────────────────────────────┘
```

Data Flow

Scenario: User Places Order

1. E-commerce app calls: `POST /api/events/order-placed` with `{ orderId: 123, userId: 456 }`
2. API Gateway (Keycloak auth) validates request
3. API looks up template "Order Confirmation" in PostgreSQL
4. API publishes message to Redpanda topic `notifications`
5. API saves row to `notification_history` with status `QUEUED`
6. API responds with `202 Accepted` (async acknowledgment)
7. Python worker consumes from `notifications` topic
8. Worker renders template: "Your order #123 has been placed"
9. Worker calls SMTP provider or SendGrid to send email
10. Worker updates `notification_history` status to `SENT`
11. Prometheus metrics increment: `notifications_sent_total`
12. React dashboard real-time polls and shows updated status
13. Grafana dashboard shows spike in throughput
14. User receives email within 5 seconds

Functional Requirements

1. User Management

Req ID	Requirement	Description	Priority
FR-1.1	User Registration	Users can register with email and password. System stores hashed password in PostgreSQL.	High
FR-1.2	User Authentication	Keycloak integration for JWT-based login/logout.	High
FR-1.3	Role-Based Access	System supports two roles: Admin and Regular User. Enforce permissions on API endpoints.	High
FR-1.4	User Profile	Users can view and update basic profile (name, email, timezone).	Medium
FR-1.5	Password Reset	Users can request password reset via email link.	Medium

2. Notification Core

Req ID	Requirement	Description	Priority
FR-2.1	Event Ingestion	Accept events via `POST /api/events/{eventType}` with JSON payload.	High
FR-2.2	Event Validation	Validate required fields; return 400 if invalid.	High
FR-2.3	Rate Limiting	Limit requests to 10 per minute per user. Return 429 if exceeded.	High
FR-2.4	Template Management	Admin can create, update, delete notification templates with variables.	High
FR-2.5	Template Rendering	System renders templates by substituting variables (e.g., `{userName}`, `{orderId}`).	High
FR-2.6	Queue Publishing	Validated notifications published to Redpanda topic.	High
FR-2.7	Status Tracking	Track notification status: QUEUED → SENT → DELIVERED/FAILED.	High
FR-2.8	Retry Logic	Failed notifications retry up to 3 times with exponential backoff.	High
FR-2.9	Dead Letter Queue	After 3 failed retries, move to DLQ for manual investigation.	Medium

3. Notification Preferences

Req ID	Requirement	Description	Priority
FR-3.1	Preference Toggle	Users can enable/disable email notifications globally.	High
FR-3.2	Per-Channel Preferences	Users can set preferences per notification type (e.g., marketing off, transactional on).	Medium
FR-3.3	Quiet Hours	Users can set quiet hours (e.g., no notifications 10 PM - 8 AM).	Low
FR-3.4	Preference Persistence	User preferences stored in PostgreSQL and cached in Redis.	High

4. Notification History & Dashboard

Req ID	Requirement	Description	Priority
FR-4.1	Notification History	Users see list of received notifications with date, subject, status.	High
FR-4.2	History Pagination	Support pagination (20 per page, sortable by date).	High
FR-4.3	Search & Filter	Filter by date range, notification type, status (delivered, failed).	Medium
FR-4.4	Admin Logs	Admin dashboard shows all system events with filters and timestamps.	High
FR-4.5	Error Logs	Detailed error messages and stack traces for failed notifications.	High

5. API Endpoints (REST)

Method	Endpoint	Purpose	Auth	Response
POST	/api/auth/register	User registration	Public	201 + JWT
POST	/api/auth/login	User login	Public	200 + JWT
POST	/api/auth/logout	User logout	Bearer Token	200
GET	/api/health	Health check	Public	200 {status: "ok"}
POST	/api/events/{eventType}	Publish event	Bearer Token	202 (async)
GET	/api/notifications	Fetch user notifications	Bearer Token	200 + array
GET	/api/notifications/{id}	Fetch single notification	Bearer Token	200 + object
PUT	/api/preferences	Update user preferences	Bearer Token	200
GET	/api/preferences	Fetch user preferences	Bearer Token	200
GET	/admin/metrics	Metrics summary (admin only)	Bearer Token + Admin	200
GET	/admin/logs	System logs (admin only)	Bearer Token + Admin	200
GET	/metrics	Prometheus metrics	Public (no auth)	200 text/plain

Note: POST/PUT/DELETE endpoints require Content-Type: application/json. All responses include error details on failure.

6. Worker Service

Req ID	Requirement	Description	Priority
FR-6.1	Queue Consumption	Worker consumes from Redpanda `notifications` topic.	High
FR-6.2	Email Rendering	Render template with user data to produce final email body.	High
FR-6.3	Email Sending	Integrate with SMTP provider (initially mock, later SendGrid/AWS SES).	High
FR-6.4	Status Update	Update notification status in PostgreSQL after each send attempt.	High
FR-6.5	Error Handling	Catch exceptions, retry with backoff, move to DLQ after 3 failures.	High
FR-6.6	Metrics Emission	Emit Prometheus metrics for sent, failed, retried counts.	High
FR-6.7	Graceful Shutdown	Worker drains queue and stops cleanly on SIGTERM.	Medium

---

Non-Functional Requirements

Performance

Req ID	Requirement	Target	Rationale
NFR-1.1	API Response Time (p95)	< 200ms	User experience; responsive API
NFR-1.2	Notification Delivery Latency (p99)	< 5 seconds	Most notifications are time-sensitive
NFR-1.3	Queue Processing Throughput	> 1000 notifications/second (MVP), scalable to 10k/s	Support growth without re-architecting
NFR-1.4	Database Query Time (p95)	< 100ms	Fast lookups for user data, preferences
NFR-1.5	Memory Usage per Service	< 500MB	Efficient containerization

Reliability & Availability

Req ID	Requirement	Target	Rationale
NFR-2.1	System Availability	99.5% uptime (SLA)	Enterprise-grade reliability
NFR-2.2	Message Durability	Zero message loss in queue	Critical for notifications
NFR-2.3	Retry Success Rate	95% of failed messages recover	Maximize delivery
NFR-2.4	Database Backup	Daily automated backups	Disaster recovery
NFR-2.5	Health Check Interval	Every 30 seconds	Quick failure detection

Scalability

Req ID	Requirement	Target	Rationale
NFR-3.1	Horizontal Scaling	Add worker instances to handle load	No single point of failure
NFR-3.2	Queue Partition Support	Redpanda partitions for parallel processing	Distribute load
NFR-3.3	Database Connection Pooling	Min 5, Max 20 connections	Efficient resource usage
NFR-3.4	API Load Balancing	Support multiple API instances	High availability

Security

Req ID	Requirement	Target	Rationale
NFR-4.1	Authentication	JWT + OAuth2 (Keycloak)	Industry standard
NFR-4.2	Authorization	Role-based access control (RBAC)	Principle of least privilege
NFR-4.3	Encryption in Transit	TLS 1.2+ for all connections	Data confidentiality
NFR-4.4	Password Hashing	bcrypt with salt	Prevent credential compromise
NFR-4.5	Rate Limiting	10 requests/minute per user	DDoS mitigation
NFR-4.6	Input Validation	Sanitize all user inputs	SQL injection, XSS prevention
NFR-4.7	Secrets Management	Environment variables, no hardcoded secrets	Secure deployments

Maintainability & Observability

Req ID	Requirement	Target	Rationale
NFR-5.1	Logging	Structured JSON logs to stdout	Easy parsing and aggregation
NFR-5.2	Metrics	Prometheus-compatible metrics	System health monitoring
NFR-5.3	Code Coverage	> 70% unit test coverage	Bug reduction
NFR-5.4	Documentation	README, API docs, deployment guide	Onboarding & maintainability
NFR-5.5	Error Messages	Clear, actionable error messages	Developer experience
NFR-5.6	API Versioning	Endpoint versioning (e.g., /v1/)	Backward compatibility

Compatibility & Deployment

Req ID	Requirement	Target	Rationale
NFR-6.1	Containerization	Docker images for all services	Consistent deployments
NFR-6.2	Local Development	docker-compose up	Quick setup for developers
NFR-6.3	CI/CD Ready	GitHub Actions workflows (future)	Automated testing & deployment
NFR-6.4	Environment Configuration	.env file for settings	Easy switching between environments
NFR-6.5	Operating System Support	Linux, macOS, Windows (via Docker)	Developer flexibility

Technology Stack

Frontend

- **React 18+** with TypeScript
- **Axios** for HTTP client
- **React Router** for navigation
- **Tailwind CSS** or Material UI for styling
- **Chart.js** or Recharts for dashboard visualizations

Backend / API Gateway

- **.NET 7+ (C#)** with ASP.NET Core Web API
- **Entity Framework Core** for ORM
- **Keycloak** for authentication/authorization (OAuth2/JWT)
- **Swagger/OpenAPI** for API documentation

Worker Service

- **Python 3.9+** with FastAPI or Celery
- **psycopg2** for PostgreSQL connection
- **redis-py** for Redis operations
- **confluent-kafka** for Redpanda producer/consumer
- **Jinja2** for template rendering
- **smtplib** or **requests** for email provider integration

Databases

- **PostgreSQL 13+** for relational data (users, notifications, templates)
- **Redis 7+** for caching and session storage
- **Redpanda** (Kafka-compatible) for message queue

Monitoring & Observability

- **Prometheus** for metrics collection
- **Grafana** for visualization and dashboards
- **stdout (JSON logs)** for centralized logging

Authentication & Authorization

- **Keycloak** for identity management (OIDC/OAuth2)
- **JWT** tokens for stateless authentication

DevOps & Deployment

- **Docker** for containerization
- **Docker Compose** for local orchestration
- **.env** files for configuration management

Development Tools

- **Git/GitHub** for version control
- **Postman** or **curl** for API testing
- **pgAdmin** for database management (optional)
- **VS Code** and **PyCharm** for development

MVP Checklist

Phase 1: Project Setup & Infrastructure (Week 1)

- [ ] Initialize Git repository with main branch
- [ ] Create project structure:
  - [ ] `/api` folder (.NET project)
  - [ ] `/worker` folder (Python project)
  - [ ] `/frontend` folder (React project)
  - [ ] `/infra` folder (Docker, docker-compose files)
  - [ ] `/docs` folder (README, API docs)
- [ ] Create `.env.example` and `.env.local` files with required variables
- [ ] Write initial README with project overview and setup instructions
- [ ] Create `docker-compose.yml` skeleton with all services (no implementation yet)
- [ ] Initialize GitHub Actions workflow file (empty for now)

Deliverable: Git repo structure, docker-compose skeleton, README

Phase 2: Database Schema & Core API (Week 1-2)

2.1 PostgreSQL Schema

- [ ] Create `users` table (id, email, password_hash, created_at, updated_at, is_active)
- [ ] Create `notification_templates` table (id, name, subject, body, variables, created_by, created_at)
- [ ] Create `notification_history` table (id, user_id, template_id, status, sent_at, created_at, error_message)
- [ ] Create `user_preferences` table (id, user_id, email_enabled, created_at, updated_at)
- [ ] Create database migrations/seed script
- [ ] Create ERD (Entity Relationship Diagram) document

2.2 .NET API Setup

- [ ] Create ASP.NET Core Web API project structure
- [ ] Configure Entity Framework Core with PostgreSQL
- [ ] Create DbContext and Models
- [ ] Implement health check endpoint: `GET /api/health` → `{ "status": "ok" }`
- [ ] Test locally with `dotnet run`

2.3 Authentication (Keycloak)

- [ ] Set up Keycloak instance (via Docker)
- [ ] Create realm and client in Keycloak
- [ ] Generate client credentials (client_id, client_secret)
- [ ] Create /api/auth/register endpoint (basic email/password)
- [ ] Create /api/auth/login endpoint (return JWT token)
- [ ] Add Bearer token validation middleware to API

Deliverable: Working API with auth, database connected, health check functional

---

Phase 3: User Management & Preferences API (Week 2)

- [ ] Implement GET /api/preferences endpoint
- [ ] Implement PUT /api/preferences endpoint (toggle email enabled/disabled)
- [ ] Implement GET /api/user/profile endpoint
- [ ] Add role-based authorization (require Bearer token on protected endpoints)
- [ ] Add Swagger documentation to all endpoints
- [ ] Write unit tests for auth and preference endpoints (target 70%+ coverage)
- [ ] Add error handling and validation

Deliverable: User management API complete, documented, tested

---

Phase 4: Notification Queuing with Redpanda (Week 2-3)

4.1 Redpanda Setup

- [ ] Set up Redpanda in docker-compose (broker)
- [ ] Create Redpanda topic notifications
- [ ] Create Redpanda topic notifications-dlq (dead letter queue)
- [ ] Test Redpanda connectivity with producer/consumer CLI

4.2 .NET API Changes

- [ ] Integrate Confluent.Kafka NuGet package
- [ ] Create Redpanda producer service
- [ ] Implement POST /api/events/{eventType} endpoint:
    - Validate event payload
    - Check rate limiting (10 req/min per user)
    - Save to notification_history with status QUEUED
    - Publish to Redpanda topic
    - Return 202 Accepted
- [ ] Implement GET /api/notifications endpoint (fetch history for user)
- [ ] Add Redpanda connection pooling and error handling

Deliverable: Event ingestion API with queue integration

Phase 5: Python Worker Service (Week 3)

5.1 Worker Setup

- [ ] Create Python project with FastAPI/Flask (for metrics endpoint)
- [ ] Install dependencies: `confluent-kafka`, `psycopg2`, `redis`, `jinja2`
- [ ] Create Redpanda consumer that polls `notifications` topic

5.2 Email Processing

- [ ] Implement template rendering with Jinja2
- [ ] Implement SMTP mock/integration (initially just log to console)
- [ ] Implement status update logic (mark as `SENT` or `FAILED` in PostgreSQL)
- [ ] Implement retry logic (retry up to 3 times with exponential backoff)
- [ ] Implement dead letter queue routing (move to DLQ after 3 failures)
- [ ] Add structured logging (JSON to stdout)

5.3 Error Handling

- [ ] Catch exceptions and log them
- [ ] Update `notification_history.error_message` on failure
- [ ] Emit metrics for Prometheus

Deliverable: Working worker that consumes from queue and processes notifications

---

Phase 6: React Frontend - User Dashboard (Week 3-4)

6.1 Setup & Auth

- [ ] Create React app with TypeScript (`npx create-react-app frontend --template typescript`)
- [ ] Install: `axios`, `react-router-dom`, `tailwindcss` (or Material UI)
- [ ] Create login/register pages
- [ ] Store JWT token in localStorage
- [ ] Create private route wrapper (redirect to login if no token)

6.2 User Dashboard

- [ ] Create main dashboard page
- [ ] Implement notification history table (paginated, sortable by date)
- [ ] Implement preferences toggle (enable/disable email notifications)
- [ ] Implement profile view/edit
- [ ] Add "Send Test Notification" button for testing
- [ ] Add loading and error states
- [ ] Implement real-time polling of notifications (every 5 seconds)

Deliverable: Working React UI for users to manage notifications

---

Phase 7: Admin Dashboard & Metrics (Week 4)

7.1 React Admin Dashboard

- [ ] Create admin route and page (accessible only to Admin role)
- [ ] Display key metrics:
  - Total notifications today
  - Delivery success rate (%)
  - Failed notifications count
  - Average delivery latency
- [ ] Display recent failures table with details
- [ ] Add search/filter for logs

7.2 Prometheus & Grafana Integration

- [ ] Add Prometheus metrics to .NET API:
  - `notifications_requests_total` (counter)
  - `notifications_sent_total` (counter)
  - `notifications_failed_total` (counter)
  - `notification_processing_duration_ms` (histogram)
  - `queue_depth` (gauge)
- [ ] Expose `/metrics` endpoint (public, no auth required)
- [ ] Add Prometheus metrics to Python worker (same counters/gauges)
- [ ] Set up Prometheus scrape targets in docker-compose
- [ ] Create Grafana dashboard with 4 panels:
  - [ ] Notifications sent per minute (line chart)
  - [ ] Error rate % (gauge)
  - [ ] Queue depth (line chart)
  - [ ] API response time p95 (line chart)

Deliverable: Admin dashboard + monitoring stack working

Phase 8: Docker & Docker Compose (Week 4)

- [ ] Create `Dockerfile` for .NET API service
- [ ] Create `Dockerfile` for Python worker service
- [ ] Create `Dockerfile` for React frontend (multi-stage build)
- [ ] Create `docker-compose.yml` orchestrating:
  - [ ] api (port 5000)
  - [ ] worker (no external port, background service)
  - [ ] frontend (port 3000)
  - [ ] postgres (port 5432)
  - [ ] redis (port 6379)
  - [ ] redpanda (port 9092)
  - [ ] prometheus (port 9090)
  - [ ] grafana (port 3000 - change to 3001 to avoid conflict with frontend)
  - [ ] keycloak (port 8080)
- [ ] Write .dockerignore files for each service
- [ ] Test: `docker-compose up` should start entire stack in one command
- [ ] Add health checks to docker-compose services
- [ ] Write startup script to wait for all services to be ready

Deliverable: Full stack runs with one `docker-compose up` command

Phase 9: Integration Testing & Documentation (Week 4-5)

9.1 End-to-End Testing

- [ ] Write test script that:
  - [ ] Registers a new user
  - [ ] Logs in and gets JWT
  - [ ] Sends a test notification via API
  - [ ] Waits 5 seconds
  - [ ] Fetches notification history
  - [ ] Verifies notification status is `SENT`
  - [ ] Checks Grafana metrics updated
- [ ] Test via curl/Postman with example payloads

9.2 Documentation

- [ ] Update README with:
  - [ ] Project overview (what it does, why, architecture diagram)
  - [ ] Quick start guide (git clone, docker-compose up)
  - [ ] Environment variables reference
  - [ ] API documentation (all endpoints, examples, responses)
  - [ ] Architecture explanation (high-level components)
  - [ ] Developer guide (how to add new notification types, extend worker)
- [ ] Create API documentation file (Swagger/OpenAPI spec)
- [ ] Create database schema documentation (ERD)
- [ ] Create deployment guide (how to deploy to cloud)

9.3 Code Quality

- [ ] Ensure > 70% unit test coverage
- [ ] Run linters (.NET, Python, React)
- [ ] Add pre-commit hooks for code formatting
- [ ] Write comments for complex logic

Deliverable: Complete, tested, documented MVP

Phase 10: Final Testing & Demo Preparation (Week 5)

- [ ] Run full end-to-end test on fresh docker-compose setup
- [ ] Verify all APIs work via Postman collection
- [ ] Create demo script showing:
  - [ ] User registration and login
  - [ ] Sending notification and seeing it in history
  - [ ] Admin viewing metrics and dashboards
  - [ ] Worker processing queue and sending email
  - [ ] Grafana showing throughput
- [ ] Load test (simple stress test with 100+ concurrent notifications)
- [ ] Verify error handling (simulate failures, retry logic)
- [ ] Create repository clean (remove build artifacts, test data)
- [ ] Write CONTRIBUTING.md for future phases

Deliverable: Production-ready MVP, ready for demo/interview

Assumptions & Constraints

Assumptions

1. Users have valid email addresses for receiving notifications
2. External email provider (SMTP) is configured and available
3. Keycloak instance is running and accessible during development
4. PostgreSQL and Redis are accessible within Docker network
5. Redpanda cluster has sufficient disk space and memory for queue buffering
6. Network latency between services is < 100ms
7. Single-region deployment (no multi-region failover)
8. Docker and docker-compose are installed on developer machines

Constraints

1. **MVP Scope:** Email channel only; SMS/push/in-app excluded
2. **Scalability:** MVP designed for <1000 notifications/minute; horizontal scaling via worker replicas
3. **Storage:** 30-day retention for notification history (configurable)
4. **Authentication:** Single identity provider (Keycloak); LDAP/SAML support deferred
5. **Deployment:** Docker Compose for MVP; Kubernetes for production (future)
6. **Cost:** Free/open-source services preferred (PostgreSQL, Redis, Redpanda, Keycloak)
7. **Performance:** API response time target is p95 < 200ms with current infrastructure
8. **Compliance:** MVP does not address GDPR/CCPA; to be added in later phases
9. **Support:** Single-tenant; multi-tenancy support deferred to Phase 2
10. **Testing:** Manual and automated tests; CI/CD pipeline deferred to Phase 2

Revision History

Version	Date	Author	Changes
1.0	2025-12-22	Architecture Team	Initial MVP requirements document

---

Sign-Off

Document Owner: Development Team
Review Date: 2025-12-22
Approval Status: Draft (Ready for Review)

Appendices

Appendix A: API Request/Response Examples

Example 1: Register User

```
POST /api/auth/register
Content-Type: application/json

{
"email": "user@example.com",
"password": "SecurePass123!"
}

Response: 201 Created
{
"id": "uuid",
"email": "user@example.com",
"token": "eyJhbGciOiJIUzI1NiIs..."
}
```

Example 2: Send Notification Event

```
POST /api/events/order-placed
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...
Content-Type: application/json

{
"userId": "456",
"orderId": "ORDER-123",
"orderTotal": "99.99",
"estimatedDelivery": "2025-12-24"
}

Response: 202 Accepted
{
"notificationId": "notification-uuid",
"status": "QUEUED",
"message": "Notification queued for processing"
}
```

Example 3: Get User Notifications

```
GET /api/notifications?page=1&limit=20
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...

Response: 200 OK
{
"data": [
{
"id": "notif-1",
"templateName": "Order Confirmation",
"subject": "Order #ORDER-123 Confirmed",
"status": "SENT",
"sentAt": "2025-12-22T10:30:00Z",
```

"body": "Your order has been placed..."
}
],
"total": 45,
"page": 1,
"pageSize": 20
}

Appendix B: Notification Template Example

Template Name: Order Confirmation
Subject: Your Order #{{orderId}} Has Been Placed

Body:
Dear {{userName}},

Thank you for your order! Here are the details:

Order ID: {{orderId}}
Total: ${{orderTotal}}
Estimated Delivery: {{estimatedDelivery}}

Track your order: https://example.com/orders/{{orderId}}

Questions? Contact support@example.com

Best regards,
The Store Team

Appendix C: Environment Variables

Database

POSTGRES_USER=notification_user
POSTGRES_PASSWORD=SecurePassword123
POSTGRES_DB=notification_db
POSTGRES_HOST=postgres
POSTGRES_PORT=5432

Redis

REDIS_HOST=redis
REDIS_PORT=6379

Redpanda

REDPANDA_BROKERS=redpanda:9092
REDPANDA_TOPIC_NOTIFICATIONS=notifications
REDPANDA_TOPIC_DLQ=notifications-dlq

Keycloak

KEYCLOAK_URL=http://keycloak:8080
KEYCLOAK_REALM=notification-platform
KEYCLOAK_CLIENT_ID=notification-api
KEYCLOAK_CLIENT_SECRET=your-secret-here

API

API_PORT=5000
API_ENVIRONMENT=development
API_LOG_LEVEL=info

Worker

WORKER_CONCURRENCY=5
WORKER_RETRY_MAX=3
WORKER_RETRY_DELAY_MS=1000

Email

EMAIL_PROVIDER=smtp
SMTP_HOST=smtp.mailtrap.io
SMTP_PORT=2525
SMTP_USER=your-user
SMTP_PASSWORD=your-pass
SMTP_FROM=noreply@notification-platform.com

Prometheus

PROMETHEUS_PORT=9090

Grafana

GRAFANA_ADMIN_PASSWORD=admin
GRAFANA_PORT=3000
