notifyengine.io

AGENDA

1. Team Roles
2. Problem Statement
3. Project Description
4. Project Schedule
5. Personas
6. Technologies
7. Algorithms
8. MVP Definition
9. Team Working Agreement
10. Retrospective
11. AI Disclosure

TEAM ROLES

Mohammed N. Yusif

Security Engineer/ Project Manager

Krish Dhorajiya

AI Engineer / Tester

Rhythm Patel

AI Engineer / Developer

TEAM ROLES

Salvatore Ardisi

SCRUM Master / Developer

Dhawalshree Mengane

Data Analyst / Developer

Zeeya Ramani

Data Analyst / Developer

WHAT PROBLEM ARE WE SOLVING?

- Every app guesses which channel to use: (email, SMS, push, or in-app) and guesses wrong constantly
- Static “email-first” rules never adapt, even when user behavior completely changes
- No visibility: when a user says “I never got that notification” support teams have zero way to investigate
- Developers waste hours writing and maintaining manual routing logic that breaks when behavior shifts

NotifyEngine replaces guesswork with an ML model that learns each user's real behavior, automatically routing every notification to the channel they'll actually engage with

One API call. No routing rules to write. No providers to wire up separately. The system handles everything and gets smarter with every notification it sends

PROJECT DESCRIPTION

What

An ML-powered notification delivery service that learns from user behavior and automatically routes notifications to each user's most engaging channel (sms, email, push and in app channels).

Who

For developers and product teams who send high-volume user notifications across different channels.

Why

Unlike static notification tools like Courier, Knock, or Novu that use fixed rules, and enterprise platforms like Braze or MoEngage that are expensive, NotifyEngine offers a simple developer-first API, per-user channel routing that continuously improves with every notification sent.

Benefits

Higher notification engagement rates, lower SMS spend, automatic failover during provider outages, and zero manual routing logic to maintain.

PROJECT SCHEDULE

Detailed sprint timeline and milestone tracking

Sprint 0

Weeks 1-3

Planning, Architecture, Team Setup

Sprint 1

Weeks 4-6

Static Foundation & E2E Delivery

Sprint 2

Weeks 7-9

ML Integration & Adaptive Routing

Sprint 3

Weeks 10-12

Circuit Breaker & Dashboard

PERSONAS

Sarah Kim
Backend Developer

Series A SaaS (75K users)

Sarah is responsible for integrating email, SMS, and push providers. She currently manages SendGrid, Twilio, and Firebase separately. When a provider fails, she must manually reroute notifications.

Pain Points

- Multiple provider integrations
- No automatic fallback during outages
- Manual rerouting during incidents
- Complex maintenance overhead

Maps to NotifyEngine

- Unified API
- Circuit breaker pattern
- Automatic channel fallback
- Reduced engineering overhead

Daniel Ross
Product Manager

Fintech app (500K users)

Daniel notices engagement rates declining over time. Email open rates have dropped significantly, but routing logic hasn't been updated in over a year.

Pain Points

- Declining engagement rates
- Static routing rules
- No behavioral insights
- No visibility into channel performance

Maps to NotifyEngine

- XGBoost adaptive routing
- Real-time engagement dashboard
- Model explainability
- Automatic learning from outcomes

Amanda Lopez
Growth Lead

E-commerce platform (2M users)

Amanda spends heavily on SMS notifications because that has always been the default channel. She suspects cheaper channels could perform similarly.

Pain Points

- High SMS delivery cost
- No cost optimization strategy
- Engineering dependency for testing
- No performance breakdown by channel

Maps to NotifyEngine

- Cost-aware routing
- Channel performance analytics
- Usage-based pricing alignment
- Zero manual configuration

TECHNOLOGIES

Languages & Frameworks

FastAPI · TypeScript · Express · Python · React · Node.js · Vite

ML & Data Science

XGBoost · Scikit-Learn · Pandas

Databases & Infrastructure

PostgreSQL · Redis · Socket.io · BullMQ

User Interface

Tailwind CSS · shadcn/ui

Tools

GitHub · Jira · Slack · VS Code · Claude AI

Testing

Jest · Pytest

DevOps & CI/CD

Docker · GitHub Actions

Validation

Zod · Pydantic

ALGORITHMS

XGBoost Gradient Boosted Decision Tree

- Supervised ML model trained on notification delivery outcomes (delivered, opened, clicked, dismissed, ignored)
- Input features: channel type, time of day, day of week, hours since last notification, notification count, user engagement history, channel-specific open rates
- Output: probability of engagement per channel (highest probability wins)

Epsilon-Greedy Exploration

- 90% of the time (exploit): route to the channel the model predicts will perform best
- 10% of the time (explore): randomly try a different channel to discover new patterns
- Prevents the model from getting stuck (if a user changed behavior, exploration detects it)

Circuit Breaker Pattern

- Monitors delivery failures per provider in real-time
- If failure rate exceeds threshold -> circuit opens -> traffic reroutes to next best channel
- Automatically retries the failed provider after a cooldown period

API TO PREDICTION TO LEARNING

1

Ingest

Request validates via Zod; API Return 202 status.

2

Predict

BullMQ passes job to FastAPI ML service for score calculation.

3

Execute

System selects Adaptive or Explore channel and triggers delivery.

4

Track

Engagement captures open and clicks.

5

Learn

Logs feedback to model to update feature importance metrics.

MVP DEFINITION

Unified Multi-Channel API

A single endpoint handling email, push, SMS, and in-app notification. Developer focus on content, not channel selection logic.

ML-Powered Adaptive Routing

XGBoost model predicts the high-probability channel per user based on engagement history and specific time-of-day patterns.

Epsilon-Greedy Strategy

90% exploit (optimal path) and 10% explore (discovery) to ensure model stays current with user behavior.

Static Routing Baseline

Priority-based fallback (email -> push -> SMS) serving as a benchmark to measure ML accuracy gains.

MVP SCOPE & ENGINEERING PRIORITIES

Feature Area	What's In (The Engine)	What's Out (Post-MVP)
Reliability	Async Job Queue: Redis + BullMQ	User Signup/Auth UI
Visibility	Real-Time Dash & ML Explanations	Marketing Template Editor
Data Cycle	Tracking Pixels & WS Acknowledgement	Real SMS (Mocked for routing)
Business	Multi-Tenancy Isolation	Billing & Subscription Tier

ROADMAP & SUCCESS

The “Pass” Test: Success requires different users receiving channels based on data, with 100% of acceptance tests passing (across 14 criteria)

Sprint 1

1

Static foundation and end-to-end delivery functional.

Sprint 2

2

ML Integration and adaptive routing benchmark testing.

Sprint 3

3

Circuit Breaker failover and dashboard finalization.

TEAM WORKING AGREEMENT

- Defines team expectations, responsibilities, and professional standards for the NotifyEngine Capstone Project
- Ensures smooth collaboration through structured communication using WhatsApp (urgent updates) and Slack (organized channels + discussions)
- Supports accountability with daily standups, progress updates, and a team rule to respond within 24 hours
- Uses Jira for task assignment, sprint tracking, backlog management, and monitoring deliverables
- Follows Agile Scrum methodology with sprint planning, reviews, and retrospectives
- Maintains quality through testing, peer reviews, documentation, and timely GitHub updates
- Encourages respectful conflict resolution and promotes a professional, inclusive team environment

RETROSPECTIVE

What Went Well

- Strong team communication using Slack channels for organized discussion and updates
- A skilled team member introduced Jira + Slack, improving task tracking and keeping the team aligned
- Team members reported issues clearly, so others could help immediately
- Tasks were distributed fairly based on strengths and responsibilities
- Collaboration was effective during problem-solving, coding, and debugging

What Could Be Improved

- Some progress updates were delayed, making tracking harder in real time
- Scheduling meetings was difficult since the team was remote initially
- As it was our first time working together, it took time to break the ice
- The 24-hour reply agreement helped, but faster updates would improve sprint flow

Lessons Learned

- Slack channels reduce confusion and keep discussions structured
- Jira improves tracking for tasks, deadlines, and sprint progress
- Clear planning and task ownership reduce delays
- Quick issue reporting is essential for remote teamwork

Action Items for Sprint 1

- Kick Off Development Tasks: Start Sprint 1 user stories from Sprint 0 with clear task ownership
- Track Progress & Communicate: Update Jira regularly and use Slack for blockers; reply within 24 hours
- Break Down Tasks & Review: Split complex tasks and hold short check-ins to adjust priorities

AI DISCLOSURE

Tools Used

Claude (Anthropic) - claude.ai & Claude Code CLI
ChatGPT (OpenAI) - research and concept exploration

How We Use AI

- Research & understanding unfamiliar technologies
- Architecture & design tradeoff analysis
- Code generation with mandatory human validation
- Documentation & communication drafting

How We Do NOT Use AI

- AI does not make architectural decisions (it presents options, we decide)
- No code is merged unless the author can explain it line by line
- AI is not used for peer reviews (per syllabus policy)

Guardrails

- Mandatory human validation checkpoints
- All AI-reported metrics independently verified by human

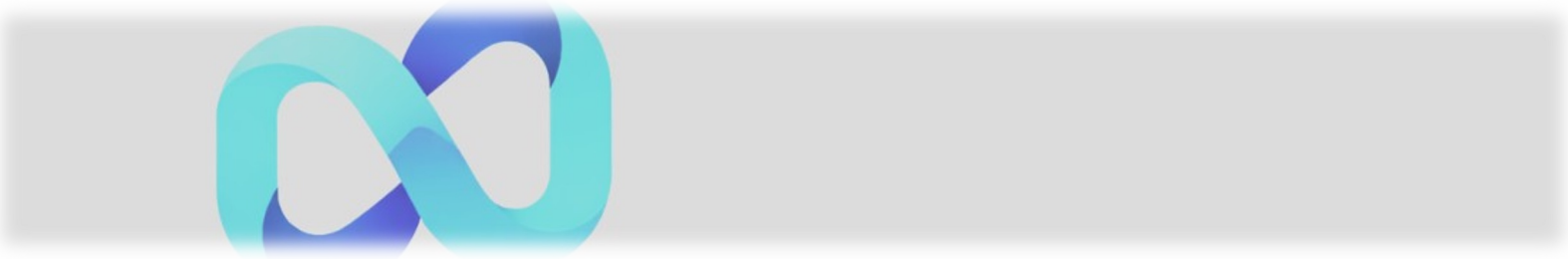
Why AI

- Reflects industry practice
- Accelerates learning, AI explains, team member implements and verifies

WIKI PAGE

Project Wiki & Documentation

https://github.com/htmw/2026S-Algo-Rhythms/wiki



THANK YOU.