Courier

www.courier.com

The State of Notifications

Report 2022

Contents

Introduction

Notifications kind of suck

What started out as a novel way for applications to interact with their users has turned into a congested flood of pop-ups that our eyes glaze over. Notifications are everywhere. They appear on every device across all applications, constantly vying for our attention, churning out messages from the most trivial of updates to mission critical alerts.

After years of oversaturating us with irrelevant and untimely spam, users have very little trust in applications to respect their time and attention. Send one too many notifications and within a few seconds, your app is muted if not uninstalled altogether.

But the answer isn't to cut notifications from the UX altogether. We know that, if done right, notifications can actually improve the UX and drive further retention. So why the disconnect? What is preventing us from designing a notification system that facilitates a better experience?

We, at Courier set out to answer this question in hopes of a future in which notifications delight rather than distract. In this industry snapshot, we'll cover:

- Where notifications fit into the communication barrier between users and applications
- What constitutes good notification UX
- How to design a notification system that supports good UX
- What technical requirements must be met to support the design
- Prospective industry predictions

User interface error

At their core, notifications serve to communicate information to the user, be that an update to an Asana task, a recent like on an Instagram post, or a discount code for an in-app purchase. But applications do not communicate with people the same way that we communicate with each other. Person to person communication has evolved over millennia into a sophisticated ritual. It's no wonder that an unrefined notification system irritates us. They do not reflect the organic style of communication that we unconsciously use every day.

IDEO is arguably one of the most well known design firms in the world. Their *Human Centered Design toolkit* popularized a problem-solving technique that placed humans at the center of product development. In the years afterwards, product owners and designers would come to embrace its axioms, calling for products to descriptively adapt to human behavior instead of forcing humans to prescriptively adapt to the product's design.

We spend so much of our time designing a delightful core product experience, carefully mapping out the user journey and obsessing over minor details that adapt to how we anticipate our customers will use our products. But once it's time to ship, we slap on a notification system and go out to celebrate.

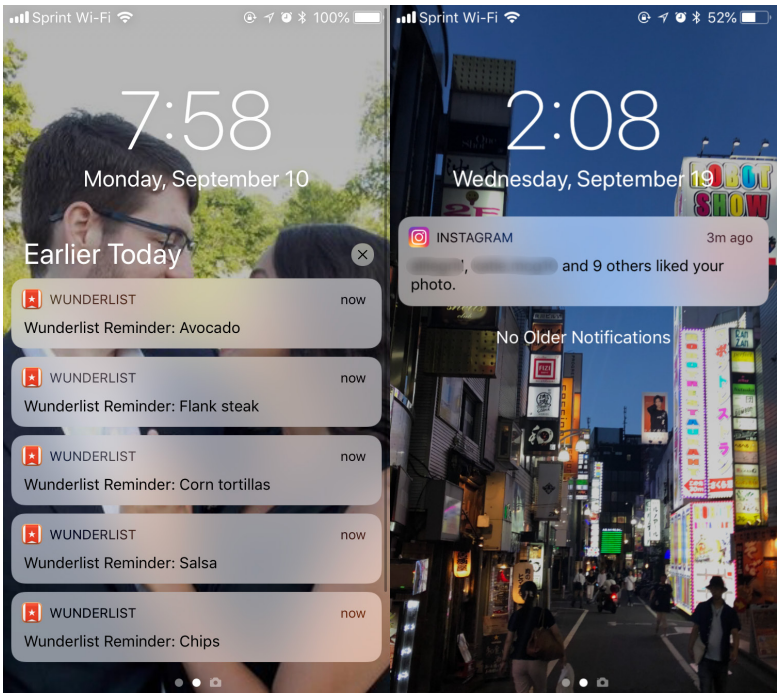
Notifications are a user-facing component. Not only do notifications interact directly with the user, they account for a large portion of user touchpoints. New users may install your app and have a memorable first-time experience, but it will be notifications that bring that user back into the app. If those notifications are not designed intentionally, users may churn regardless of how worthwhile or lovely the core product is.

> *This insight lies at the core of the disconnect between notifications and their audience.* ***Notifications are not designed to communicate with us the way we communicate with each other.***

At first glance, we may reduce our communication behavior to just speech, but it's much more sophisticated. Over thousands of years, we've evolved to recognize subtleties. We cue in on body language to understand another person's emotional state. We can read between the lines with a tonal shift of the voice. We know when a conversation is best had in person rather than on the phone.

Human communication is rich in context. We account for social norms, intimacy, cultural nuances, group dynamics, and a hundred other data points to inform how we communicate with another person. It's how we know to wait until the morning to tell our partner about a vivid dream and not the middle of the night and that a difficult conversation should be had in person and not in a flurry of texts.

This is the reason that we get frustrated at notifications. Compared to how we organically communicate with each other, notifications are an artless swarm of noisy alerts. That's why bad UX is so easy to spot - it does not conform to our standard of communication:



Source

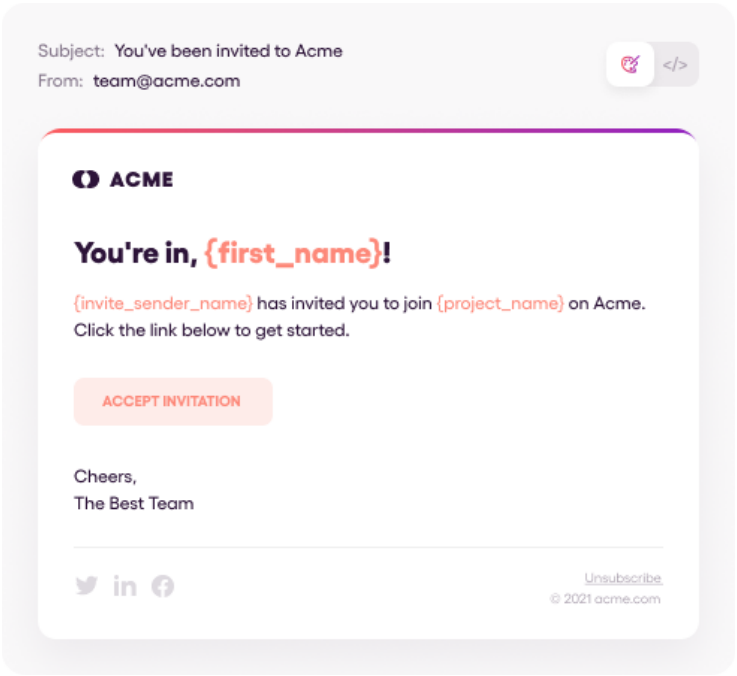
Unbatched notifications clutter our feed. If we need to communicate the same pattern of information to another person, we batch it. That's why we create grocery lists and to-do boards instead of blurting out "We need eggs."

Source

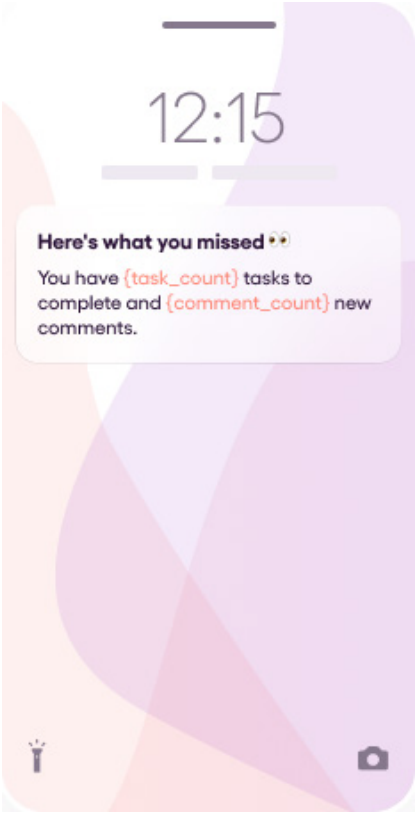
Poorly timed and irrelevant notifications can be disruptive. We have an intuitive sense of what and when to bring up a topic. It's why we send memes to a select group of people and why we don't have cognitively intense discussions as soon as we wake up.

Notifications spammed across multiple communication channels can be tedious. If we're confirming plans with a friend for the evening, we know a text is sufficient. We don't need to shoot a text, leave a voicemail, and send an email all at the same time.

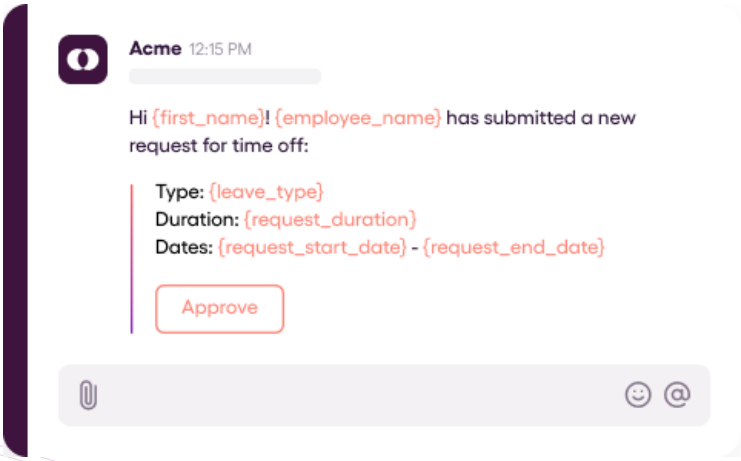
Good UX merely rises to the baseline standard of communication that we expect from others, while bad UX stands out like a sore thumb. And after years of sloppy design, our tolerance for indiscriminate notifications is low. We have a short probationary period to prove that our applications will respect our users' time and attention. Violate that trust and your app gets muted, cutting off a critical line of communication between you and your user.



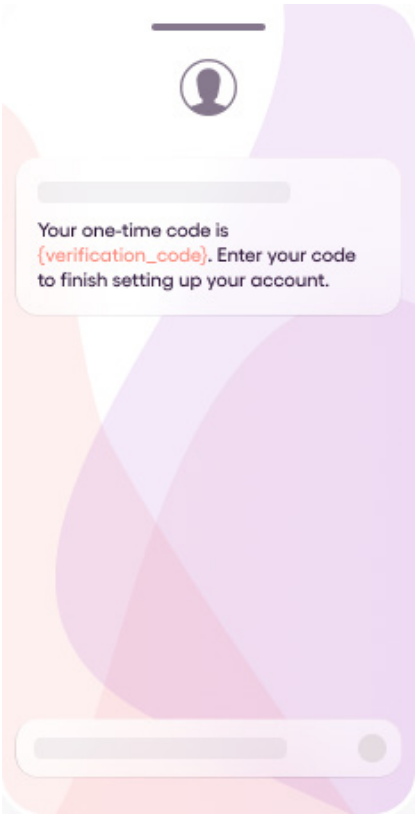
Email Notification



Push Notification



Slack Notification



SMS Notification

Modern notification design

When done properly, notifications feed relevant information to the user, triggering them to interact with the application driving further downstream notifications and future engagement. Good notification UX results in a snowball effect within your application. To achieve this outcome, notification delivery must emulate human standards of communication by using context to attract the user's attention at just the right time. Applications must be able to answer questions like:

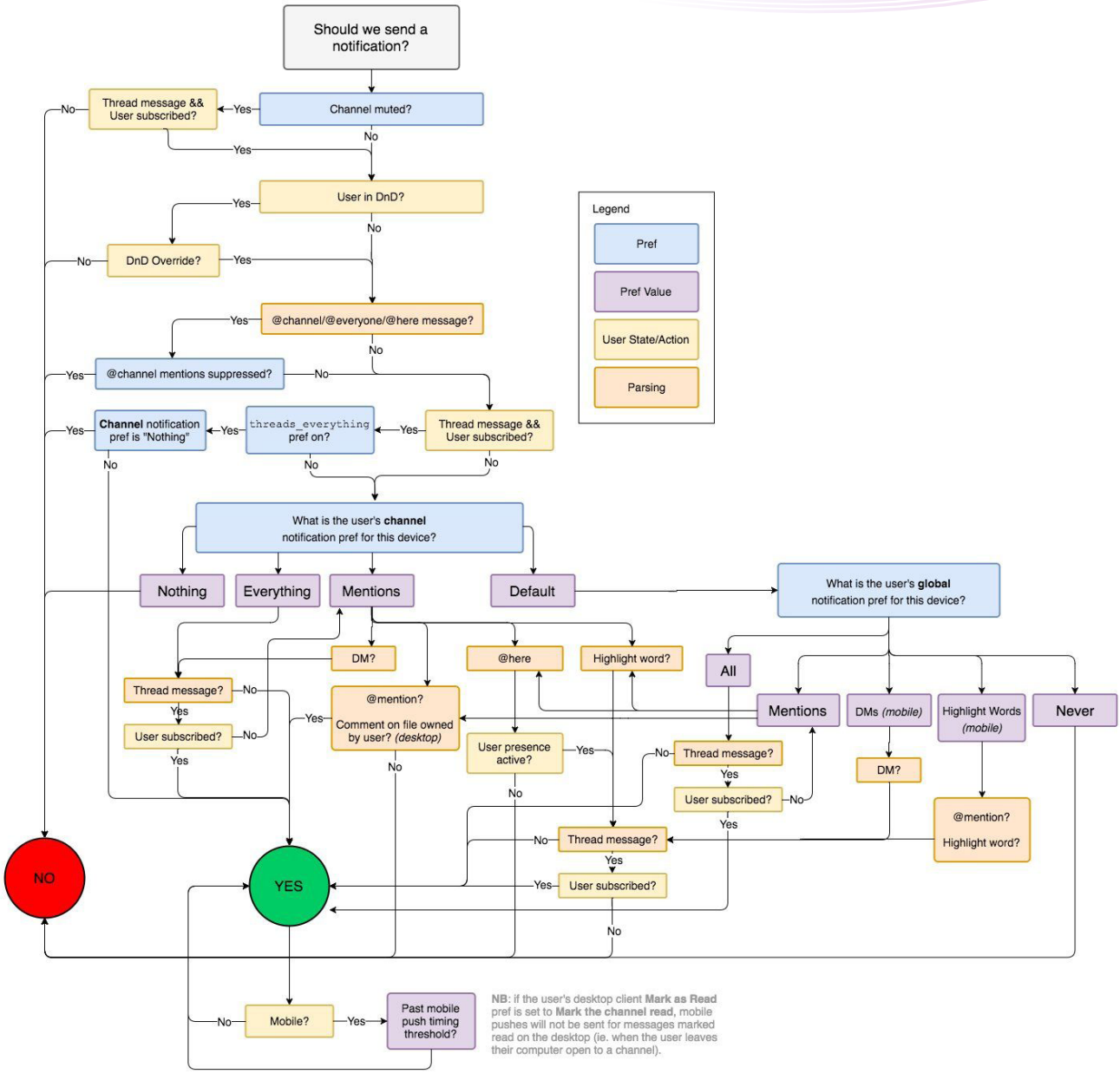
- **Content** - What goes into this notification?
- **Target** - Who does it go to?
- **Timing** - When does it go out?
- **Frequency** - How often should it go out?
- **Channel** - What channel should it go out through?

Boilerplate notification systems are not completely devoid of context, but they do not have the level of sophistication that we intuit in person-to-person communication. Creating that level of sophistication is an exercise in data collection and usage. People communicate by ingesting a variety of data points like body language, tone, subject matter, environment, and more. We use these data points and feed it through our own rules system, evaluating against social norms, intimacy, personal preferences, and communication styles. We can mimic these human patterns by designing our notification systems similarly, generating a granular rules system that consumes an extensive data profile for each user.

Unfortunately there is no single playbook that tells you what data to gather or what rules to use. The contents, target, timing, frequency, and channel of notification will be highly dependent on who your customers are, the type of product/service your company provides, the stakes of each notification, and more. However, we can look at some examples of Slack and LinkedIn's well-designed notification engines and draw some broader UX design principles.

Slack

Slack is a workspace messaging application. Notifications are the main channel of contact to their 10+ million daily active users. Send too few and important work gets missed. Send too many and users get irritated. This exact frustration was the impetus that founded the notifications team at Slack. Their work resulted in the now-infamous notification flowchart.



Source

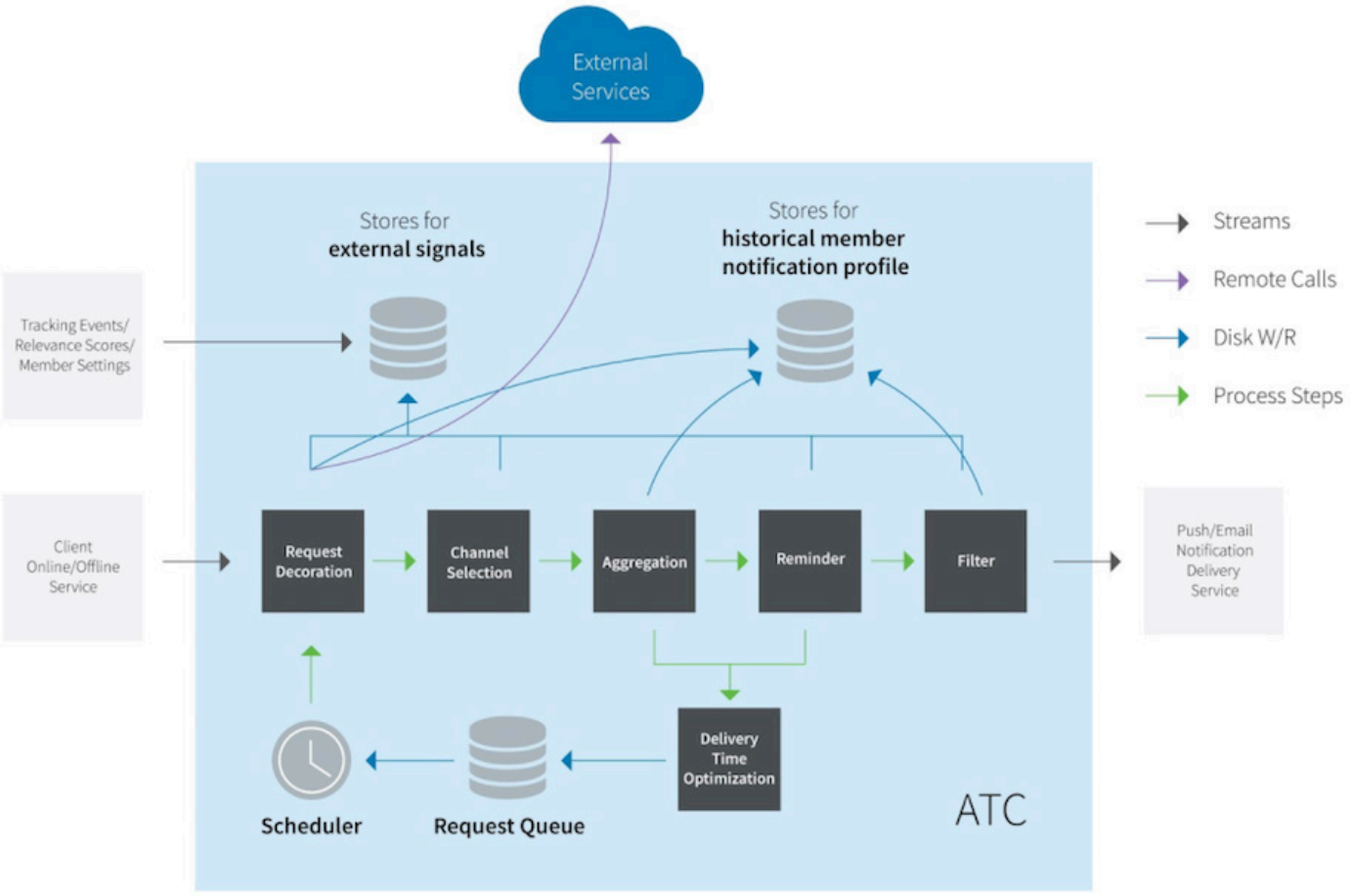
While daunting at first glance, Slack's notification flowchart is a carefully designed rules system that accounts for various contextual data points. Each fork follows a line of questioning that adds more context to the content, target, timing, frequency, and channel of each notification. All together, it's designed to gauge the user's attention and pull people back into the app at exactly the right time.

Note that Slack's rules are contextualized to the product. Slack does one thing very well, messaging. Control over these messages falls into the hands of the user. Slack makes very few assumptions, aside from its default configurations. Users have a plethora of possible options to customize their notification settings, including do-not-disturb mode, defined working hours, keyword notifications, muted threads, and lower the volume. Together, create a unique data profile that Slack then feeds through its rules engine.

LinkedIn

LinkedIn's notification system is slightly more complex with multiple application services within the greater LinkedIn product. Originally, each application team was allowed to create their own notification logic, but this resulted in a heavily fragmented and erratic system that spammed low-quality, irrelevant, and unregulated notifications to users. Eventually, LinkedIn unified all notifications under one platform, the Air Traffic Controller (ATC).

Unlike Slack, LinkedIn has a greater variety of notifications like connection requests, job opportunities, network updates, likes and comments, news, and more. Asking the user to modify their notification preferences for each category of notification would be overwhelming, so LinkedIn began tracking user behavior to build data profiles and inform notification delivery.



Source

The ATC uses a combination of historical user data, timing, and preferences to deliver personalized notifications. Each request begins by fetching the users historical data, their preferences, and devices. Channel selection optimizes which channel the notification will be best received - email, SMS, in-app, desktop, or push. Aggregation and Reminder batches like notifications and feeds them through some business logic to optimize for the best time to deliver a notification. The final filtering acts as a guardrail for the user's notification experience.

All together, the ATC works to filter through only the most relevant information to LinkedIn users. It's a sophisticated rules engine that parses many data points and is constantly updating, learning, and adapting to changing user behavior. But, unlike Slack, LinkedIn has too varied a set of notifications for users to configure on their own. Instead, LinkedIn pushes only the most relevant notifications by capturing current and historical user behavior and routing through a complex rules engine that constantly updates, learns, and adapts to changing user behavior.

Modern design patterns

Notification systems are as varied as they come. Instagram's internal mechanics will be vastly different from Jira, such as Slack and LinkedIn are. Their design will depend on company, industry, and user specific variables. However, we do see some patterns across multiple systems that can be distilled into best practices.

Let Users Control Their Notifications

As seen in both Slack and LinkedIn's rules engine, sending notifications is nothing more than educated guesswork on sending the right notification to the right person at the right time through the right channel. What better way to eliminate the guesswork by letting users themselves decide? There is a direct correlation between the granular control users have over notifications and the relevance of those notifications. Slack does this extremely well, allowing users to mute individual threads, assign keyword triggers, and set working hours.

The challenge with granular control is not what do users get to control, but when. Recall that the threshold for receiving disruptive notification is low. The sooner an application can learn about its users preferences, the better chance of retaining the user. However, users don't immediately jump to their settings to fine tune notification preferences. Some gentle prompting can be quite useful here. For example, the first time a Slack user receives a notification over the weekend, they are prompted to change their notification settings if they do not want to be disturbed over the weekend.

Gather Appropriate Context

After letting users dictate their own preferences, the next-best design pattern is to use context clues within a rules engine. Context is a fairly broad term, but some examples include:

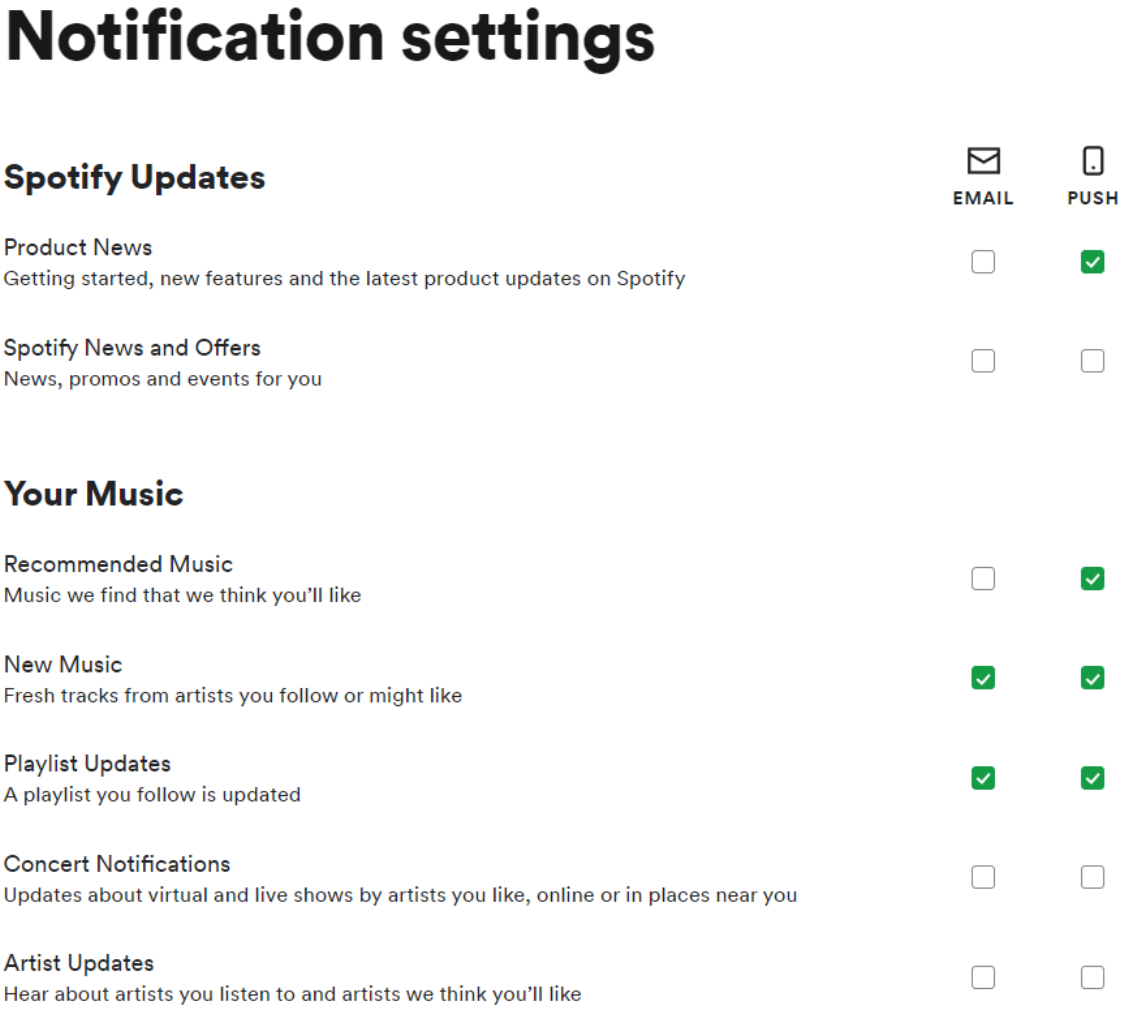
Context	Example
User Preference Explicitly set user preferences	Modifying Slack notification to send alerts after hours if a particular keyword is used
Sensory Data Time of day, location, temperature	Starbucks pushes discount notifications when users are near a starbucks, but only in the mornings or lunch time, when people are more likely to drink coffee
User Profile Historical user behavior and user archetypes	LinkedIn will begin sending job opportunities if users change their job status to "Seeking"
Priority Knowing when to override settings to push a high value notification through	United Airlines will override preferences to send notifications if a user's flight has been delayed

Contextual data points help your notification's rules engine become more intelligent and adaptable to evolving user behavior.

Multi-Channel Notifications

With multiple devices and channels of communication, there are more ways than ever to contact a user: email, in-app notifications, SMS, push notifications, phone call, etc. Knowing when and how to leverage each channel can improve the user experience significantly. For a truly effective system, notifications should be mapped to many channels, selecting the most appropriate channel to ensure maximum value.

Multi-channel does not mean spamming the same notification across different channels, but rather using channels as another tool in the notification toolkit to improve deliverability. These channel preferences can be informed directly by the user, by tracking engagement, or using your best judgment. For example, if users don't react to an email notification in a timely manner, applications can send notifications via a more disruptive channel, especially if a response is required to continue the workflow.



Spotify allows user to select preferred channels

How to build a notification system

If designed correctly, notifications will create a positive feedback loop, pulling users into the application, driving engagement, and triggering further downstream notifications that repeat the cycle. However, poorly designed notifications create a negative feedback loop, repeatedly bothering the user until they silence or delete the app all together.

Notifications are a high-stakes service. Get them right and you'll engage and retain users. Get them wrong and users that may otherwise enjoy the application will churn. Unfortunately the window to design these systems properly is small. As companies scale and product-market-fit aligns, resources begin to stretch thin. Notifications will ultimately become deprioritized in favor of features, integrations, and back-end scaling. It's best to get it right on the first try.

Building a notification system from scratch has two broad components:

1. **Design Considerations** - Knowing the basics of modern notification UX, what do we need to include in our notification system to support this experience?
2. **Technical Requirements** - What technical requirements must be met to support a system that can reliably scale and deliver notifications?

Design Considerations

Notifications are the resulting product of a notification system. In order to achieve the outcomes we see in Slack, LinkedIn, and other modern notification systems, we must first design the system itself.

Designing for Non-Technical Users

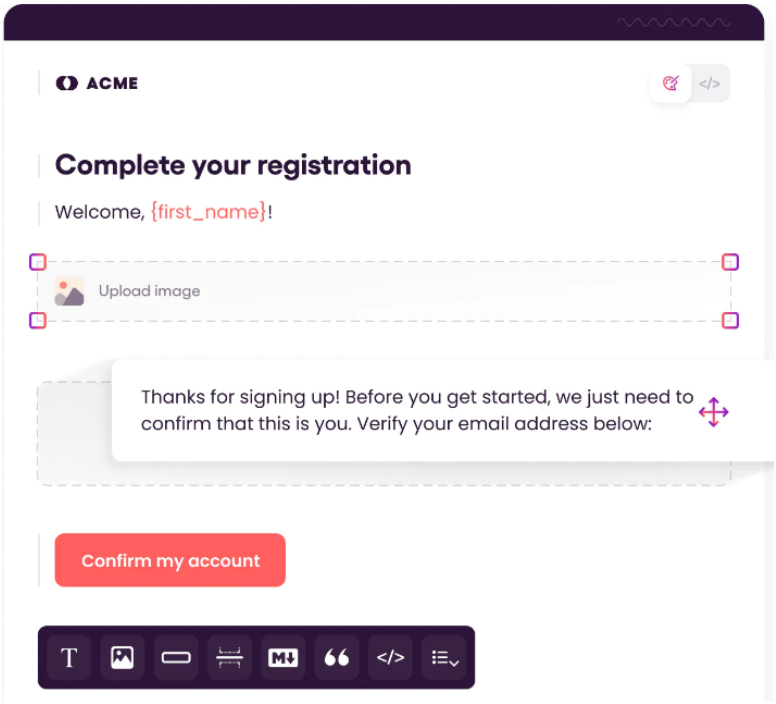
A notification system has two different classes of users:

- **End Users:** Users that consume the core product and receive notifications
- **Internal Users:** Core product teams that configure notifications for end users to receive

While end users see the final product that is a relevant notification, they are not the ones administering them. Internal users like sales, marketing, and customer success are the ones who initially configure each notification. For example, customer success teams may want to notify end users about a shipment delay. Sales teams may want to serve discount codes and product teams may want to batch metrics into one weekly alert.

> *Notifications are designed for the end user, but notification systems are designed for the internal user.*

Generally, internal users are not technical. They may not know how to modify a message's contents using markdown or how to redeploy code when a user's email has changed. Configuring notifications should not require engineering resources. Instead, the notification system should have a clean interface for non-technical users to navigate the system. Tools like drag and drop editors, out-of-the-box templates, and built-in branding can make the internal user experience much more intuitive and less demanding of engineering resources.



Drag and drop editors make it easy to create notification templates

Centralization

Disparate notification systems will lead to a disjointed UX and fragmented notification practices. This was the reason that LinkedIn built their ATC, to unify the notification experience across all teams. A centralized notification system will not only increase product velocity, but also create a consistent UX for all users across different areas of the product.

API First

As your product scales, notifications will continue to diversify into use cases that cannot currently be foreseen. Given that most of these notifications will be triggered by some event, the best way to minimize any potential future constraint on scale is to codify features through APIs. Each system feature that can be bundled into an API request will significantly reduce demand on engineering resources and create additional options for non-technical users to better sculpt notifications for their targeted end users.

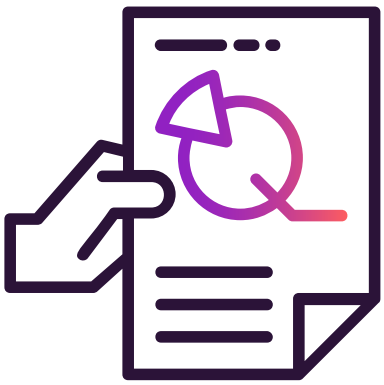
This means using an APIs to:

- Send asynchronous notifications to users
- Add and remove channels like SMS, email, etc
- Manage user profile and update preferences
- View the status of a notification
- Send notifications to a large number of users at once
- Create and modify templates
- Arbitrarily batch users and create lists
- Create custom triggers to automate a series of steps

The added benefit of layering APIs on top is that the underlying provider channels are abstracted away. Should your company move from Microsoft Teams to Slack or Sendgrid to Mailgun, developers can easily exchange one for the other with minimal changes to the codebase or notification service uptime.

Documentation

Internal documentation is always a helpful reference to facilitate the adoption of a new tool. Documenting common use cases, providing references, and written guides will go a long way in answering common questions and helping users better understand a system's full capabilities. Keep in mind that documentation should target both non-technical users as well as developers that may need to modify the notification system in the future, such as new notification channels.

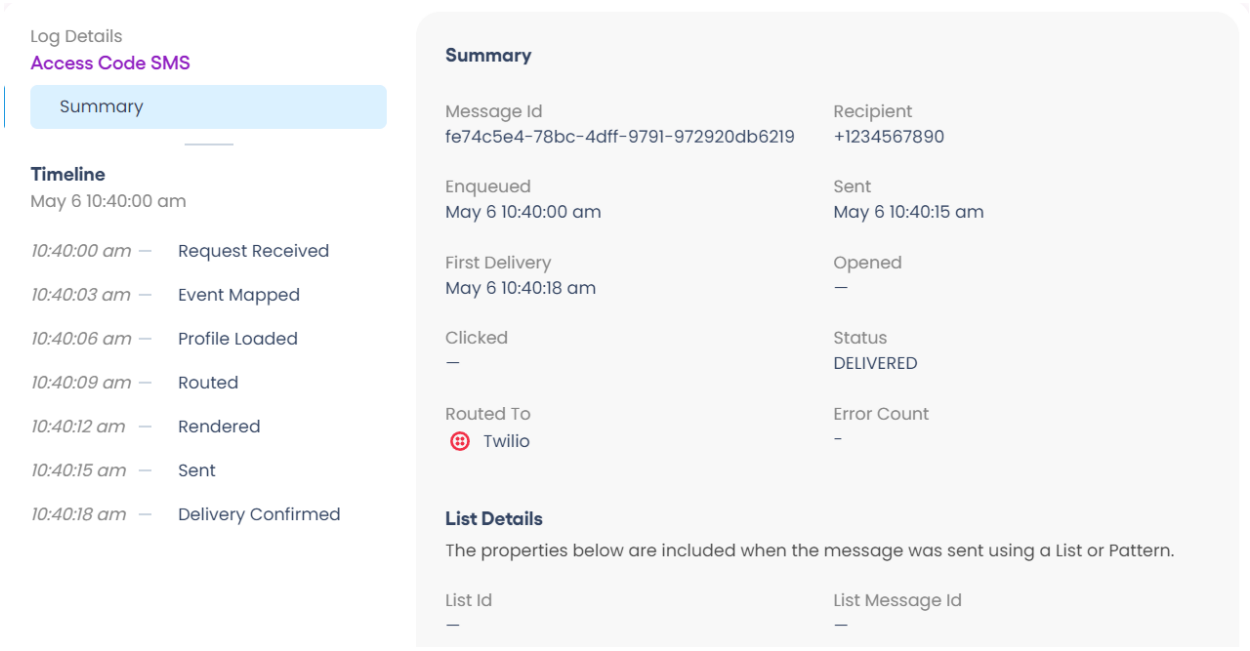


Analytics and Logging

Best practices for any application, product line, or service dictates that we should gather and parse data to make informed decisions. This is especially true for notifications. Recall that notifications are a key communication channel between the application and user, meaning we can divine quite a bit about user behavior by looking at the data. For example, conversion rates tell us which channel works best to capture new user interest.

Similarly, logs can help troubleshoot issues as they arise. Records like recipient, time, channel, and content can help internal teams quickly diagnose faults and errors. Customer success teams would be particularly interested in knowing if notifications have failed to deliver or if the user just missed them.

Analytics and logs should be presented in a format that's easy to understand so even non-technical users can grasp key insights at a glance.



Sample Log Dashboard

Technical Requirements

Thus far, we've envisioned a notification system that is thoughtfully designed with sophisticated data scoring algorithms traversing through a web of API requests across multiple channels towards targeted users at all hours of the day. This level of complexity is why LinkedIn and Slack both have dedicated engineering teams tasked with maintaining their notification service. It's also why we approach the technical requirements from the perspective of an enterprise-grade internet application.

Scalability

Once you find product-market fit your product's user base will grow quickly, and so too will notification volume. When the time comes, engineering resources will be diverted to solving other critical parts of your application rather than improving your notification system throughput. But you might cut into product growth if customers do not receive notifications. Eventually, a problematic notification system will impact the UX and create downstream business problems. Furthermore, notifications do not follow linear message volume; activity will unpredictably spike or remain uncharacteristically quiet.

Poor scalability results in problems like:

- Messages are not received due to delays, timeouts, or errors
- Messages are received beyond their relevance window
- Overwhelming message queues can result in a system-wide outage

Instead of having to constantly diverting engineering resources to deal with scalability errors, there are a number of tactics you can employ to support an ever-growing notification pipeline:

Pick a tech stack that scales - Each element of your notification system must support jobs at scale. It's not enough to support message throughput, but also:

- How can you distribute computation-heavy tasks like scoring?
- Where are you storing data? What can be stored in a separate DB and what must be stored on the host's disk? What's the quickest way to look up data?
- How will you deal with rate-limited from your provider's APIs?

Monitoring - Services like AWS Cloudwatch or DataDog can streamline the troubleshooting processed as long as the right evaluation metrics are chosen. Operational alerts like latency, throughput, and queue length can both alert SRE teams and provide useful data when performing retros.

Build in Redundancy - Always plan for your services to fail somewhere. While building in redundancies can be time consuming, the alternative of an outage could be even more damaging. Ask yourself how to best run copies of the same job so traffic can be diverted when needed.

Reliability

Reliable notifications are always delivered, without duplicates and on time. Assuming that a properly scaled application service can handle all internal concerns, the biggest threat to reliable communication are the third party providers that route notifications to the end user, like Twilio, Amazon SES, Firebase, Sendgrid, and dozens of others.

If and when these services inevitably go down, how will you respond?

- **Idempotency** - If your API request to send a notification is disrupted, how can you safely retry without accidentally performing the same operation twice? Nothing is more frustrating than receiving the same notification twice.
- **Fail Over** - What's the back-up plan if your SMS provider goes down? Can you switch from Twilio to MessageBird on the fly? Or perhaps falling back to email is your SMS provider is down.
- **Timeliness** - Some notifications are only useful if delivered within minutes, while others can go hours or days without a response. If messages are being dropped, you need to have a way to decide, per message, which makes sense to drop and which must be delivered.

Deliverability

In order to support consistently reliable notifications, each must be optimized and tracked for delivery, customized for each channel.

Channel	Optimization	Tracking
Email	SPF, DKIM, and DMARC need to be in place to avoid spam filters Cross-client email testing/preview	Can track open rate and click through Deliverability data is dependent on the email server you are sending to Constantly shifting landscape, what data is available is changing all the time
SMS	If you are sending at scale you need to use multiple numbers to ensure you're not blocked by the provider Regional pricing varies greatly, vendor support for various regions varies as well	Carrier may silently block your delivery so tracking is tricky, tracking click through on links is your best bet It's a good idea to build your own domain tracking, otherwise your deliverability can be affected due to the URL shortening services like Bitly
Mobile Push	Typically as your volume of mobile push notifications goes up deliverability will go down as users will revoke push permissions Best optimization tactic is to make each notification as useful and timely as possible	You can track engagement as a proxy but there is no way to track real deliverability

	You can optimize the process of asking for permission but not actual deliverability iOS is introducing message batching in Fall 2021 which will change the UX of iPhone push notifications (based on ML)	
In App	Basic web development user experience, ensure you test on different devices and browsers Many 3rd party notification services are blocked by ad blocking services	Can track the same way you'd track any other engagement on your website
Slack/Chat	Authentication and permissioning are complex, it's best to adhere to that particular chat platforms best practices to ensure deliverability	Varies by platform Slack provides data about if use was online when notification was delivered, click throughs w/hyperlinks or webhook enabled buttons (this requires server side tracking infrastructure)

Industry predictions

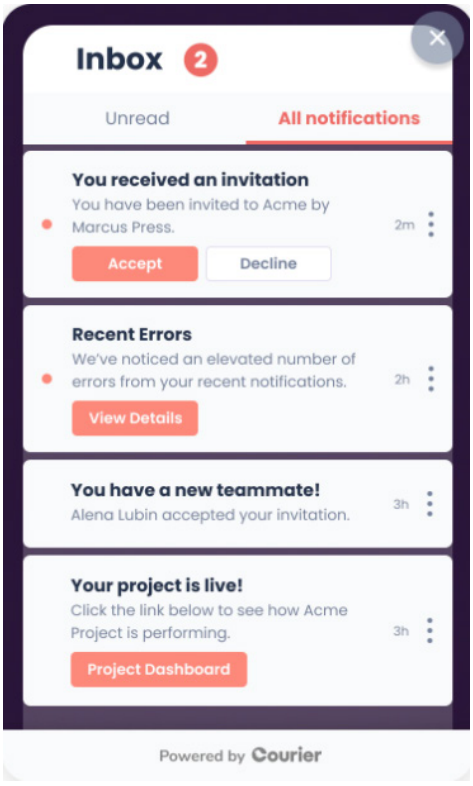
As we continue to create B2B and B2C products, notifications will become an increasingly important channel of communication to customers.

We've put together a few predictions that we anticipate shaping the future of notification technology.

Apple's famous marketing campaign still holds true over 10 years later

Universal Notification Inbox

With so many notifications fragmented across dozens of apps and devices, we'll naturally want to consolidate. We expect to see a universal inbox of all notifications to give users a central place to view notifications, like a GMail for notifications. This will allow end users to better customize all their notification experiences using a single source of truth, reducing the number of disruptive mobile notifications while increasing engagement with relevant notifications.



Source

Enriched Contextual Awareness

Many applications, like LinkedIn and Slack, already provide granular control over notifications and route them using intelligent scoring. We'll continue to see hyper-personalized notifications continue to trend. Consider:

- Apps like Foursquare and TicketMaster gcan suggest venues whenever you visit a new city
- News applications can read your biorhythms and send you heartwarming stories when your stress levels are high
- Yelp can read your profile and suggest exciting new restaurants to dine at in the evening
- Social media applications can learn the way you write and send notifications with a personalized tone, be that professional, quirky, humorous, or casual

We've only scratched the surface for the level of customization and personalization we can come to expect from notifications.

Notifications Will Drive Engagement

A major driver for increased personalization is that expectation that notifications will drive further user retention. The data has already validated this and companies are putting resources into leveraging this fact. In the future, personalized notifications will drive the majority of app engagement after companies have mastered the notification snowball effect. As suggested by **Noah Weiss**, SVP of Product at Slack,

> *"The best apps will be the ones you don't have to remember to use. They'll remind you. Soon that'll be the only type of app."*

Conclusion

Notifications pepper our lives with dings, pings, and plinks. They account for many product touchpoints and shape our perception of a brand or application. Yet notifications are often not considered part of the core product experience. They're an afterthought, tacked on at the end of a product delivery lifecycle. But companies that give notifications the proper care and attention are quickly gaining an important edge.

What these companies understand is that notifications are a form of communication. And humans have a high standard of communication after millennia of evolution. Most boilerplate notification systems are not sophisticated enough to emulate human communication. That's why we get so annoyed at irrelevant content, untimely alerts, and repeated messages. Human speech is complex, using countless context clues and norms to gauge the appropriate way to broach a conversation with someone else. It's natural that we expect to experience at least a similar quality of communication with our devices and applications.

Companies like Slack and LinkedIn have modeled their notification systems after organic human communication. They've developed sophisticated rules engines that ingest many different data points to feed only the most relevant of notifications at the right time, on the right channel, and to the right person.

These systems are not simple. Sophistication requires thoughtful design. Not only must the system itself be carefully designed to meet users current and future needs, but developers must also build systems that can reliably scale and deliver notifications no matter the volume.

Companies that can master these principles will gain the trust of their users. Their notifications will create a feedback loop of activity, artfully drawing users into the application and driving future notifications. With an oversaturation of notifications in our lives, a well designed system will help companies maintain a competitive advantage of their peers and foster positive brand sentiment.


Courier
www.courier.com