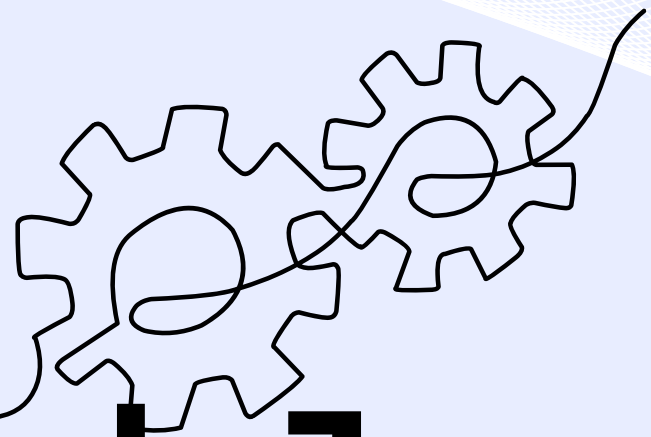

Tech Jargon *Part 2*

for

Product Managers

How Systems Talk

APIs | REST | GraphQL | Webhooks | SDK

From an Engineer-Turned-PM

The Integration & API Mental Model

A way of understanding your product as part of a network of systems that talk via APIs.

It helps PMs think in terms of contracts, endpoints, and data flows instead of only screens, so integrations become deliberate design choices, not hand-wavy "connect it somehow" requests.

When you understand this mental model, you can:

- Scope integrations precisely ("We need 3 endpoints")
- Spot reuse opportunities ("We already have an API for that")
- Design for flexibility (vs building one-off integrations)
- Have informed conversations about rate limits, webhooks, and data contracts

This guide covers the 10 essential terms that unlock this way of thinking.

API

Application Programming Interface

An API is a defined way for one system to talk to another—sending requests and getting structured responses back.

Think of it as a menu at a restaurant: it tells you what you can order (available functions) and how it will be served (response format), without revealing the kitchen's recipes.

We already have an internal API for that
let's reuse it instead of building new logic

For PMs, this means you can think in terms of reusable building blocks: before requesting new features, ask if an API already exists that can be wired into your product.

REST

Representational State Transfer

REST is a common style for building APIs using HTTP methods like GET, POST, PUT, DELETE and resource-based URLs such as /users/123. It's simple, widely adopted, underpins most JSON-based APIs you'll work with.

Think of it as a standardized filing system: every resource has a clear address, and you use simple verbs (GET, POST, PUT, DELETE) to interact with it.

We'll expose a REST endpoint—`GET /customers/{id}`—so the frontend can fetch the details

For PMs, this means you can talk about resources and operations ("get orders", "create invoice") instead of only UI screens, which makes scoping and API discussions much clearer.

GraphQL

Query language for flexible data fetching

GraphQL lets clients specify exactly what data they need in a single request, rather than hitting multiple REST endpoints or over-fetching large payloads. It's especially useful for complex UIs and multiple clients (web, mobile) needing different shapes of data.

Think of it as ordering à la carte instead of fixed combo meals: you get exactly what you want, nothing more, nothing less.

With GraphQL, the app can query just the fields it needs instead of pulling the whole object.

For PMs, this means you can solve some API pain points—like chatty UIs and over-fetching—by considering GraphQL where flexibility and performance matter across many clients. GraphQL adds flexibility, but it also introduces more backend complexity, so it’s not a default replacement for REST.

API Gateway

Centralized entry point for requests

An API gateway sits in front of multiple services and routes incoming requests to the right backend, often handling authentication, rate limiting, logging, and response aggregation.

Think of it as a hotel front desk: all guests check in at one place, which then directs them to the right room, verifies their identity, and tracks who comes and goes.

All external traffic will go through the API gateway, which will handle auth and routing.

For PMs, this means you have a central place to think about external access, security, and usage limits—important when launching public APIs or working with partners.

Endpoint

Specific URL or function within an API

An endpoint is a specific address plus method—like GET /users or POST /orders—that represents one operation in an API. Each endpoint has its own parameters, behavior, and response shape.

Think of it as different phone extensions in a company: each goes to a specific department that handles one type of request.

We just need a new endpoint for updating the user's billing address.

For PMs, this means you can reason about scope at the endpoint level: "Which endpoints do we need for this feature?" rather than vague "integrate with the billing system" conversations.

Rate limiting

Controlling request frequency

Rate limiting sets a maximum number of requests a client can make in a given period—like 100 requests per minute. It protects systems from overload and abuse, especially for public or partner APIs.

Think of it as a speed limit on a highway: it keeps traffic flowing smoothly for everyone and prevents one driver from causing a jam.

The API limit is 500 requests per minute per API key, so we're safe.

For PMs, this means you must consider how rate limits affect bulk operations, power users, and integrations—and negotiate limits that match your product's usage patterns.

Throttling

Slowing down traffic to prevent overload

Throttling actively slows or temporarily delays requests when a client or the system hits certain thresholds. It's often used together with rate limits as a safety valve during traffic spikes.

Think of it as a bouncer at a club: when it gets too crowded, they slow down entry to keep things manageable inside.

We'll throttle requests from that client until their usage drops back under the threshold.

For PMs, this means You should understand when heavy usage might trigger throttling, so you can design UX, messaging, and SLAs that handle "slow down" moments gracefully.

Webhook

Real-time event notifications pushed to another system

A webhook is a callback: when something happens in one system (like "invoice paid"), it sends an HTTP request with data to another system's URL. It's the reverse of polling—the receiving system doesn't have to keep asking.

Think of it as a doorbell: it alerts you when someone arrives, rather than you checking the door every minute.

We'll fire a webhook to your endpoint whenever a payment succeeds.

For PMs, this means webhooks are your friend for real-time integration—think "we'll notify you when it happens" instead of "you keep checking if it happened."

Payload

The actual data being sent in a request

The payload is the body of an API request or response—often JSON—that contains the fields and values (like {"userId": 123, "plan": "pro"}). Headers and URLs describe the request; the payload carries the content. Think of it as the contents of a package: the shipping label tells you where it's going, but the payload is what's actually inside the box.

The endpoint is fine, but we need to add planType to the payload.

For PMs, this means you can be specific about what data is included or missing: "We need these fields in the response payload to render the screen.

API key / token

Secret used to authenticate API clients

An API key or token is a credential a client includes with requests so the API can identify and authenticate who's calling. It's like a password for systems, often tied to rate limits and permissions.In practice, API keys are often long-lived, while tokens are usually short-lived and scoped to specific permissions.

Think of it as a backstage pass: it proves you're authorized to access certain areas and determines what you're allowed to do.

We'll generate a new API token for that partner and scope it to read-only access.

For PMs, this means you need to consider how clients obtain, rotate, and secure keys—and what happens if one is compromised—when designing integrations and partner programs.

Stay Tuned

Up Next

A plain-English guide to Keeping It Secure & Organized: Auth, Tenancy & Debt, and why it matters for PMs.