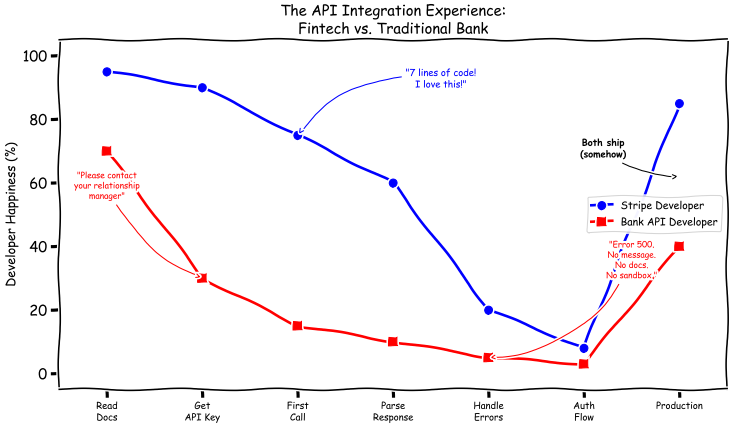
Lesson 6.3: API Economy and Platform Architecture

Module 6: The Infrastructure Problem

Prof. Dr. Joerg Osterrieder

February 9, 2026

APIs in Finance: A Candid Look



The API is the contract between systems. Get it wrong and nothing talks to anything—get it right and you unlock an entire ecosystem.

Learning Objectives

By the end of this lesson, you will be able to:

1. Define API, REST, GraphQL, and webhook and explain when each is appropriate in financial services
2. Trace the regulatory path from PSD2 to PSD3 and explain how Account Information Services (AIS) and Payment Initiation Services (PIS) work
3. Describe the architecture of an API gateway and explain why it is the control plane of modern banking
4. Compare Banking-as-a-Service (BaaS) and embedded finance models and identify revenue flows
5. Evaluate OAuth 2.0, mTLS, and rate limiting as layers of API security
6. Assess developer experience (DX) as a competitive advantage for platform businesses

---

APIs are the connective tissue of digital finance. Every payment app, neobank, and trading platform depends on APIs to function.

Bridge: From Internal Modernization to External Openness

Where we have been:

- Module 6 Lessons 1–2: Legacy systems, core banking modernization, monolith-to-microservices migration
- Banks invested billions replacing COBOL back-ends and adopting cloud infrastructure
- Internal plumbing is necessary but not sufficient

Where we are going:

- This lesson: How banks **expose** their capabilities to the outside world via APIs
- APIs turn a bank into a **platform**
- Regulation (PSD2/PSD3) forces openness; strategy determines who profits from it

Banks are modernizing internally. The API is how they open up externally.

Without APIs, a modernized core banking system is a high-performance engine with no wheels. APIs connect the engine to the road.

What Is an API?

Definition:

- **Application Programming Interface**: a contract that defines how two software systems communicate
- Specifies: endpoints, request format, response format, error codes, authentication
- Analogy: a restaurant menu—you do not visit the kitchen; you order from a defined list

Why APIs matter in finance:

- Enable third parties to build on bank data
- Power every fintech app (Revolut, Stripe, Plaid)
- Allow real-time integration between institutions

Types of financial APIs:

- **Account APIs**: balances, transactions, statements
- **Payment APIs**: initiate transfers, direct debits
- **Identity/KYC APIs**: verify customer identity
- **Lending APIs**: credit decisions, loan origination
- **Market Data APIs**: prices, order books, trade execution

Key metric:

- Plaid connects to >12,000 financial institutions via APIs (2024)

An API is not a technology choice—it is a business decision. Every API you expose defines who can build on your platform and how.

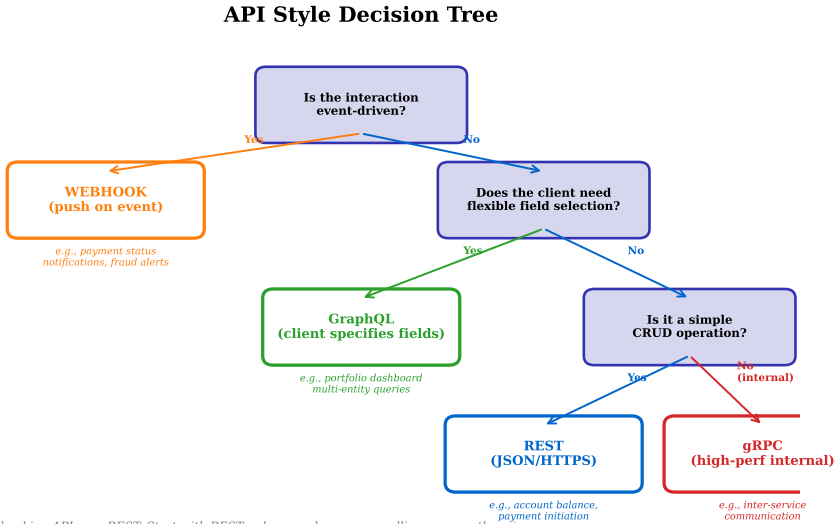
REST vs. GraphQL vs. Webhooks

Aspect	REST	GraphQL	Webhook
Paradigm	Request/response, resource-based	Query language, client specifies fields	Event-driven push
Data fetching	Fixed endpoints, may overfetch	Client requests exactly what it needs	Server pushes on event
Financial use	Account balances, payments	Portfolio dashboards, multi-entity queries	Payment status, fraud alerts
Complexity	Low (well-understood)	Medium (schema management)	Low (fire-and-forget)
Caching	Easy (HTTP caching)	Harder (POST-based)	N/A

Industry reality: >90% of banking APIs are REST (JSON over HTTPS). GraphQL is growing in aggregator and dashboard use cases. Webhooks are essential for real-time notifications.

REST dominates because simplicity wins at scale. GraphQL shines when clients need flexible queries across complex financial data models.

Choosing the Right API Style



>90% of banking APIs use REST. Start with REST unless you have a compelling reason otherwise.

PSD2: The Regulatory Foundation of Open Banking

What is PSD2?

- **Payment Services Directive 2** (EU, effective Jan 2018)
- Forces banks to open customer data and payment rails to licensed third parties—via APIs
- Created two new roles:
  - **AISP**: Account Information Service Provider
  - **PISP**: Payment Initiation Service Provider

Key requirements:

- Banks must provide free, standardized APIs
- Customer consent is mandatory (Strong Customer Authentication—SCA)
- Third parties must be licensed by national regulators

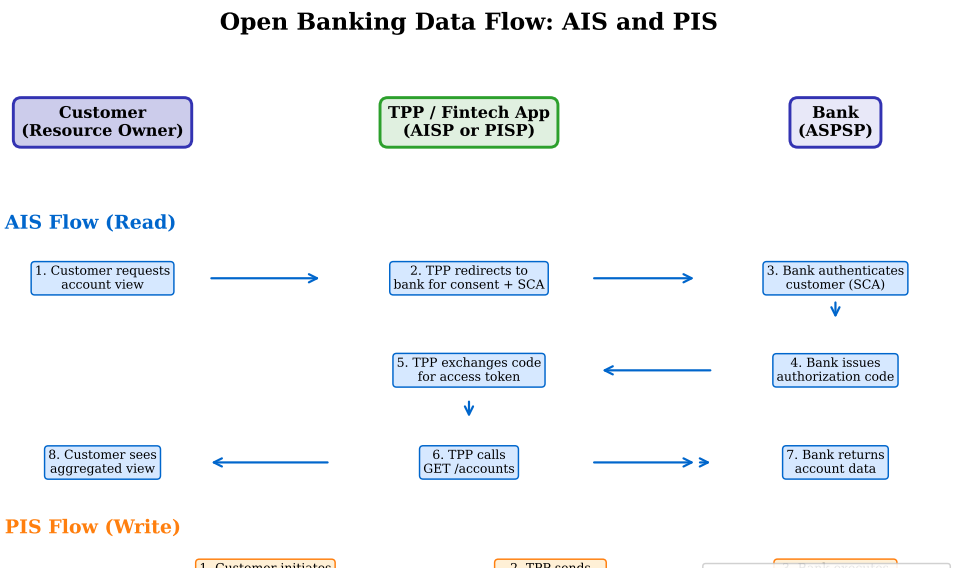
PSD2 in practice:

- **AIS** (read): aggregator apps show all your accounts in one place (e.g., Tink, Plaid)
- **PIS** (write): initiate a bank transfer without going through the bank's own app (e.g., Klarna, TrueLayer)

Impact by the numbers:

- 500+ licensed AISPs/PISPs in EU by 2024
- UK Open Banking: 7.5 million users, 1 billion API calls/month (2024)
- EU API call volume grew 50% year-over-year (2023–2024)

PSD2 was revolutionary: for the first time, regulation forced incumbents to open their most valuable asset—customer data—to competitors.


Open Banking Data Flow: AIS and PIS
Customer (Resource Owner)
TPP / Fintech App (AISP or PISP)
Bank (ASPSP)
AIS Flow (Read)
1. Customer requests account view
2. TPP redirects to bank for consent + SCA
3. Bank authenticates customer (SCA)
4. Bank issues authorization code
5. TPP exchanges code for access token
6. TPP calls GET /accounts
7. Bank returns account data
8. Customer sees aggregated view
PIS Flow (Write)

PSD3 and the Payment Services Regulation (PSR)

Why PSD3?

- PSD2 created APIs but quality varied wildly
- Banks used "compliance theater": APIs that technically existed but were slow, unreliable, or undocumented
- Proposed June 2023, expected adoption 2025–2026

Key changes in PSD3/PSR:

- **API performance standards**: latency, uptime, error rate SLAs
- **Dashboard access**: permanent API access tokens (no repeated SCA)
- **IBAN/name verification**: mandatory for all transfers

From PSD2 to open finance:

- PSD2 covered payments and accounts only
- **Financial Data Access (FiDA) regulation**: extends open data to insurance, pensions, investments, mortgages
- Creates a comprehensive "open finance" framework

Global convergence:

- UK: Open Banking → Open Finance roadmap
- Brazil: Open Finance (phase 4, 2024)
- India: Account Aggregator framework
- Australia: Consumer Data Right (CDR)

PSD3 fixes PSD2's gaps: it moves from "banks must offer APIs" to "banks must offer good APIs." Quality, not just existence, becomes the regulatory standard.

What Is an API Gateway?

Definition:

- A single entry point that sits between external consumers and internal microservices
- The **control plane** of your API ecosystem
- Handles cross-cutting concerns so individual services do not have to

Core functions:

- **Routing**: direct requests to the correct service
- **Authentication**: verify identity (OAuth tokens)
- **Rate limiting**: prevent abuse (100 req/sec per client)
- **Transformation**: convert formats (XML $\leftrightarrow$ JSON)
- **Logging & analytics**: track usage, latency, errors

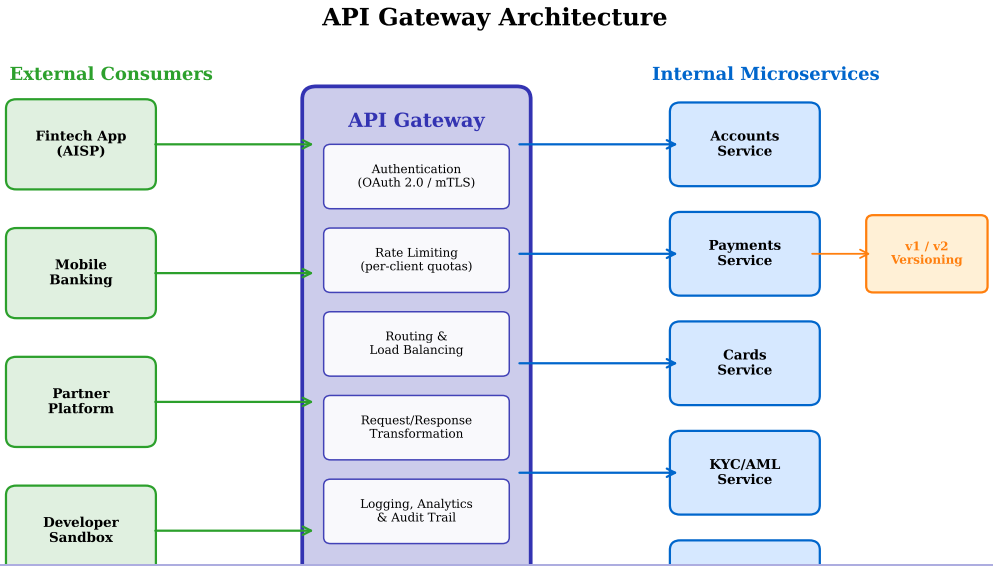
Why banks need an API gateway:

- Hundreds of internal services, each with different protocols
- External developers see one clean, versioned API surface
- Centralized security enforcement
- Regulatory audit trail in one place

Leading solutions:

- **Kong**: open-source, Lua/Nginx-based
- **Apigee** (Google): enterprise, strong analytics
- **AWS API Gateway**: serverless, pay-per-call
- **MuleSoft** (Salesforce): integration-heavy

The API gateway is to a bank's digital platform what the reception desk is to a building: every visitor passes through it, is identified, and is directed to the right floor.


API Gateway Architecture
External Consumers
Fintech App (AISP)
Mobile Banking
Partner Platform
Developer Sandbox
API Gateway
Authentication (OAuth 2.0 / mTLS)
Rate Limiting (per-client quotas)
Routing & Load Balancing
Request/Response Transformation
Logging, Analytics & Audit Trail
Internal Microservices
Accounts Service
Payments Service
v1 / v2 Versioning
Cards Service
KYC/AML Service

OAuth 2.0: The Authorization Standard

What is OAuth 2.0?

- An **authorization** framework (not authentication)
- Allows a user to grant a third party limited access to their bank data *without sharing credentials*
- The standard behind "Log in with Google" and every open banking consent flow

Four key roles:

1. **Resource Owner**: the customer
2. **Client**: the third-party app (e.g., budgeting app)
3. **Authorization Server**: issues access tokens
4. **Resource Server**: the bank's API

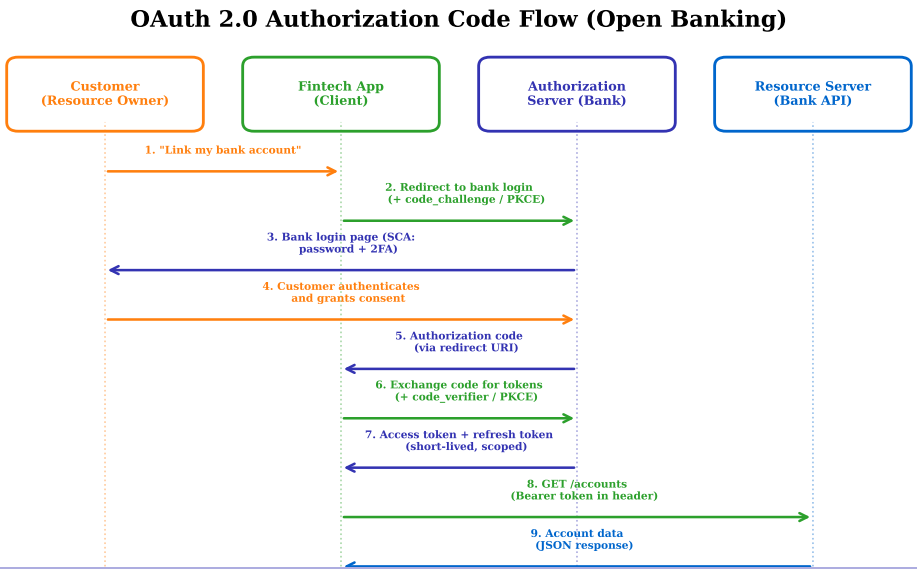
Financial-grade API (FAPI):

- OAuth 2.0 alone is not secure enough for banking
- **FAPI 2.0**: stricter profile with:
  - Proof of Possession (DPoP) tokens
  - Pushed Authorization Requests (PAR)
  - Client certificate-bound tokens
- Required by UK Open Banking, Berlin Group standards

Token types:

- **Access token**: short-lived (5–60 min), grants API access
- **Refresh token**: long-lived, used to obtain new access tokens
- **Scope**: limits what the token can do (read-only, payments, etc.)

OAuth 2.0 is the "valet key" of the digital world: it gives the parking attendant access to drive your car but not to open your glove box.



mTLS, Rate Limiting, and Defence in Depth

What is mTLS?

- **Mutual TLS**: both client and server present certificates
- Standard TLS: only the server proves its identity
- mTLS: the client (third-party app) also proves its identity with a certificate
- Required by PSD2 for AISP/PISP communication with banks

Why mTLS matters:

- Prevents unauthorized systems from calling bank APIs
- Certificates issued by qualified trust service providers (QTSPs)
- Creates a cryptographic chain of trust

Rate limiting:

- Caps the number of API calls per client per time window
- Prevents abuse, DDoS, and runaway scripts
- Typical: 100–1,000 requests/minute per API key
- Responses include `X-RateLimit-Remaining` header

Defence in depth—layered security:

1. **Network**: firewall, WAF, IP allowlisting
2. **Transport**: TLS 1.3, mTLS certificates
3. **Application**: OAuth 2.0 / FAPI tokens
4. **Data**: field-level encryption, tokenization
5. **Monitoring**: anomaly detection on API traffic

Security is not a single gate—it is a series of checkpoints. mTLS verifies identity at the transport layer; OAuth verifies authorization at the application layer.

What Is Banking-as-a-Service (BaaS)?

Definition:

- A model where a **licensed bank** exposes its core banking capabilities (accounts, payments, cards, lending) as APIs
- Non-bank companies embed these capabilities into their own products
- The bank provides the license, compliance, and infrastructure; the partner owns the customer relationship

BaaS vs. Open Banking:

- Open Banking: regulation-driven, read-mostly (AIS), standardized
- BaaS: market-driven, full banking stack, proprietary APIs

BaaS providers:

- **Solarisbank** (Germany): licensed bank offering BaaS APIs
- **Railsr** (formerly Railsbank): cards, accounts, compliance
- **Synapse** (US): banking infrastructure (wound down 2024—cautionary tale)
- **Treasury Prime** (US): bank-fintech connector
- **ClearBank** (UK): clearing and settlement APIs

Revenue model:

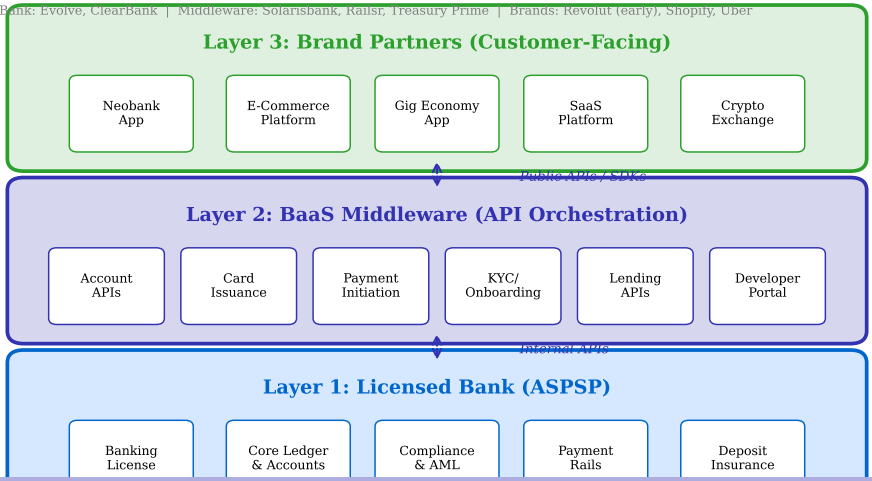
- Per-account fees ($1–5/account/month)
- Per-transaction fees (0.1–0.5%)
- Platform fees + revenue share

BaaS separates the banking license from the banking experience. Any company can offer financial products without becoming a bank.

Banking-as-a-Service (BaaS) Stack

Examples:

Bank: Evolve, ClearBank | Middleware: Solarisbank, Railsr, Treasury Prime | Brands: Revolut (early), Shopify, Uber



Case Study: The Synapse Collapse (2024)

What happened:

- Synapse: US BaaS middleware connecting fintechs to Evolve Bank
- Filed for bankruptcy in April 2024
- **Problem**: $85M in customer funds could not be reconciled between Synapse's ledger and Evolve's ledger
- Customers of fintech apps (Yotta, Juno, Copper) lost access to their money

Root causes:

- Synapse maintained its own "virtual ledger" that diverged from the bank's records
- Inadequate reconciliation between layers
- Regulatory grey zone: who is responsible—the bank, the middleware, or the fintech?

Lessons for BaaS:

- **Ledger integrity**: middleware must not create shadow accounting
- **Regulatory clarity**: the licensed bank is ultimately responsible
- **Operational resilience**: multi-provider redundancy
- FDIC issued guidance on BaaS risks (2024)

The Synapse collapse exposed the fragility of BaaS chains. When the middleware fails, the entire stack—and customer deposits—are at risk.

What Is Embedded Finance?

Definition:

- Integrating financial services (payments, lending, insurance) into **non-financial** products and customer journeys
- The user never leaves the host platform
- Finance becomes invisible infrastructure

Examples:

- **Shopify Capital**: loans offered at the point of sale inside the Shopify dashboard
- **Uber driver payments**: instant earnings payout via debit card
- **Apple Pay Later**: BNPL embedded in iPhone checkout
- **Tesla Insurance**: car insurance offered during vehicle purchase

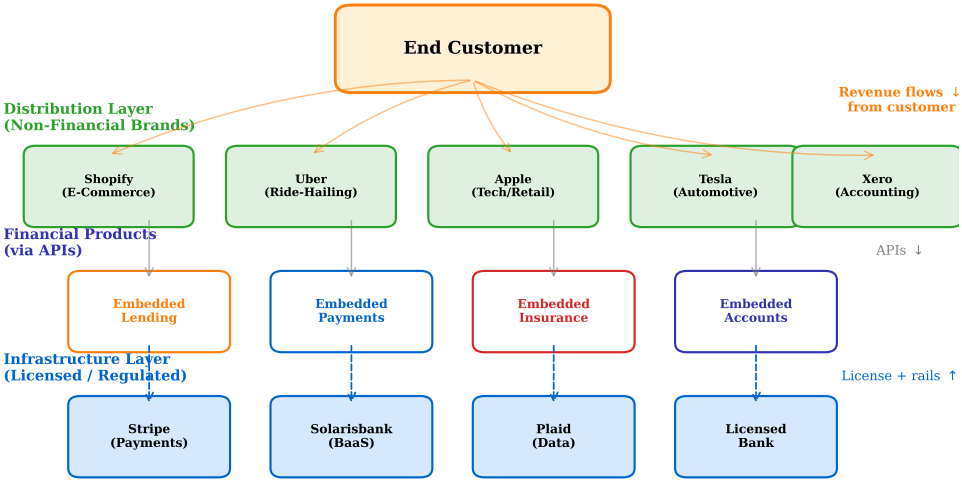
Market size:

- Embedded finance revenue projected at $230B globally by 2028 (Bain & Company)
- Embedded payments: 65% of total
- Embedded lending: 20% of total
- Embedded insurance: 10% of total

Why it works:

- Context: offer credit when the customer needs it
- Convenience: fewer steps, no app switching
- Data advantage: the platform knows usage patterns better than a bank
- Distribution: reach customers where they already are

Embedded finance is the end-game of BaaS: financial services disappear into the products people already use. The best financial product is one you do not notice.


Embedded Finance Ecosystem
End Customer
Revenue flows ↓
from customer
Distribution Layer
(Non-Financial Brands)
Shopify
(E-Commerce)
Uber
(Ride-Hailing)
Apple
(Tech/Retail)
Tesla
(Automotive)
Xero
(Accounting)
Financial Products
(via APIs)
APIs ↓
Embedded
Lending
Embedded
Payments
Embedded
Insurance
Embedded
Accounts
Infrastructure Layer
(Licensed / Regulated)
License + rails ↑
Stripe
(Payments)
Solarisbank
(BaaS)
Plaid
(Data)
Licensed
Bank

Platform Business Models in Finance

What is a platform business?

- Creates value by facilitating **interactions** between producers and consumers
- Does not own the product—orchestrates the ecosystem
- Network effects: value grows as more participants join

Financial platform archetypes:

- **Marketplace**: connects lenders and borrowers (LendingClub, Funding Circle)
- **Infrastructure**: provides building blocks (Stripe, Plaid)
- **Aggregator**: unifies multiple bank feeds (Tink, Yodlee)
- **Super-app**: bundles services (WeChat Pay, Grab, Revolut)

Platform economics metrics:

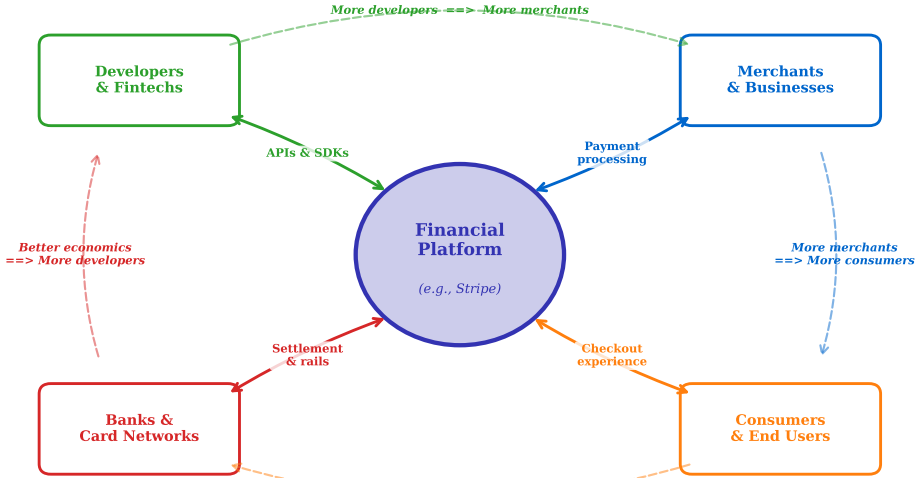
- **Take rate**: percentage of transaction value captured (Stripe: 2.9% + \$0.30)
- **API call volume**: proxy for ecosystem activity
- **Developer count**: measure of platform adoption
- **GMV**: Gross Merchandise Value flowing through the platform

Network effects in finance:

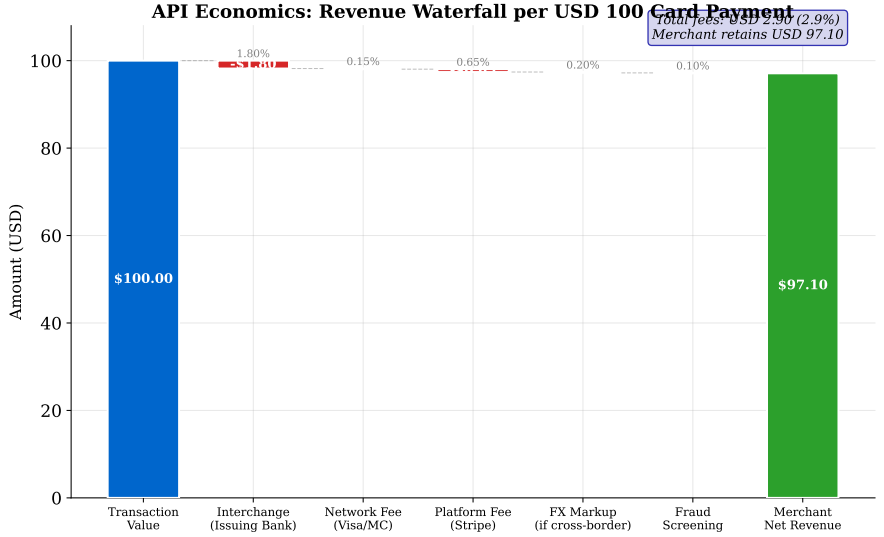
- More merchants $\rightarrow$ more consumers $\rightarrow$ more merchants (Visa/Mastercard)
- More developers building on Stripe $\rightarrow$ more payment volume $\rightarrow$ better fraud models

The most valuable financial companies are platforms, not banks. Visa, Mastercard, and Stripe succeed by enabling others rather than lending money themselves.

Financial Platform: Multi-Sided Network Effects



API Economics: How APIs Generate Revenue



Developer Experience (DX) as Competitive Advantage

What is DX?

- The **quality of the experience** developers have when using your APIs
- Encompasses: documentation, SDKs, sandbox environments, error messages, support
- DX is to API platforms what UX is to consumer apps

Why DX determines platform winners:

- Stripe won payments not on price but on DX: "7 lines of code to accept a payment"
- Plaid won aggregation by providing the best sandbox and clearest docs
- Bad DX = developers choose a competitor

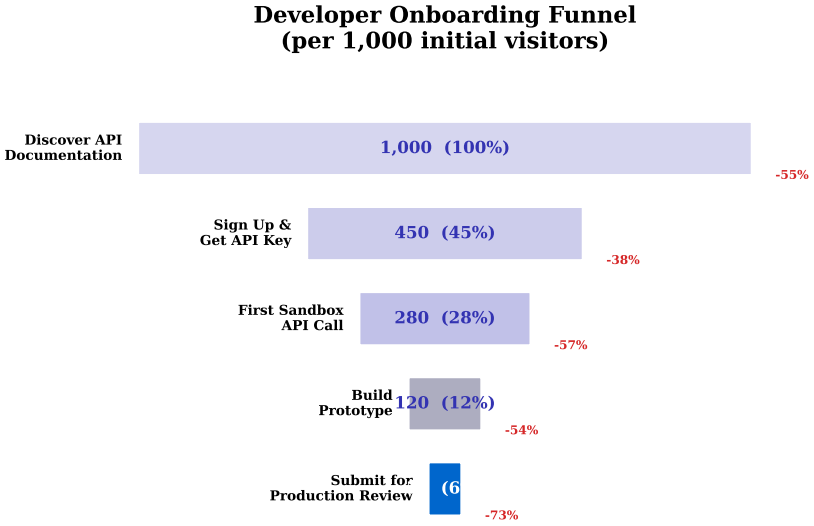
DX best practices:

- **Time to first API call** < 5 minutes
- Interactive API explorer (Swagger/OpenAPI)
- Sandbox with realistic test data
- SDKs in 5+ languages (Python, JS, Go, Java, Ruby)
- Idempotency keys for safe retries
- Versioned APIs with deprecation policy
- Clear, actionable error messages

Anti-patterns (common in banks):

- Requiring a sales call before API access
- PDF-only documentation
- No sandbox environment
- Breaking changes without notice

Stripe's success story is fundamentally a DX story. The company that makes it easiest for developers to integrate financial services wins the platform war.


Developer Onboarding Funnel
(per 1,000 initial visitors)
Discover API Documentation
1,000 (100%)
-55%
Sign Up & Get API Key
450 (45%)
-38%
First Sandbox API Call
280 (28%)
-57%
Build Prototype
120 (12%)
-54%
Submit for Production Review
(6
-73%

API Strategy: Build, Buy, or Partner?

Strategy	Advantages	Disadvantages	Example
Build (proprietary)	Full control, differentiation, data ownership	Slow, expensive, requires engineering talent	JP Morgan, Goldman Sachs
Buy (BaaS)	Fast to market, low capex, proven infrastructure	Vendor lock-in, margin compression, Synapse risk	Neobanks on Solarisbank
Partner (hybrid)	Balance of control and speed, shared risk	Integration complexity, governance overhead	Stripe + issuing banks

Decision factors: regulatory requirements, time-to-market pressure, engineering capability, strategic importance of the capability, available capital.

Most fintechs start with "buy" (BaaS), graduate to "partner" at scale, and only "build" capabilities that are core to their competitive advantage.

Risks and Challenges of the API Economy

Technical risks:

- **API sprawl**: hundreds of undocumented, inconsistent endpoints
- **Version management**: breaking changes cascade to all consumers
- **Latency**: API gateway adds overhead (10–50ms per hop)
- **Single point of failure**: gateway outage = total outage

Security risks:

- **Token theft**: stolen OAuth tokens grant unauthorized access
- **API abuse**: scraping, enumeration attacks
- **Shadow APIs**: undocumented endpoints that bypass security
- OWASP API Security Top 10 (2023)

Business risks:

- **Vendor lock-in**: dependency on a single BaaS provider (Synapse lesson)
- **Margin compression**: every intermediary takes a cut
- **Regulatory uncertainty**: PSD3/FiDA rules still evolving
- **Data sovereignty**: who owns the data flowing through APIs?

Operational risks:

- **Monitoring at scale**: millions of API calls/day
- **Incident response**: cascading failures across partners
- **SLA management**: guaranteeing 99.99% uptime across the chain

APIs amplify both opportunity and risk. Every connection is both a capability and an attack surface. Governance, monitoring, and resilience are non-negotiable.

Key Takeaways

1. **APIs are business strategy, not just technology**: every exposed API defines your platform boundaries and competitive positioning
2. **REST dominates financial APIs** for simplicity and cacheability; GraphQL for flexible queries; webhooks for event-driven flows
3. **PSD2 forced banks to open up**; PSD3 will force them to open up *well*—with performance SLAs and quality standards
4. **API gateways are the control plane**: routing, security, rate limiting, and analytics in one layer
5. **OAuth 2.0 + FAPI + mTLS** form the security stack; defence in depth is mandatory for financial APIs
6. **BaaS separates license from experience** but introduces chain-of-custody risk (Synapse collapse)
7. **Embedded finance** is the end-game: financial services disappear into non-financial products
8. **Developer experience (DX)** is the primary competitive differentiator for API-first platforms

---

The API economy transforms banking from a destination to an ingredient. The winners are those who make it easiest for others to build on their capabilities.

Summary: Lesson 6.3

Concepts covered:

- API fundamentals: REST, GraphQL, webhooks
- Open banking: PSD2, AIS, PIS, PSD3, FiDA
- API gateway architecture and functions
- Security: OAuth 2.0, FAPI, mTLS, rate limiting
- Banking-as-a-Service (BaaS) and the Synapse collapse
- Embedded finance ecosystem and market sizing
- Platform business models and network effects
- API economics and monetization
- Developer experience (DX) as competitive strategy

What comes next:

- Lesson 6.4: Cloud infrastructure, multi-tenancy, and operational resilience in financial services

Practical advice:

- Design APIs contract-first (OpenAPI spec before code)
- Invest in DX—it compounds like interest
- Always plan for the failure of your BaaS provider
- Security is layers, not a single lock
- Monitor API usage—it tells you what your ecosystem actually values

APIs are the nervous system of digital finance. Master them, and you understand how modern financial services are assembled, distributed, and monetized.