Pre-Class Discovery Handout: Open Banking Business Models

Activity 1: Business Model Canvas Detective

Scenario: Pick ONE open-banking aggregator you have read about in depth (for example Plaid, TrueLayer, Tink, Yapily, or Salt Edge). Fill in the canvas below by investigating how that aggregator actually works. Focus on the mechanics of monetising someone else's data, not on marketing language.

Canvas Element	Your Analysis
Customer Segments *Who pays the aggregator: developers, enterprise fintechs, or end-users?*	
Key Partners *Which banks are partners? Is the relationship regulatory or commercial?*	
Revenue Streams *What types of revenue (per-call, subscription, bundled) does the aggregator collect?*	
Channels *How does the aggregator reach and convert developer customers?*	
Cost Structure *What is the dominant cost line and what does it reflect structurally?*	

Q1: What single friction is this aggregator's core wedge? Is that friction still there if a bank ships its own direct API?

Q2: Does this aggregator earn its revenue from the developer side, the enterprise side, or a mix of both? Why does the answer matter for its customer-acquisition strategy?

Q3: If this aggregator disappeared tomorrow, what would the fintechs that depend on it lose — and could a single bank reproduce that loss with a direct API?

Activity 2: Unbundling Map

Scenario: The raw bank API is the commodity layer. Aggregators add adjacent products on top of it. Match each aggregator on the left to the primary wedge product it launched with — not its full product catalogue today.

© Joerg Osterrieder 2026

Aggregator		Launch Wedge Product
Plaid	____	Developer-friendly account-information connector for North American banks
TrueLayer	____	Data-then-payments aggregator built on European mandated APIs
Tink	____	Data normalisation and consent orchestration for Nordic and European banks
Yapily	____	Pure connectivity with no end-user interface for enterprise buyers
Salt Edge	____	Cross-region coverage spanning both mandated and contract-access markets

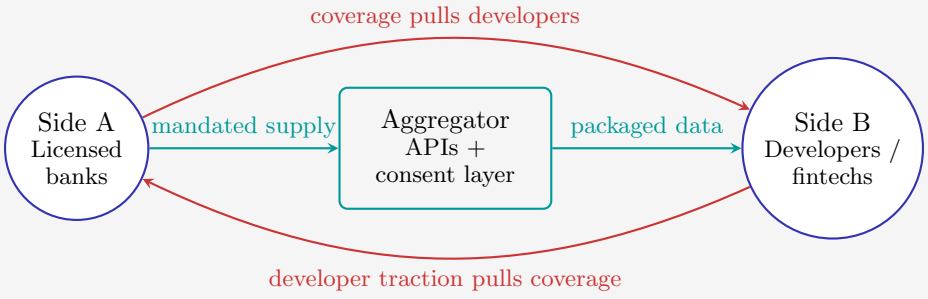
Q1: For each aggregator, describe in one sentence the friction its wedge product removes for a developer building on top of bank data.

Q2: Which of these aggregators has most aggressively added adjacent products after the wedge? What did it add, and in what order?

Q3: Why might an aggregator that launches with a raw connector eventually want to offer identity, fraud, and vertical-specific packages — and what does it risk by doing so?

Activity 3: The Platform Puzzle

Scenario: An open-banking aggregator is a two-sided platform where the supply side (licensed banks) is mandated to participate, and the demand side (developers and end-users) must still be cultivated. The aggregator sits between them, routing consent and data in both directions. Neither side finds the aggregator's data offering valuable without the other.



Q1: Why does an aggregator with wider bank coverage become more attractive to developers, and vice versa? Name the specific cross-side mechanism and explain why mandated supply does not eliminate it.

Q2: Which side should a new open-banking aggregator prioritise first, and why? How does this differ from the side a classic card network would target first?

Q3: Once an aggregator reaches critical mass in a region, what specifically makes it hard for a later entrant with the same connector stack to catch up?

© Joerg Osterrieder 2026

Solutions

Activity 1: Business Model Canvas Detective

A1: Model answer for Plaid: The core wedge is the developer integration pain of building and maintaining connectors to hundreds of North American banks individually. When a bank ships its own direct API, the friction shifts rather than disappears — the developer now has to maintain both the bank's direct API and Plaid's, because most applications need coverage across many banks. The wedge persists as long as one-by-one integration remains expensive at the portfolio level.

A2: Plaid earns predominantly from the developer side, not from end-users. Early revenue came from per-call or per-connection pricing charged to fintech applications; more recent revenue includes identity, income-verification, and onboarding packages sold to enterprise fintechs at tiered subscription pricing. The mix matters because developer-side pricing competes on breadth and latency; enterprise-side pricing competes on product depth and reliability.

A3: If Plaid disappeared, dependent fintechs would lose a single integration point for bank coverage across the North American market. A single bank could not reproduce that loss: it could provide its own direct API, but the portfolio-level friction of maintaining dozens of direct integrations would remain. That portfolio friction is what the aggregator removes and what no single bank can match by improving its own API.

Canvas elements (Plaid):

- **Customer Segments:** primary — fintech developers and engineering teams; secondary — enterprise fintechs buying identity, income, and onboarding packages.
- **Key Partners:** licensed banks whose connectors Plaid maps; cloud infrastructure providers; identity-data vendors used in premium overlays.
- **Revenue Streams:** per-connection or per-call pricing on the developer tier; tiered subscription with premium-data overlays on the enterprise tier.
- **Channels:** the developer portal, technical documentation, SDKs, and conference-based developer marketing; enterprise sales for the higher-tier products.
- **Cost Structure:** connector engineering and maintenance, uptime operations, compliance and privacy staffing, cloud infrastructure. Notably absent: branch infrastructure, end-user-facing apps, retail marketing.

Activity 2: Unbundling Map

A1: Plaid removes the friction of maintaining a connector portfolio across North American banks. TrueLayer removes the friction of stitching together mandated European APIs into a usable developer surface. Tink removes the friction of inconsistent data formats and consent flows across Nordic and European banks. Yapily removes the friction of being a bank-supplier-who-is-also-a-competitor — enterprise buyers prefer a pure-connectivity vendor that will not later sell a branded product downstream. Salt Edge removes the friction of operating across both mandated and contract-access markets with a single connector stack.

A2: TrueLayer has most aggressively rebundled: it launched with raw account-information connectors, added payment initiation and variable recurring payments, followed with refund orchestration and bank-to-bank checkout, and subsequently introduced vertical-specific packages for e-commerce, fraud, and identity. The ordering illustrates Christensen's unbundling-to-rebundling cycle: data-adjacent products seed the consent relationship; payment-adjacent products reuse the consent primitive for higher-risk actions; vertical packages arrive last because they require operational capacity the earlier waves funded.

A3: A full-stack aggregator captures more revenue per developer customer and becomes harder to displace as integrations deepen. It risks, however, that each product added crosses new regulatory perimeters (payments, identity, fraud prevention), and the regulatory overhead compresses

© Joerg Osterrieder 2026

margin. If the aggregator imports regulated-vendor cost structure faster than it builds regulated-vendor premium, it loses the narrow-classification arbitrage that financed the earlier engineering investment.

Activity 3: The Platform Puzzle

A1: The cross-side mechanism is a classic two-sided network effect, with the specific feature that the supply side is mandated rather than commercial. Wider bank coverage makes the aggregator more attractive to developers because developers need coverage to build applications that work for real customers. Developer traction in turn makes the aggregator more attractive to banks — once many developers route through an aggregator, banks find it cheaper to partner with that aggregator than to serve many developers directly. Mandated supply does not eliminate this mechanism because banks still choose how much engineering investment to put into their mandated APIs, and aggregators shape the revealed preference of the developer side.

A2: A new aggregator typically prioritises the **supply side first** in open banking, because supply is constrained by partnership complexity rather than regulatory absence. A card network faces the opposite problem: payment-acceptance supply (merchants) is commercially expensive to acquire, so the network usually seeds the merchant side before consumers. In open banking, the mandated supply is easier to assemble than the demand-side adoption, which is the inverse of the card-network problem.

A3: A mature aggregator accumulates a structural moat made of three layers: engineering (the connector stack handles schema drift, uptime variability, and latency variance that a later entrant must rebuild), relationships (partnership paperwork and reliability history with banks is multi-year capital), and operational trust (developers who have integrated do not re-integrate elsewhere unless the aggregator fails badly). A later entrant must match all three simultaneously to displace the incumbent aggregator, which is harder than matching any one of them.

© Joerg Osterrieder 2026