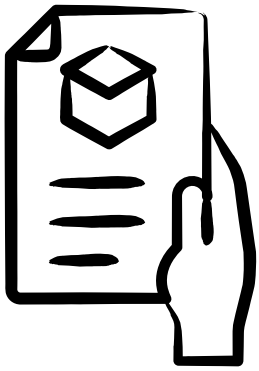
FYNDNA

Designing for Progressive Transformation in Retail Banking

An Architecture Rationale from FYNDNA

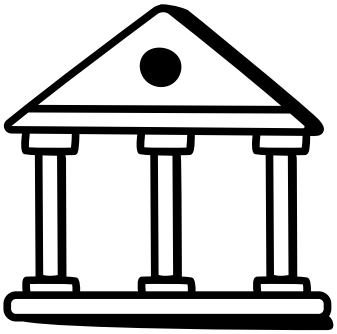
About this Document



This document shares the thinking behind FYNDNA's architecture.
It is not meant to present a perfect or universal architecture. Instead, it explains the practical decisions behind the architecture, shaped by the realities of building and running systems in live banking environments.
The focus is not on future promises or ideal end states. It is on how change can be introduced safely and gradually while systems remain live, regulated and operating at scale.
The examples and numbers referenced here come from capabilities that are already in production.

Table of Contents

01 Why Banking Systems Look the Way They Do



Most banking systems were built at a time when the world was much simpler.

Banks were organised around products. A savings system handled savings, a loan system handled loans, cards, deposits and other products lived in their own silos. Customers mostly interacted through branches, transaction volumes were predictable and change was infrequent. In that context, building product-centric systems made complete sense.

Each product system owned its own data, rules and configurations. Business logic lived inside the product. This approach prioritised correctness, reconciliation and reliability over frequent change. If something needed to be modified in a savings account, it was done within the savings system itself. The result was stability and control.

When digital channels such as internet banking, mobile apps and APIs emerged, they were largely added on top of these existing systems. The core structures remained unchanged. This allowed banks to introduce new channels without disrupting systems that were already running reliably.

The problem emerged later, as the nature of banking began to change.

Customers started using multiple products together. Regulations began cutting across products instead of applying to just one. Payments moved towards real time. Systems designed to operate during business hours were suddenly expected to run 24×7. As a result, the clean boundaries between product systems began to blur. Expectations have evolved not just for customers, but also for bank users – who increasingly expect faster responses, better visibility and systems that support decision-making rather than slow it down.

As system boundaries blurred, changes began cutting across products, teams and platforms, pushing stable systems to change far more often than they were designed for.
At the same time, the environment itself has become riskier — with more fraud, mule accounts and scams making fragmented data and delayed controls far more dangerous than before.

The challenges banks face today are not because legacy systems were poorly designed. They were designed for a different world.

Today's issues come from asking those systems to support a very different kind of change – more frequent, more cross-cutting and at a much larger scale. At the same time, banks are trying to shift towards a more customer-centric way of working, while their underlying systems remain organised around products.

Any modernisation approach has to start from this reality. Without understanding why existing systems were built the way they were, it becomes easy to underestimate both their strengths and the risks involved in changing them.

02 Progressive Transformation as a Design Lens

Once we understand why banking systems were built the way they were, a clear reality emerges. Banks today are expected to do two things at the same time. They have to remain stable – because trust, money and regulation leave no room for disruption.
At the same time, they have to keep changing – because customer expectations and competition continue to move forward.

In most transformation efforts, banks do define a future direction and an intended end state.
This is necessary to align teams, investments and priorities.

What we observed in live environments, however, is that moving towards that direction rarely happens through a single, clean transition. Large systems cannot be changed all at once. Old and new platforms need to run together for long periods, while real customer traffic continues every day.
The real challenge, therefore, is not deciding where a bank wants to go, but ensuring that change can happen safely and incrementally while systems remain live, regulated and under load.
This shaped how we approached the architecture.

Rather than treating transformation as a one-time move, we focused on how systems could support continuous change while operating at scale.

This means building a platform that:

- Can change over time without locking the bank into hard infrastructure decisions upfront
- Can be applied across very different banking environments and
- Can remain secure, observable and compliant throughout the process

The platform is designed to be cloud-native, while remaining cloud-agnostic and deployable across public cloud, hybrid cloud and on premise environments without being tightly coupled to any single provider.

The intent is to benefit from cloud scale and resilience, while keeping infrastructure choices flexible over time. CNCF compliant open-source components and standard infrastructure patterns help ensure portability, consistency across environments, and the ability to evolve the underlying technology without widespread disruption.

At the same time, the architecture does not assume a single type of bank or market. A retail bank in the Middle East, a large bank in Australia or a bank operating in the United States face very different regulatory frameworks, product structures and operating models. The approach is to keep core architectural principles consistent, while handling differences through configuration, policy and data rather than through separate systems or one-off builds.

Security, compliance and auditability are treated as design inputs rather than concerns addressed later. Industry standards such as BIAN and ISO 20022 provide useful reference frameworks for structuring capabilities, data and events. This helps ensure that system behaviour remains predictable and traceable, even as changes are introduced while systems are live.
As a result, compliance becomes easier to maintain over time.

Successful change at scale also depends on systems remaining easy to understand as they evolve. If teams cannot reason about how systems behave in production, introducing change becomes risky. For this reason, we deliberately favoured approaches that are easier to operate and manage in practice.

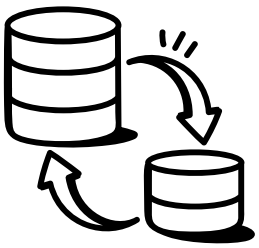
As platforms grow in scale and distribution, infrastructure complexity increases. To manage this, AI is applied selectively to simplify environment setup and ongoing infrastructure maintenance
The focus is on making it easier and faster to set up and maintain environments, without affecting how live banking systems behave.

Taken together, these choices shape what we refer to as Progressive Transformation.

In this model, coexistence is assumed, not avoided. New capabilities are introduced gradually under live traffic and expanded only as confidence builds.

Progress is measured not by how much has been replaced, but by how reliably change can be introduced without disrupting customers, operations or compliance.

Everything described in the sections that follow is built on the below assumption:



Systems will be live, regulated and under load while change is happening.

03 Separating Stability from Change – Core Architectural Choices

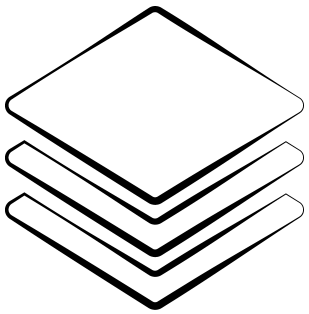
Early on, we realised something fundamental. Not everything in a bank changes at the same speed. Some things change frequently – customer data, products, pricing, onboarding flows, payment mechanisms, rules, limits and their related policies.

Other things should change as little as possible – posting transactions, maintaining balances, accounting and settlement. Problems start when these two get mixed together.

Over time, systems that were meant to remain stable started carrying more and more business behaviour. As a result, even small changes began to feel risky. A pricing tweak meant touching the core. A KYC policy update meant coordinating changes across multiple systems. What should have been routine, became slow, fragile and hard to control. So, we made a clear decision.

Keep execution close to the core product processor. Move business behaviour out.

Core product processors continue to do what they are best at – posting transactions, maintaining balances, handling accounting and ensuring correctness. Stability matters more than flexibility here. Everything that changes more often is handled outside the core, across two layers.



At the **enterprise layer**, we place business capabilities such as customer management, product definition, pricing, originations, payments, consent, limits, and collateral. These capabilities are defined once and reused across products and channels, instead of being rebuilt inside each core system. The capabilities at this layer help in making the architecture customer centric.

Below that sits a **foundation layer**. This layer provides shared services such as workflow, rules, document handling, security, analytics and coexistence controls. It provides common capabilities used by both enterprise components and core systems, avoiding duplication and fragmentation.

These layers reduce what the core needs to handle over time. The core becomes lighter and more stable, while change is absorbed outside.

This approach does not replace existing cores. It protects them.

To make this separation work in practice, new enterprise components are designed to interact with each other through events, rather than tightly coupled integrations.

When something changes, say a customer update, a pricing decision, a consent change or a payment event, it is published once and other enterprise capabilities react independently.

This event-first approach keeps enterprise components loosely coupled. It avoids chain reactions when one system slows down, simplifies integration and allows new capabilities to be introduced without disturbing what is already running.

However, we recognise that not all existing core systems operate this way. Many cores were built earlier, with business logic embedded inside them and may rely on synchronous APIs, batch processes or internal configurations rather than consuming events.

Because of this, enterprise components work with existing cores using coexistence approaches, instead of forcing everything to work the same way:

Where the core can publish or consume events, we integrate using events.

Where it cannot, enterprise components interact through controlled APIs or adapters.

For example, pricing is defined once in the ePricing capability. A new lending system consumes pricing decisions directly from ePricing. An existing lending system may continue to use its embedded pricing logic, while ePricing operates alongside it for newer products or journeys.

This approach does not attempt to re-engineer existing cores. New capabilities can be introduced without disturbing running systems. Coexistence becomes manageable rather than painful.

The same separation also applies to the underlying technology stack.
Components such as databases or messaging layers are chosen based on clear requirements and can be replaced over time if better alternatives emerge, without forcing changes across the rest of the platform.

04 Engineering for Scale, Availability and Continuity

With stability and change separated at an architectural level, the next challenge was making this model work in real banking environments.

If enterprise capabilities are going to operate outside the core, they need to do so at very large scale. They need to remain available at all times. And, they need to continue functioning even when parts of the system fail. For large banks, this is not just a technology concern – The regulators expect these systems to keep running because the impact of failure is much wider.

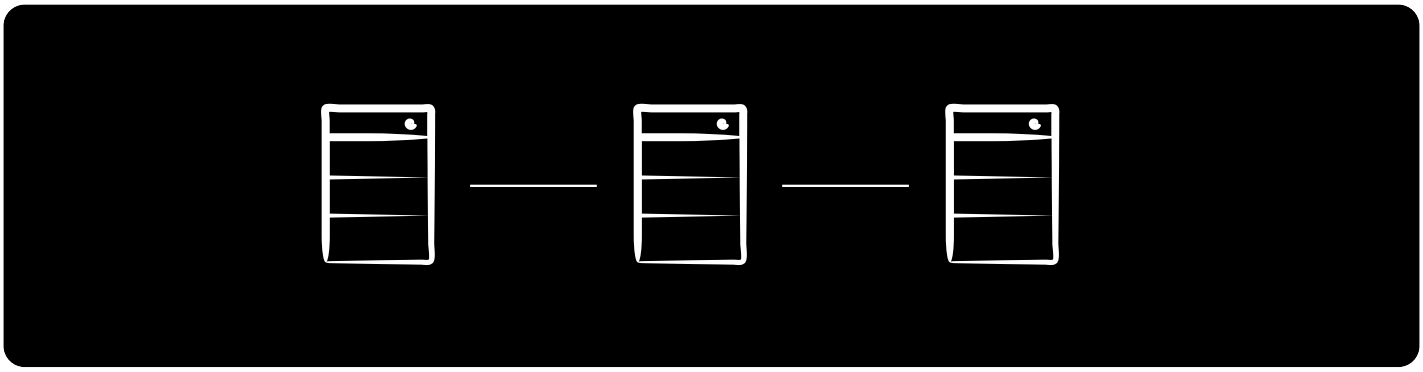
These requirements were not theoretical. They came directly from live environments- population scale customer data, real-time payments, regulatory expectations and systems that cannot be taken down for maintenance or experimentation.

We started with a few simple assumptions

- **Scale will continue to increase**
- **Failures will happen**
- **Change will need to be introduced while systems are live**

Everything that follows in this section is a direct response to those assumptions.

4.1 Designing for Scale First



Scale was a first-order concern from the beginning.

We were not designing for pilots or limited volumes. Customer data runs into tens of millions. Payment traffic can spike suddenly and unpredictably. At the same time, some enterprise capabilities need to handle consistently high transaction volumes, while others work with large and variable data sets such as customer profiles, documents and relationships.

This made horizontal scalability non-negotiable.

Capacity needed to be added incrementally, without redesigning systems or interrupting live operations.

Different kinds of work also needed to run side by side. Transactional, time-critical processing, read-heavy access and reporting or analytical workloads all had to coexist without interfering with each other. Growth or load in one area should not slow down or destabilise the others.

To support this, enterprise data is structured in a way that separates what is stable from what needs to evolve. Core banking attributes remain strongly defined and consistent (stored using relational structures), while bank-specific or country-specific extensions are handled more flexibly (using JSON style attributes). This allows data models to grow over time without repeated redesigns or disruption to live systems.

This pattern is used across enterprise components such as customer management, originations, enabling data structures to expand safely while systems remain live.

Designing for scale upfront was essential. Without it, none of the continuity or live-change mechanisms described later would hold under real load.

In production, this is backed by a horizontally scalable and strongly consistent data layer for enterprise data such as customer, product and pricing. Alongside this sits a distributed messaging layer that allows work to be split, traffic to be controlled and different types of processing to remain isolated as volumes increase.

This is supported by running enterprise services as independent workloads that can scale out and recover on their own as demand changes. This allows capacity to be added gradually and failures to be handled without manual intervention or disruption to live systems.

4.2 Controlling How Change Enters Production

One of the earliest risks we had to address was migration risk.

When a new capability is introduced such as a new payment flow or a new version of an existing component, the main concern is not whether it works in isolation. It is how it behaves under real, live traffic.

Moving all traffic at once puts everything at risk.

We introduced an Insulation Layer between changeable enterprise capabilities and stable downstream systems.

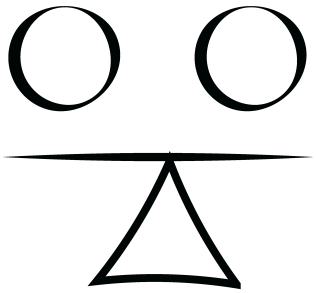
Instead of treating production cutover as a single event, this layer allows new capabilities to be introduced gradually. New behaviour begins by handling a very small portion of live traffic, while existing systems continue to process the rest. Behaviour is observed under real conditions, and traffic is increased deliberately only as confidence builds.

The insulation layer ensures that early issues remain contained. If something behaves unexpectedly, the impact is limited and reversible, without forcing immediate changes in stable systems.

This changes the role of production. Instead of being a final exam, production becomes a controlled learning environment. Adoption is guided by what we see in real usage, not what we assume upfront.

This approach is especially important during migrations and coexistence scenarios. Old and new systems can run side by side, traffic can be shifted safely, and large, irreversible switches are avoided.

4.3 Controlling How Much Stress Reaches Stable Systems



Once change is entering production in a controlled way, the next challenge is load. Not all traffic is the same. Some operations are critical and time-sensitive, while others are less urgent. Allowing all requests to flow directly into core systems assumes that they can absorb every spike or burst. In practice, this is rarely true and stable systems can quickly become overwhelmed.

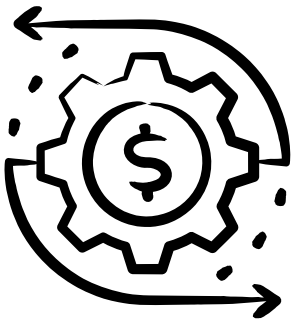
To address this, we introduced a dedicated **Control Layer**, internally referred to as the **Governor** that limits how much stress reaches stable systems.

Instead of treating the core as the first point of contact for every request, the Governor continuously monitors the health of downstream systems.

When the core is operating normally, traffic is allowed to flow. When the core shows signs of stress or unavailability, the Governor restricts incoming traffic to prevent overload.

This ensures that stable systems are protected during periods of high load or partial failure.
If downstream systems slow down, the impact is contained upstream rather than being pushed into the core. These controls are always in place, not just during incidents or peak loads. They prevent problems from spreading and help stable systems remain stable, even when demand is uneven or unpredictable.

4.4 Ensuring Continuity When the Core Is Unavailable



Even with controls in place, there are times when the core is unavailable – sometimes planned, sometimes not.

In many banking environments, this brings everything to a halt. Customer facing services stop working, operations teams are forced into reactive mode and manual workarounds take over. What should be a contained issue quickly turns into a visible disruption for customers and staff.

We chose to design explicitly for this scenario, through a dedicated capability we refer to as **eStandIn.**

Rather than treating core unavailability as an exception, eStandIn allows the platform to continue operating safely when it occurs.

Certain transactions especially customer facing payments can still be accepted and handled without immediately reaching the core.

These transactions are managed in a controlled manner and are completed or reconciled once the core becomes available again. From the customer's perspective, essential services continue to work, even if parts of the backend are temporarily unavailable.

This capability is not about scale or migration. It is about continuity.

4.5 Active–Active as an Operating Posture

Once change is controlled, load is regulated and continuity mechanisms are in place, a different way of operating becomes practical.

Instead of running one primary system with a standby waiting in the background, systems operate across regions at the same time. Multiple regions actively serve traffic, capacity is shared and failures are handled through routing decisions rather than shutdowns and restarts.

Active-Active is not treated as a disaster recovery setup that is tested occasionally. It is how the platform runs every day.

Applications are designed to operate independently in each region, using local infrastructure and services. Regions are intentionally deployed with significant geographic separation, often more than 1000 kilometres apart, so that a failure in one region does not turn into a wider outage. Traffic can be served from multiple regions at the same time, with capacity shared across them.

In order to ensure that disruption in one cloud provider does not bring the two regions down, the two regions are deployed on different cloud providers.

Because systems are already live in multiple regions, failover and failback are no longer high stress events. They become controlled transitions, with capacity already available and system behaviour well understood.

Active–Active operation also improves resource usage. Instead of keeping capacity idle in standby, resources are used continuously, supporting higher throughput and smoother scaling as demand grows.

This operating posture is sustained through continuous health monitoring and routing control. Routing and failover decisions are driven by system health and operational policies, allowing traffic to be shifted safely between regions when needed, rather than relying on emergency failover procedures.

Active-Active, in this sense, is not a feature that is switched on. It is an operating posture that becomes practical only after scale, change and continuity are addressed deliberately.

4.6 Operational Visibility as a Core Capability

Operating at scale, across regions and under continuous change requires more than resilient systems. It requires clear, real-time visibility into how those systems are actually behaving.
In many environments, observability is added only after problems surface. Metrics are partial, logs are fragmented and teams rely on manual investigation during incidents. This slows response and increases operational risk.

We chose to treat operational visibility as a core capability from the start. Health, performance and system behaviour are continuously observed across enterprise services and the supporting infrastructure. Signals from different parts of the platform are brought together, so issues can be detected early and understood in context, rather than being viewed in isolation. This visibility supports both day to day operations and failure scenarios.

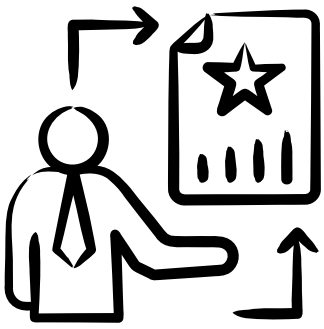
Teams can see how systems behave under load, how traffic is flowing across regions, and where pressure is building before it turns into customer facing disruption.

During incidents, this reduces guesswork. Faults can be isolated more quickly, corrective actions can be taken with confidence and recovery requires less manual intervention. Operational visibility also plays a preventive role. By understanding normal behaviour patterns, the platform can surface anomalies early and support proactive intervention, instead of reacting only after customers are impacted.

Together with scale, change control, continuity and active – active operation, this visibility ensures that the platform remains operable, understandable and predictable under live conditions.

05

From Principles to Practice



The architectural choices described so far were shaped while building systems that are already live and operating at scale. This section shows where those choices show up in practice, across key banking domains.

5.1 Customer : Managing Change Centrally

Customer data changes frequently and cuts across products, channels and regulatory requirements. When customer information is owned and updated separately inside individual product systems, even simple changes begin to create duplication, inconsistency and operational risk. We introduced a dedicated customer capability at the enterprise layer to act as the system of record for customer data.

Instead of attempting a large, one time cutover, we allowed responsibility for customer data to move gradually to the enterprise layer, while existing product systems continued to run as they always had. Customer data from multiple product systems was brought together at the enterprise layer, creating a unified view without disturbing product processors or live transactions.

Today, customer updates still happen within individual product systems and are then shared with the enterprise layer. Over time, this responsibility is being shifted gradually, with product-level updates being switched off one by one as confidence builds. The aim is to move all customer updates to the enterprise layer without disrupting systems that are already running.

This phased approach has allowed customer data for more than 93 million customers to be consolidated from multiple product systems.

It shows that enterprise ownership can be introduced step by step without touching stable execution systems and without destabilising systems already in production

5.2 Payments: Changing Systems While They Are Live

Payments operate under very different constraints from most other banking functions. They run at high volume, increasingly in real time and have very low tolerance for disruption. Even small changes can have immediate and visible impact.

Because of this, the transaction posting parts of payments were kept deliberately stable, while change and variability were handled at the enterprise level. Our enterprise payments capability sits above transaction posting systems, handling payment orchestration and absorbing change without disturbing the transaction posting systems.

Instead of replacing payment systems in one go, new payment capabilities were introduced alongside what was already running. Existing and new payment paths operated in parallel, and live traffic was moved gradually in small, controlled steps. This allowed teams to observe behaviour under real conditions and roll out new networks and changes step by step without forcing large or disruptive cutovers.

In production, both the application and its underlying components like database, messaging layer have been upgraded without any system downtime.

Today, this model processes over 50 million payment transactions a month, well over 1.5 million per day, while running continuously.

More importantly, it showed that even payments, which is one of the most sensitive areas in a bank can be changed safely while systems are live and operating at scale.

5.3 Pricing: Making Change Easier

Pricing was another area where frequent business change creates risk. In traditional setups, pricing logic often sits deep inside product systems or processors. As a result, even small pricing changes require updates across multiple systems, making change slow, coordinated and hard to control.

We addressed this by separating pricing decisions from execution. Pricing rules are defined once at the enterprise level and evaluated using customer, product and transaction context. Execution systems then focus on applying the outcome, rather than calculating prices themselves.

This allows pricing to change independently, without forcing changes in transaction processing systems. Business teams can introduce new pricing structures, offers and adjustments more easily, while stable execution continues as before.

In production, this model operates at very large scale supporting 100 million+ pricing and charge evaluations per day, applied across 80 million+ account balances and 60 million+ master records.

It demonstrated that pricing can evolve continuously, without increasing operational risk or destabilising live systems.

5.4 Originations: Managing Change Over Time

Origination is one of the most change heavy parts of a bank. Customers, products, policies, documents and risk checks come together over multiple steps and timeframes, often involving several external services. Small changes in any of these areas can quickly ripple through the entire journey. To manage this, origination was placed at the enterprise layer and treated as a business process rather than a fixed flow.

Instead of designing rigid journeys, origination flows are broken into smaller steps, with decisions made progressively as information becomes available. This avoids the need to collect everything upfront and make a single decision at the end, and allows the process to adapt as conditions change.

This approach makes it easier to introduce new journeys, bundled offerings and policy changes without rewriting entire flows or forcing coordinated releases across systems. It also helps absorb change from external service providers without destabilising the overall process.

The platform is being built to handle origination volumes of hundreds of thousands of applications per month, across more than 60 external service integrations.

The architecture is designed to scale beyond this, while remaining flexible enough to support continuous change under live conditions.

5.5 Products: Reducing Product Complexity

In large banks, products naturally grow over time. New variants are added to support pricing differences, eligibility rules, regulatory needs or new channels. Gradually, this leads to hundreds, sometimes thousands of product setups spread across multiple systems.

As this complexity builds, even small product changes begin to require updates in several places. What should be routine adjustments turn into slow, coordinated efforts, increasing both effort and risk.

To address this, we placed product definition at the enterprise layer and separated it from core. Products, variants, documents and rules are defined once and shared where needed, while transaction posting systems continue to focus on what they do best, maintaining balances and posting transactions.

As part of an ongoing simplification effort, a product landscape of over 1,000 existing product setups is being examined.

The focus is not on cleaning everything up in one go, but on reducing complexity step by step, without disrupting systems that are already running.

Over time, this reduces product complexity while allowing change to be introduced in a more controlled way.

5.6 Continuity: Operating Through Core Unavailability

Core system unavailability can interrupt payments, impact customer journeys and create operational pressure. As systems move towards real-time, tolerance for such interruptions continues to reduce.

Instead of stopping transaction flows, critical payment journeys continue under controlled conditions, with reconciliation handled once the core becomes available. This allows customer-facing services to remain active even when parts of the backend are temporarily unavailable.

This approach is live in production for real-time domestic payments, operating at approximately 300 transactions per second with response times under 250 milliseconds, and is designed to scale further as adoption increases.

It has also been applied to high-concurrency scenarios, handling over 500 transactions per second on a single account without disruption.

Adoption has been introduced gradually across channels and transaction types, without requiring changes to existing customer journeys. This allows banks to continue operating reliably even when core systems are temporarily unavailable.

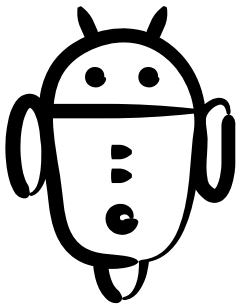
6 Our Approach to AI: Purposeful & Controlled

AI is becoming part of almost every conversation in banking. At the same time, banking operates in a highly regulated environment where predictability, explainability and accountability are essential.

From the start, our position has been clear. Using AI everywhere is not the answer. Using AI in the right places, with clear boundaries, is. That principle shapes how AI is used across the platform. At the core of banking operations such as transaction posting, pricing outcomes, eligibility decisions, the outcomes must remain deterministic. They need to be predictable, traceable and auditable. For this reason, AI is deliberately not used to make core transactional decisions. These paths remain rule-based and explicit.

AI is used selectively to support people and improve efficiency, only in contexts where it adds clear value and can be governed responsibly, without taking control away from the bank.

In some cases, it can support users in an agent-like manner by helping with information gathering and follow-ups, while remaining fully under human control.



AI is applied primarily in banker-facing and operational contexts. For example, AI can help explain how a particular fee or charge was computed by summarising the relevant rules, inputs and outcomes in clear language. The calculation itself remains deterministic;
AI supports interpretation and communication.

Another area being explored is assisted credit decisioning. Here, the intent is not to automate decisions, but to support authorised users by highlighting relevant information or summarising complex data. Final decisions continue to rest with humans.

Across all these use cases, AI is introduced gradually, with clear ownership and accountability. It is never allowed to bypass governance, consent or audit requirements.
As the banks gain confidence, they can expose the AI based functionality gradually to their customers as well.

Closing Thoughts

This document is not a blueprint for how banks should rebuild everything. It reflects how we approached change in a live, regulated environment carefully, incrementally and with respect for what already works.

One idea runs through the choices described in this paper. Large banks do not struggle because they lack ambition or intent. They struggle because change is often pushed into systems that were never designed to absorb it.

Our focus was therefore not on accelerating change, but on reducing the friction around it by keeping transaction execution stable, and moving change to the right layers.

Over time, this work has shaped how we think about transformation in large banks.

In practice, change rarely succeeds when it is forced directly into tightly coupled cores.
What we have seen instead is a gradual progression from reducing coupling around the core, to introducing shared enterprise capabilities and eventually to a state where change becomes continuous rather than episodic.

The intent is not to force this progression, but to allow it to happen safely, while systems remain live and compliant.

In this context, progressive transformation is not about speed.

It is about Confidence that

- **New capabilities can be introduced without breaking what already runs the bank.**
- **Systems can coexist, scale and recover under real-world conditions.**
- **Architectural decisions hold up not just on paper, but in operation.**

Ultimately, this approach allows banks to keep moving forward without betting everything on a single moment of change. Transformation becomes something the architecture can sustain. It is continuous, controlled and survivable rather than a one time event that carries unacceptable risk.

This is the outcome we set out to support through our architecture.