OnArrival for Hotel platform

Ranking intelligence for invisible platforms, a PM case study.

Authored by - Vibha Kamath

Date - 30th March, 2026

—---

Table of Contents

Problem Depth - Page 2-5

Product Proposal - Page 6-17

Decision Variable and Tradeoffs - Page 18-23

Experience Design - Page 24-44

Metrics - Page 45-48

Rollout Strategy - Page 48-51

PROBLEM DEPTH

What is OnArrival and where does it sit -

OnArrival is a travel technology platform that sits between fragmented hotel supply and enterprise customers: fintechs, banks, and consumer platforms, who want to embed hotel booking inside their existing products without building supply infrastructure themselves. OnArrival is not a consumer OTA. The end user never books on OnArrival directly, they book inside other apps or portals that are powered invisibly by OnArrival underneath. This distinction matters because it shapes every product decision. OnArrival's success is entirely dependent on whether their enterprise customers' products succeed.

The specific problem with Hotel supply -

Hotel supply is structurally messy in ways that are invisible to the end user but deeply consequential for the platform. The same property frequently appears across multiple suppliers, each with different pricing, different content quality, different cancellation terms, and different booking reliability. Prices and availability change rapidly between the moment a user searches and the moment they attempt to book. Post-booking failures are common, a confirmed booking can unravel at check-in. And critically, the cheapest available option is not always the best option, a low price from an unreliable supplier can convert well at the booking step but create a support escalation, a failed check-in, or a frustrated user at the other end.

The problem OnArrival needs to solve is therefore not simply which hotel to show, but two connected decisions, first, which properties to surface and in what order for a given user's context, and second, which supplier's version of each property to show and ultimately route the booking through, when multiple supplier versions exist with different tradeoffs across price, reliability, content accuracy, and margin.

The Stakeholders and their conflicting needs -

1. **The End Traveller** needs reliability above all. They want the hotel to match what was shown, the booking to not fail, and no surprises at check-in. They are the most vulnerable stakeholder because they bear the direct human cost of a bad ranking

decision, and yet they have the least visibility into why a particular option was shown to them.

2. **The Enterprise Customer's Business team** needs conversion, strong margins, and low support burden. They want a travel product their users trust enough to keep returning to, which means “end user retention” is ultimately their measure of success, even if they track it as repeat booking rate or support ticket volume.

3. **The Enterprise Customer's Developer** needs the ranking logic to be configurable and predictable. They need to pass runtime signals: user segment, price sensitivity, trip purpose, and receive consistent and explainable results they can build a product on. A black box ranking system they cannot influence or understand creates a product they cannot confidently ship or debug.

4. **The Supplier** needs fair representation of their inventory and sufficient booking volume. Also, Suppliers have a reason to improve their reliability, but only if OnArrival gives them visibility into why they are being deprioritised. Without that transparency, a Supplier losing volume has no way of knowing what to fix.

These needs conflict in practice. The cheapest option drives conversion but frequently comes from a less reliable supplier. The highest margin option may not be price competitive enough to convert. The most reliable supplier may not always have the best content quality. And enterprise customers want control over ranking rules, but unconstrained partner control can override platform intelligence in ways that ultimately damage their own users' experience

These conflicts do not resolve themselves. OnArrival must build a ranking logic that makes deliberate choices about how to balance them, not differently for each customer, but through a single shared system that each enterprise customer can configure for their own commercial priorities and user needs.

Why this is specifically hard as a platform -

1. OnArrival has no direct consumer relationship. When a booking fails or a hotel doesn't match expectations, the end user contacts the Customer’s support team, not OnArrival. OnArrival hears about failures indirectly, filtered through their enterprise customer, often

after trust has already been damaged. This creates a data lag that a consumer OTA like MakeMyTrip does not face, they hear from unhappy users immediately and can act on it quickly.

2. OnArrival starts with imperfect data. Unlike an established OTA with years of booking history per supplier, OnArrival has to make ranking decisions early on with borrowed signals: supplier reputation, third party reviews, content quality assessments, rather than its own observed reliability data. Ranking decisions at launch are inherently a calibrated leap of faith, not a data-driven certainty.

3. Different enterprise customers can have fundamentally different needs from the same ranking logic. The ranking logic cannot be one-size-fits-all. It must be configurable at multiple levels, across different enterprise customers and across different user cohorts within each customer, without requiring OnArrival to build a completely separate system for each one.

4. OnArrival cannot fix supply quality directly. They can deprioritise an unreliable supplier, route around them, or eventually offboard them, but they cannot reach into a supplier's system and fix their reliability. The platform's intelligence has to compensate for supply-side failures it cannot control, and do so in a way that is invisible to the end user.

What happens if we solve this badly -

1. If the system over-optimises for conversion and price, cheap but unreliable options get surfaced consistently. Bookings convert well initially. But post-booking failures accumulate: content mismatches, failed check-ins, cancellation surprises. End users start raising support tickets with the Enterprise Customer. The Customer's support costs rise and their travel product's reputation suffers. End users lose trust and stop booking through the Customer's app. The Enterprise Customer blames OnArrival, escalates commercially, and either demands fixes under contractual pressure or churns entirely. OnArrival loses the contract.

2. If the system over-optimises for reliability and safety, only the most conservative supplier options get surfaced. Prices look uncompetitive compared to MakeMyTrip or direct hotel

booking. End users don't convert because they find better prices elsewhere. The Enterprise Customer’s product sees poor adoption and might question the ROI of the OnArrival integration. OnArrival still loses the contract, just for a different reason and on a slower timeline.

3. The third failure mode is subtler but equally damaging. If OnArrival gives enterprise customers unlimited manual control over ranking without guardrails, well-intentioned but commercially motivated overrides can quietly deprioritise user experience without anyone realising it. This happens not because enterprise customers want to harm their users, they do not, but because the person configuring the override is optimising for a short term metric like margin, the consequences show up slowly in a different team's dashboard, and the enterprise customer simply does not have access to the supplier reliability data that would change their decision. Over time these overrides degrade the quality of platform intelligence, make supplier reliability data harder to trust, and erode end user confidence, not just for one partner but across the platform as a whole. This is why guardrails are not a restriction on partner autonomy. They are a protection for partners against the unintended consequences of their own decisions.

What good looks like, the north star -

The north star for this capability can be “end user retention”, measured as “repeat booking rate” through the enterprise customer's product over time. This is the lagging indicator that confirms the entire system is working as intended.

But retention is slow to move and indirect. The faster, more actionable signals that predict it are “trust erosion events”, measurable moments where the platform failed the end user:

- Booking failure rate after payment confirmation
- Post check-in content mismatch complaints(ex - photos do not match, or mentioned amenities not available at the location, etc.)
- Support ticket rate per completed booking(no.of support tickets raised/total no. of completed bookings)
- Cancellation rate driven by expectation mismatch rather than user choice.

To solve this, OnArrival needs to build a capability that sits at the heart of every hotel search and booking decision, *the Inventory Intelligence Engine.*

PRODUCT PROPOSAL

What we are building

The Inventory Intelligence Engine is a ranking and decisioning system that sits underneath OnArrival's hotel inventory layer. Its core job is to evaluate every available supplier version of a hotel option across multiple signals and produce a ranked list that balances conversion, end user trust, and commercial outcomes simultaneously.

The first problem it solves is property ranking, which hotels should appear in a user's search results and in what order. Not every hotel in a destination is equally relevant for every user.The same destination searched by two different users can warrant a different ranked list depending on the signals the engine receives about each user's context. Property ranking decides what the user sees based on that context.

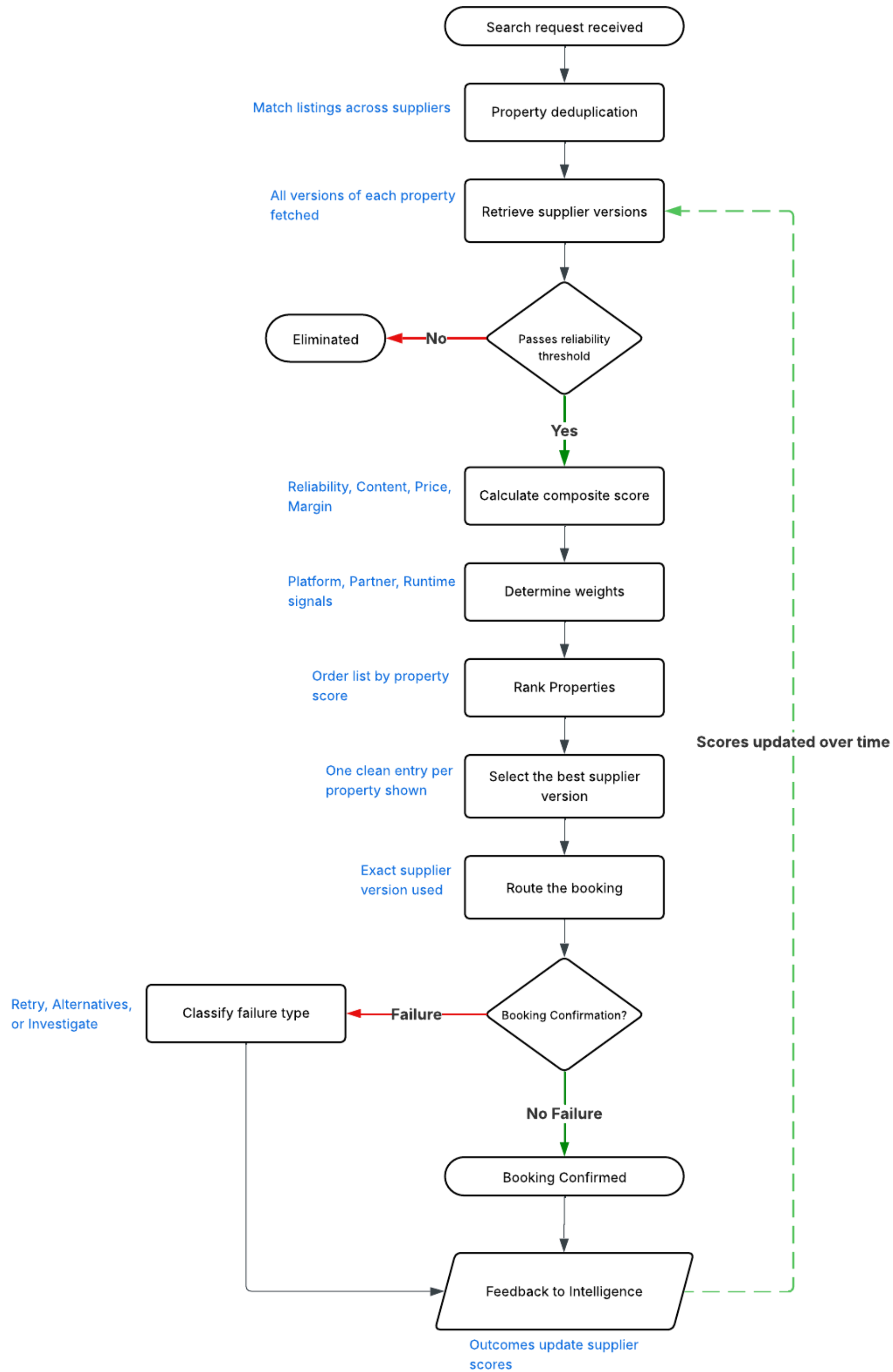
The second problem is supplier version selection, for each property that appears in the results, which supplier's version should represent it. The same Taj hotel in Mumbai might appear from three different suppliers, each with different pricing, cancellation terms, content quality, and reliability history. These are three meaningfully different options even though they are nominally the same hotel. Supplier version selection decides which of those versions the user sees and which one the booking is routed through.

The user sees one clean ranked list, one entry per property. What they do not see is the engine's work underneath, evaluating multiple supplier versions per property, selecting the best one for their context, and ordering the full list by property relevance. That invisibility is the point. The complexity lives in the engine. The user experiences a clean, trustworthy result.

Alternatives considered and rejected

Approach	What it is	Why rejected or chosen	Future role
Simple fixed weighted scoring - Rejected	Static formula with fixed weights applied uniformly to every decision regardless of user or partner context.	Cannot support cohort level personalisation. A premium business user and a price sensitive leisure user receive identically weighted results. Cannot adapt as partner needs evolve without manual reconfiguration.	No future role. The configurability gap is structural, not a timing issue.
Dynamic algorithmic scoring - Rejected for V1	ML- driven model that learns optimal weights automatically from booking outcomes without manual configuration.	Requires abundant historical booking data that OnArrival does not have at launch. Significant data science and engineering investment exceeds available V1 bandwidth.	Natural evolution path for V3 once sufficient platform data has accumulated.
Configurable weighted scoring - Recommended	Partners configure baseline weights at onboarding. Runtime signals adjust weights per user within the partner's range. Platform guardrails apply minimum reliability thresholds regardless of configuration.	Works with limited data at launch. Supports multi-level configurability without building separate systems per partner. Extensible toward dynamic scoring as data and engineering capacity grows.	Foundation for V1 and V2. Transitions to ML-driven scoring in V3.

How the engine works - the mechanism



Property deduplication

Before the engine can compare supplier versions of the same property, it first needs to correctly identify which listings across different suppliers represent the same physical hotel. The engine does this by combining multiple matching signals, latitude and longitude coordinates as the primary signal, supported by property name similarity, standardised industry property identifiers where available, and confirming signals like phone number and official website. These signals are combined into a match confidence score. High confidence matches are automatically deduplicated, and moderate confidence matches are flagged for manual review. Deduplication accuracy improves over time as supplier data quality improves and the operations team resolves flagged cases.

Step 1 - Retrieve all supplier versions: For every property relevant to the search request, the engine retrieves all available supplier versions. A search for hotels in Mumbai may return the same Taj property from three different suppliers, each with different pricing, cancellation terms, content quality, and reliability history. These are treated as distinct options at this stage, not duplicates. The engine needs to see all versions before it can select the best one.

Step 2 - Apply the minimum reliability threshold: Before any scoring begins, the engine eliminates supplier versions whose composite reliability score falls below the platform's minimum threshold. This is a hard guardrail. No partner configuration, no runtime signal, and no commercial override can bring an eliminated option back into ranking. A supplier version that fails this check does not get scored. It does not appear in results. The end user never sees it.

This step is what protects the platform's integrity regardless of what any individual partner configures. It is also what makes guardrails a protection for partners rather than a restriction, they are prevented from inadvertently surfacing options that would damage their own users' experience.

Step 3 - Calculate a composite score for each remaining option: Each surviving supplier version receives a composite score calculated from four signals. All signals are normalised to a 0 to 100 scale before being combined, this ensures signals measured in different units(percentage, currency, ratio, etc.) can be meaningfully compared and added together.

Composite Score = (Reliability Score × Reliability weight) + (Content Quality Score × Content weight) + (Price Competitiveness Score × Price weight) + (Margin Score × Margin weight)

All four weights sum to 1. The weights are not fixed, they shift based on user context and partner configuration as described in the step 4.

Step 4 - Determine the weights: This is where the engine's intelligence lives. Weights are determined by combining three inputs in a defined hierarchy - Platform guardrails, Partner configuration, and Runtime signals, in the same order.

Platform guardrails - This is set by OnArrival, which applies universally, and cannot be overridden. Define minimum reliability threshold and maximum weight ranges for each signal. These exist to ensure no partner configuration can produce outcomes that damage end users or the platform's overall integrity.

Partner configuration - This is set by the enterprise customer at onboarding. Defines baseline weight preferences for their product, as to how much to weight reliability versus price versus margin overall, and how those weights should differ across user cohorts within their product. A reliability-first partner sets high reliability weights as their baseline. A price-first partner sets higher price weights. Both operate within the platform guardrails.

Runtime signals - This one is passed by the developer for each specific user at the time of the search request. User segment, price sensitivity, and a summary of the user's stay history, such as average property tier booked and past cancellation behaviour, adjust the weights within the partner's configured range for that individual user.

For a premium segment user with low price sensitivity, the engine applies:
Composite Score = (Reliability Score × 0.40) + (Content Quality Score × 0.35) + (Price Score × 0.10) + (Margin Score × 0.15)

For a price sensitive user from the same partner, the weights shift:
Composite Score = (Reliability Score × 0.30) + (Content Quality Score × 0.20) + (Price Score × 0.35) + (Margin Score × 0.15)

*Please note, the weights used in these equations are illustrative. Actual weights would be determined by OnArrival at the platform level and configured by each enterprise customer within the permitted range

based on their commercial priorities and user cohort needs.

Reliability always carries meaningful weight because the platform minimum threshold applies regardless of user context. Margin weight stays relatively stable because it reflects the enterprise customer's commercial interest which does not vary by individual user.

Step 5 - Calculate the property score and rank the list: With supplier version scores calculated, the engine now determines the property level ranking, which hotels appear and in what order.

Each property's score is derived from the best available supplier version score for that property, combined with property level relevance signals, location match to the search query, star rating alignment to user segment, amenity match to user’s segment and stay history, and the property's overall reliability track record across all suppliers.

Properties are ordered highest to lowest by their property score. This determines the order of the list the user sees.

Tiebreaker logic - When two properties have identical composite scores, the engine applies a pre-defined tiebreaker hierarchy:

- First tiebreaker: Reliability score
- Second tiebreaker: Price competitiveness
- Third tiebreaker: Content quality
- Final tiebreaker: Longest track record on OnArrival's platform

The tiebreaker hierarchy is configurable at the partner level; a margin-first partner can set margin as the primary tiebreaker, ensuring partner configuration is respected consistently across all ranking decisions.

Step 6 - Select the best supplier version for each property: For each property in the ranked list, the engine selects the highest scoring supplier version to represent that property. This is the version the user sees, the price, the content, the cancellation terms. This is also the version the booking will be routed through if the user proceeds.

The user sees one clean entry per property. The engine's supplier selection work is invisible. What the user experiences is a trustworthy, well-ranked list of hotels, each one backed by the best available supplier version for their specific context.

Step 7 - Route the booking: When the user selects a property and proceeds to book, the engine routes the transaction through the specific supplier version that was ranked, selected, and shown. Not any supplier for that hotel, the exact supplier version the user saw, evaluated, and made their decision based on. This consistency between what was shown and what is delivered is foundational to end user trust.

Step 8 - Fallback logic at booking failure: If the selected supplier returns an error at the booking step, the engine classifies the failure before deciding what to show the user. This classification step is critical, two different failure types require two completely different responses.

If the failure is identified as a transient technical error - supplier API timeout, momentary connectivity issue, a system failure unrelated to actual availability, the inventory likely still exists. The engine surfaces an honest non-alarming message: *"Something went wrong on our end. Your booking was not confirmed and you have not been charged."* A prominent retry option is shown for the same property. Alternative options ranked by the engine appear below, visible but secondary, since the room is likely still available.

If the failure is identified as an inventory shortage - the supplier returns a specific no-availability response and a re-check confirms the room is genuinely gone, retrying will only frustrate the user. The engine surfaces a different message: *"This room is no longer available. It may have just been booked by someone else."* No retry option is shown. Alternative options become the primary focus, ranked by the same signals and weights that produced the original result, so they feel like genuine recommendations.

If the engine cannot determine which failure type occurred, it allows one retry attempt with an honest message, '*Something went wrong. Please try again*.' If the second attempt also fails, the retry option is removed and alternative options become the primary focus. Two consecutive unidentified failures from the same supplier trigger an internal investigation. If the failures are traced to the supplier's API instability, OnArrival notifies the supplier and gives them a defined window to resolve the issue. If the instability persists or recurs frequently despite notification, a

negative signal is logged against their reliability score, which over time affects their ranking position on the platform.

All identified failures - (i). Transient errors traced to the supplier, and (ii). Confirmed inventory shortages are logged and fed into the platform's ongoing supplier intelligence, ensuring every booking event contributes to more accurate scoring and better ranking decisions over time.

Step 9 - Feed outcomes back into platform intelligence: Post-booking outcomes update the engine's platform intelligence scores continuously.

- Did the booking confirm successfully without failure?
- Did the user raise a support ticket after booking?
- Did the post stay experience match what was shown?
- Did the user cancel due to expectation mismatch rather than a genuine change of plans?

This closed feedback loop is what makes the engine genuinely intelligent over time. At launch it operates on proxied signals and conservative defaults. As bookings accumulate, observed data replaces proxies, scores become more accurate, and ranking decisions improve with every completed stay.

The Two Operating modes -

The engine does not operate the same way on day one as it does six months after launch. Its behaviour evolves as real booking data accumulates.

Launch mode: At launch OnArrival does not have its own booking history to rely on. So the engine starts with the best available substitutes - *information suppliers share during onboarding, industry benchmarks, and early technical testing*. Price and margin scores work from day one since those only need supplier pricing data which is already available. For everything else, new suppliers start with a middle range score, not assumed to be great, not assumed to be bad, and prove themselves through actual bookings over time. The rankings at launch will not be perfect but they will be safe and reasonable.

Mature mode: Over time this changes. Every booking that happens on the platform, whether it succeeds, generates a complaint, or leads to a cancellation, feeds back into the engine and updates each supplier's scores with real evidence. The more bookings accumulate, the more

accurate the scores become, and the better the engine gets at surfacing the right option for the right user.

How scores are calculated -

All four signals are normalised to a 0 to 100 scale before being combined. This ensures signals measured in different units can be meaningfully compared and added together in the composite score formula.

Reliability Score

Reliability Score = (Booking success rate × 0.35) + (Post check-in complaint rate × 0.25) + (Support ticket rate × 0.20) + (API uptime rate × 0.15) + (Cancellation dispute rate × 0.05)

*Please note, the weights used in this equation are illustrative.

Booking success rate carries the highest weight because a failed booking is the most direct trust erosion event. Support ticket rate and API uptime rate together capture the operational cost a supplier imposes on the enterprise customer and the friction they create in the booking experience. Cancellation dispute rate carries the lowest weight because it is partly driven by user behaviour rather than supplier quality alone.

Some signals work in reverse, lower is better. Support ticket rate and complaint rate are examples. A supplier with fewer support tickets is more reliable than one with more. To keep all signals on the same 0 to 100 scale where higher always means better, these signals are flipped before being applied. A supplier generating 8 support tickets per 100 bookings scores 92. A supplier generating 20 tickets per 100 bookings scores 80. The worse the rate, the lower the score, consistent with every other signal in the formula.

The platform reliability threshold is set by OnArrival at a defined minimum score, for example 70 out of 100. Supplier versions scoring below this threshold are eliminated in Step 2 before scoring begins. The exact threshold would be determined by OnArrival based on their platform data and acceptable risk tolerance, and reviewed periodically as more booking data accumulates.

Content Quality Score

Content Quality Score = (Content completeness × 0.30) + (Photo quality and recency × 0.25) + (Cross supplier consistency × 0.25) + (Post check-in mismatch rate × 0.20)

*Please note, the weights used in this equation are illustrative.

Content completeness measures what percentage of required fields are populated. Photo quality and recency considers both resolution and how recently photos were updated. Cross supplier consistency checks whether this supplier's content aligns with other suppliers for the same property, large discrepancies flag potential inaccuracy and are penalised. Post check-in mismatch rate is the most direct signal but requires booking history to accumulate.

Content quality scoring is partially available from day one, completeness and photo assessment can be run programmatically against supplier data without any booking history.

Price Competitiveness Score

Price Competitiveness Score = (Within platform price ranking × 0.60) + (Market rate comparison × 0.40)

*Please note, the weights used in this equation are illustrative.

Within platform price ranking compares this supplier's price against other suppliers offering the same property. Market rate comparison checks how the platform price compares to the same property on public booking platforms. This score is fully available from day one.

Where a property is not listed on public booking platforms, the market rate comparison sub-signal is unavailable and the score is calculated from within platform price ranking alone.

Margin Score

Margin Score = (Supplier margin / Maximum available margin for this property) × 100

The supplier offering the highest margin for a given property scores 100. Others are scored

proportionally. The scores reflect how close each supplier is to the best available margin, not an absolute measure of margin itself. Let’s understand what we mean by proportional here, with a small sample example -

Three suppliers offer the same property with different margins:
Supplier A = 15% margin; Supplier B = 18% margin; Supplier C = 20% margin

Now, since supplier C has the highest margin, which is 20%. So, Supplier C scores 100. And, every other supplier is scored as a proportion of that 20%:

Supplier A = (15 / 20) × 100 = 75

Supplier B = (18 / 20) × 100 = 90

Supplier C = (20 / 20) × 100 = 100

Supplier A offers 15% margin, the best available is 20%. Supplier A is offering 75% of the best possible margin for this property, so they scored 75.

Supplier B offers 18% margin. That is 90% of the best available. So they scored 90.

Supplier transparency and feedback -

The engine's intelligence only improves if suppliers understand why their scores are changing and have a reason to act on that information. OnArrival therefore can provide each supplier with visibility into their performance scores through a supplier-facing dashboard, showing their reliability score, content quality score, booking volume, and how they compare to the platform benchmark.

When a supplier's score drops below a defined threshold, OnArrival proactively notifies them with specific signal level detail, not just "your reliability score dropped" but "your booking failure rate has increased from 4% to 11% over the last 60 days." This gives the supplier a specific problem to fix rather than a vague signal to interpret.

This transparency serves OnArrival's interests as much as the supplier's. A supplier who understands why they are being deprioritised has a reason to improve. A supplier who loses volume without explanation has no reason to do anything except look for other distribution

channels. Supplier transparency is not generosity, it is how OnArrival will continue to maintain a healthy, competitive supplier ecosystem over time.

Why Configurable Weighted Scoring is the right foundation

The four challenges from the problem framing directly shaped this recommendation.

1. **OnArrival starts with imperfect data** - Configurable weighted scoring works at launch with approximated signals and improves progressively as real booking data accumulates. A dynamic machine learning model would have nothing to learn from without booking history.

2. **Different enterprise customers have fundamentally different needs** - The three layer weight system accommodates any partner configuration at both partner level and cohort level without building separate systems. A fixed weight system could not do this without manual reconfiguration for every new partner or cohort rule change.

3. **OnArrival cannot fix supply quality directly** - The minimum reliability threshold and continuous score updating allow the engine to compensate for supplier side failures it cannot control, routing around unreliable suppliers automatically and invisibly without the end user ever knowing.

4. **Engineering bandwidth is limited** - Configurable weighted scoring is designed to be built in phases, with each phase delivering value independently. The full capability does not need to be complete before the first enterprise customer can go live.

With the engine's design and decision making approach established, the next question is what variables it should weigh and what tradeoffs it must navigate, which is where the real product judgment lies.

DECISION VARIABLES & TRADEOFFS

The Inventory Intelligence Engine will make hundreds of ranking decisions every day. Each decision has to be shaped by a set of variables, some fixed by the platform, some configured by partners, some passed in real time by developers. Understanding which variables matter, when they matter the most, and what each prioritisation sacrifices is where the real product judgment lives.

The Primary decision variables

Four signals drive every supplier version score the engine calculates. They are not equal in importance and they are not all configurable in the same way.

Variable	What is it optimized for?	Who does it primarily serve?	Configurable?
Reliability	Booking success and post stay trust	End traveller	Partially, has a platform threshold that cannot be overridden
Content quality	Accurate representation of the property.	End traveller	Yes, weighted by partner and runtime signals
Price competitiveness	Conversion and perceived value	End traveller and Enterprise customer	Yes, weighted by partner and runtime signals
Margin	Commercial return per booking	Enterprise customer	Yes, weighted by partner configuration

Reliability is the only variable with a non-negotiable minimum threshold. Every other variable is configurable within partner and platform defined ranges. The reason is simple, reliability is the only variable whose failure cannot be fixed in the moment. A wrong price can be corrected. A confusing photo can be explained. A booking that fails at check-in cannot be undone. The end user is stranded. No amount of support resolves that experience immediately.

This is why reliability is never fully traded away regardless of any other signal. It is the foundation everything else is built on.

The Contextual variables

The four primary variables do not carry equal weight in every situation. Context determines which matters most, and context is revealed through a second layer of variables that the engine uses to adjust weights dynamically.

Contextual signal	What it reveals?	Available at launch?	Becomes richer over time?
User segment	The user's tier within the enterprise customer's product, premium, standard, corporate	Yes, known from existing profiles.	No, it’s a static signal
Price sensitivity	How much the user prioritises price relative to other factors	Yes, inferred from segment or passed explicitly	Partially, can be enriched by stay history
Stay history	Average property tier booked, price ranges chosen, property types preferred	No, it requires booking history to accumulate	Yes, it's the primary intent signal at maturity
Search behaviour	Number of properties viewed, time spent on detail page, amenity filters applied	Partially, available within session	Yes, the patterns emerge over time

The key insight here is that contextual signals have a maturity curve. At launch the engine relies primarily on who the user is, their segment within the enterprise customer's product. Over time it learns what the user does, their booking patterns, preferences, and behaviour. The engine's

ability to infer user intent and adjust variable weights accordingly improves as platform data accumulates.

This means the engine makes better decisions on day 1000 than on day one, not because the variables change, but because the signals that determine their weights become richer and more accurate over time.

The Hidden cost variable

There is a fifth variable that does not appear in the composite score formula directly but influences ranking in a meaningful way, “support cost per booking”.

Two suppliers can have similar headline reliability scores but generate very different support burdens. Supplier A with an 85 reliability score might generate 2 support tickets per 100 bookings. Supplier B with the same score might generate 9 support tickets per 100 bookings. Their composite scores look similar. But Supplier B is quietly costing the enterprise customer four times more in operational overhead per booking, overhead that does not show up in conversion metrics but shows up in support team capacity, resolution time, and ultimately in the enterprise customer's perception of OnArrival's platform quality.

Support cost per booking is not a separate variable sitting outside the engine's formula, it is already embedded within the reliability score through the support ticket rate sub-signal, ensuring it shapes ranking decisions as a natural part of how supplier quality is assessed.

The Key tradeoffs

Every prioritisation decision sacrifices something. The engine manages these tensions deliberately, it does not eliminate them. Being honest about what each choice costs is what separates a thoughtful platform design from an optimistic one.

Tradeoff	What do you gain?	What do you give up?	How does the engine manage it?
Cheapest option vs safest option	Higher conversion rate	Reliability, cheaper suppliers are often less reliable	Minimum reliability threshold ensures no option below the

			safety threshold is shown regardless of price
Margin vs conversion	Higher commercial return per booking	Price competitiveness, higher margin options are often less price competitive	Margin weight is stable but capped, it influences ranking without dominating it
Partner control vs platform intelligence	Commercial flexibility for partners	User experience consistency, partners may make commercially motivated overrides that harm users	Guardrails ensure partner configuration operates within platform defined boundaries
Short term booking rate vs long term trust	More bookings today	End user retention, optimising for immediate conversion can erode trust over time	North star is end user retention not booking rate, the engine is calibrated against the lagging indicator not the leading one
Flexibility vs simplicity - partner level	Partners can configure the engine precisely for their commercial context and user cohorts	Go-live speed, more configuration options mean more setup before the platform delivers value	Core configuration is available and necessary from day one. More sophisticated configuration is layered progressively as partner understanding and platform data matures
Flexibility vs simplicity - developer level	Developers can pass rich runtime signals for precise personalisation	Integration confidence, a deeply flexible API with many parameters is harder to learn, debug, and build on	Core runtime signals work from day one. Progressive complexity is introduced through documentation and sandbox guidance as the developer's understanding deepens

Flexibility vs Simplicity - A closer look

This tradeoff operates at two levels simultaneously - Partner level and Developer level, and both matter commercially.

At the partner level, the tension is between configuration depth and go-live speed. A deeply configurable system gives partners precise control over ranking behaviour for different user cohorts and commercial contexts. But too much configuration required upfront creates a barrier to launch, a partner who cannot get the platform working quickly will question the integration investment before they have seen any value.

The engine resolves this by separating configuration into what is necessary at launch and what becomes meaningful over time. Three layers of configuration are essential from day one -
(i). Platform guardrails set by OnArrival
(ii). Partner baseline weights set at onboarding
(iii). Core runtime signals passed by the developer
These are sufficient to produce meaningful, trustworthy ranking results immediately. More sophisticated configuration, fine grained cohort rules, behaviour based weight adjustment, supplier preference rules based on observed platform performance, is introduced progressively as the partner's understanding of the platform deepens and as platform data makes those configurations meaningful rather than arbitrary.

At the developer level, the tension is between API power and integration confidence. A flexible API with many runtime signals and configuration parameters is more powerful but harder to learn. A developer who struggles in the sandbox does not just face a technical problem, they face a commercial one. The developer is an informal evaluator whose experience directly influences whether their product manager or CTO recommends OnArrival to the decision maker. A difficult sandbox experience becomes negative word of mouth before a contract is ever signed.

But the answer is not to simplify at the cost of capability. A developer who integrates quickly but encounters unpredictable results, unclear error messages, or a sandbox that does not reflect real production behaviour will lose confidence slightly later, and their feedback will be equally damaging.

The right standard is speed to a confident first integration, where the developer not only succeeds quickly but understands what happened, why the results look the way they do, and feels equipped to build on top of the platform without fear of surprises in production. The API must work meaningfully with basic configuration from day one, platform guardrails, partner baseline weights, and core runtime signals like user segment and price sensitivity are sufficient to produce trustworthy results immediately. More sophisticated runtime signals and configuration options are layered in progressively as the developer grows their understanding of the platform. Simple enough to build on confidently from day one. Deep enough to grow with over time.

When each variable matters most

Reliability matters most always, it is the non-negotiable foundation and the only variable with a platform floor that nothing can override.

Content quality matters most when the user segment is premium or when stay history reveals a pattern of high expectation bookings, these users are most likely to raise content mismatch complaints and have the lowest tolerance for surprises at check-in.

Price competitiveness matters most when price sensitivity is high and user segment is standard, these users are actively comparing and will leave for a better price elsewhere if the platform does not surface competitive options.

Margin matters consistently but never dominates, it is the enterprise customer's commercial interest and deserves representation in every ranking decision, but never at the cost of reliability or at the expense of end user trust.

Support cost matters most when a supplier's headline reliability score masks a disproportionate ticket generation rate, it is the hidden variable that composite scores alone do not reveal, and the one most likely to create friction in the enterprise customer's operations without a clear traceable cause.

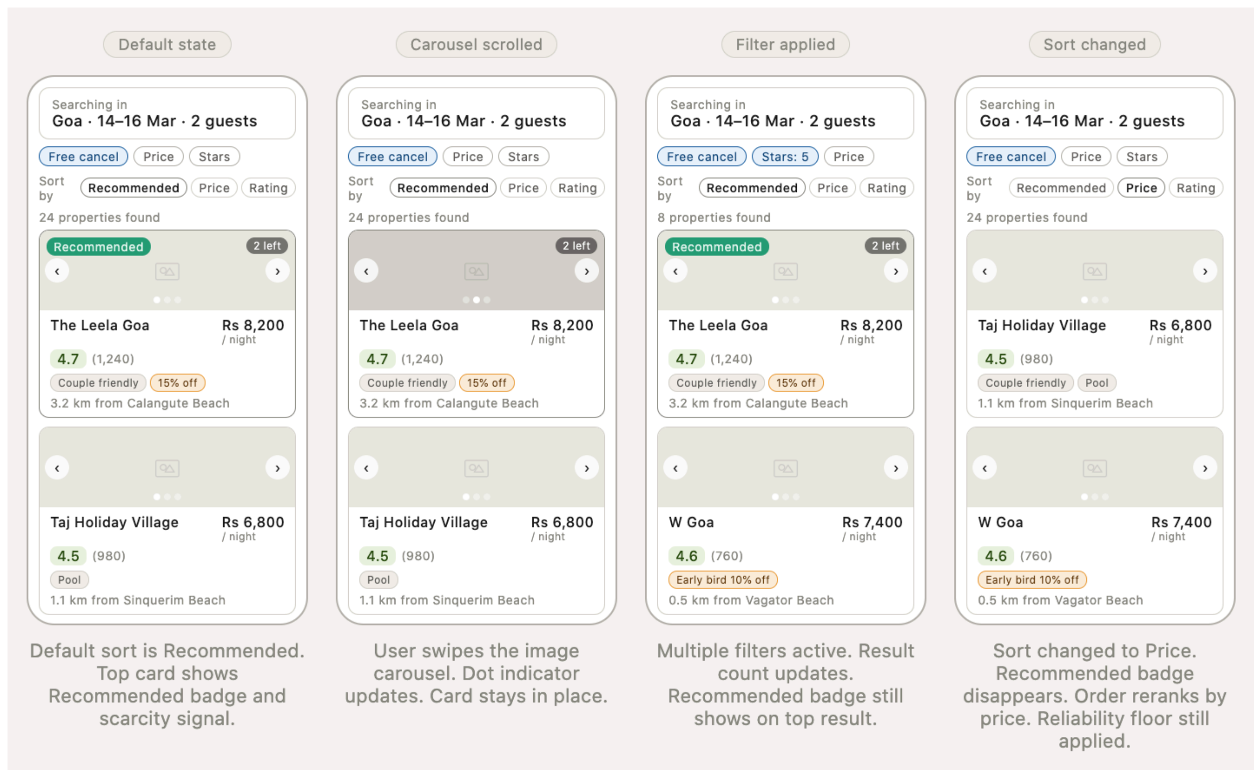
User intent matters progressively, at launch it is inferred from the user segment, at maturity it is revealed through stay history and search behaviour within the session. The engine defers to the richer signal whenever it is available, and improves its inference accuracy as booking data accumulates.

EXPERIENCE DESIGN

Search results page

The search results page is the first moment the engine's intelligence becomes visible to the user. They have entered a destination, dates, and number of guests. They tap search. What appears next either builds immediate trust or loses them to a competitor.

The default ranked list the user sees is the engine's recommendation, ordered by the composite property score calculated from reliability, content quality, price competitiveness, and margin signals. The user does not see any of this. What they feel is that the results are relevant to them, that the platform understood what they were looking for before they had to say it explicitly.



*The 1 screen search result page shown in 4 for better visual purpose. (Full view)

The search results card

Each property appears as a card in the ranked list. The card is designed to give the user enough information to decide whether to tap through, without overwhelming them with detail that belongs on the property page.

Each card shows:

- A cover photo selected to match the user's context, for a leisure user the pool or view, for a premium segment user the room or lobby. The photo is not chosen arbitrarily, it reflects the runtime signals the engine already has about this user. A small carousel allows the user to swipe through two or three additional images without leaving the results page, reducing the need to tap into every property just to evaluate it visually.

- Property name and star rating category

- Overall user rating, aggregated from verified guest reviews

- Price per night inclusive of all guests and rooms in the selected configuration. Showing per night rather than total stay price gives the user a clean reference point and the freedom to mentally adjust based on how many nights they want to stay, since guest count and room count stay fixed while nights is the variable most likely to be reconsidered.

- Contextual tags derived from the user's own search inputs, a user who searched with two guests gets a "Couple friendly" tag where applicable. A user who indicated travelling with a pet sees "Pet friendly." These tags are not generic labels, they reflect the user's stated context back at them, creating the feeling that the platform understood their specific situation.

- One or two location signals relevant to the user's context, distance to the airport for a business segment user, distance to a key landmark or beach for a leisure user. Not both for everyone, only what is relevant to this specific user's context.

- A scarcity signal where genuinely applicable, "Only 2 rooms left" or "Filling fast" based on real inventory data from the supplier. This is useful information not a dark pattern. It

only appears when the data justifies it.

- An active offer or discount label if one exists for this property, the offer name or discount percentage visible on the card so the user does not have to tap through to discover savings.

What the card does not show

Supplier name, reliability score, composite score, or any signal from the engine's internal decision making. The engine's work is invisible. The user experiences its output as a well curated, relevant list, not as an algorithm.

Filters

Filters make part of the ranking logic visible and controllable by the user. They do not expose the engine's scoring, they let the user narrow the result set based on their own explicit preferences:

- Price range
- Star rating category
- Cancellation policy - free cancellation, partially refundable, non-refundable
- Amenities - pool, wifi, breakfast included, gym, pet friendly
- Property type - hotel, resort, villa, airbnb

Filters narrow the result set. Within the filtered set the engine's ranking still applies, the most relevant option for this user within their chosen criteria appears first.

Sort order

The user can change the sort order if they choose - most relevant which is the engine's default, price low to high, price high to low, and highest rated. Allowing sort order changes gives the user flexibility to compare options on their own terms, particularly useful when searching in an unfamiliar destination or travelling with a group where price sensitivity may be higher than usual.

An important design principle here, sorting by price does not expose the user to unreliable options. Every property in the results list has already passed the minimum reliability threshold before appearing. The engine's guardrail operates invisibly underneath regardless of how the

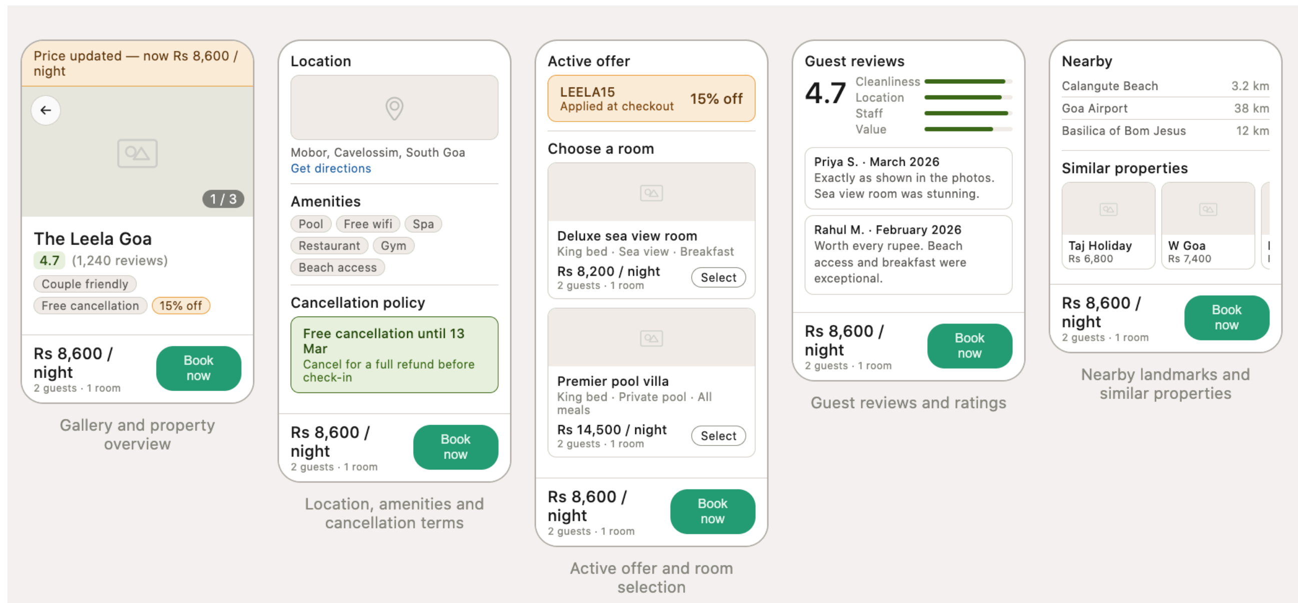
user chooses to sort. A user who sorts by price and selects the cheapest option is choosing from a set that has already been validated for safety. They are exercising informed choice within a curated and trustworthy set.

The recommended signal

The top ranked property in the default sort carries a subtle "Recommended" tag. This is the only signal the user sees that acknowledges the engine's ranking, and it communicates trust without explaining the mechanism. The user does not need to know the composite score. They need to feel that this option has been chosen for them thoughtfully. The recommended tag does that in one word. When the user switches to any other sort order - price, highest rated, or any filter combination, the tag disappears. It belongs to the engine's ranking, not to the user's reordered view. Showing it on a result the user has manually repositioned would create confusion rather than trust.

Hotel details page

The details page has one job, giving the user enough confidence to book. They arrived here already interested. The page's role is not to sell harder but to remove the remaining doubts that stand between interest and commitment. Based on what users need most, two signals carry the most weight at this stage, cancellation terms that reduce the risk of committing, and reviews that validate the property matches what was shown.



*The complete hotel detail's page flow of 1 screen shown in 5 here for better visual purposes, and the persistent "book now" bar which appears throughout.(Full view)

The page opens with a full screen image gallery. This is the first moment the user can see the property beyond the three carousel images on the search card. Tapping any image expands it to full screen. Guest uploaded photos are surfaced alongside the official property photos, these carry particular trust weight because they are unfiltered representations of what the property actually looks like rather than professionally staged shots.

Below the gallery the property name, aggregate rating, and contextual tags carry over from the search card, couple friendly, free cancellation, offer label, so the user does not lose context when transitioning between pages.

Property details follow, a map showing the property's location with a tap to directions link, the full address, a scannable amenities list, and property rules. This section answers the practical questions, where exactly is this, how do I get there, and what does the property allow.

Cancellation terms appear early, before room selection and before pricing detail. This is intentional. A user who sees flexible cancellation terms early feels less risk in continuing to evaluate. A user who discovers non-refundable terms only at checkout feels surprised and sometimes deceived. Surfacing cancellation terms prominently and early is both honest and commercially smart, it reduces abandonment at the booking step.

If an offer or discount is active for this property it appears next, so the user sees the discounted price before they encounter room level pricing. Discovering a discount after already seeing a higher price creates unnecessary confusion.

Room types follow, each room presented as a card with its own image carousel, room level amenities, meal plan options, and price per night for the selected configuration. The engine has already selected the best supplier version for this user, the room cards reflect that version's pricing and terms. The user chooses between room types, not between suppliers.

Reviews appear after room selection, detailed, with a breakdown by category such as cleanliness, location, staff, and value. Guest uploaded photos sit within the review section, showing the property through the eyes of people who actually stayed there. Nearby landmarks follow, relevant to the user's context, beaches and restaurants for a leisure user, business district proximity for a corporate segment user.

Similar properties sit at the bottom, not as an invitation to leave but as a safety net. A user who has read everything and still feels uncertain should be able to compare without going back to the search results page.

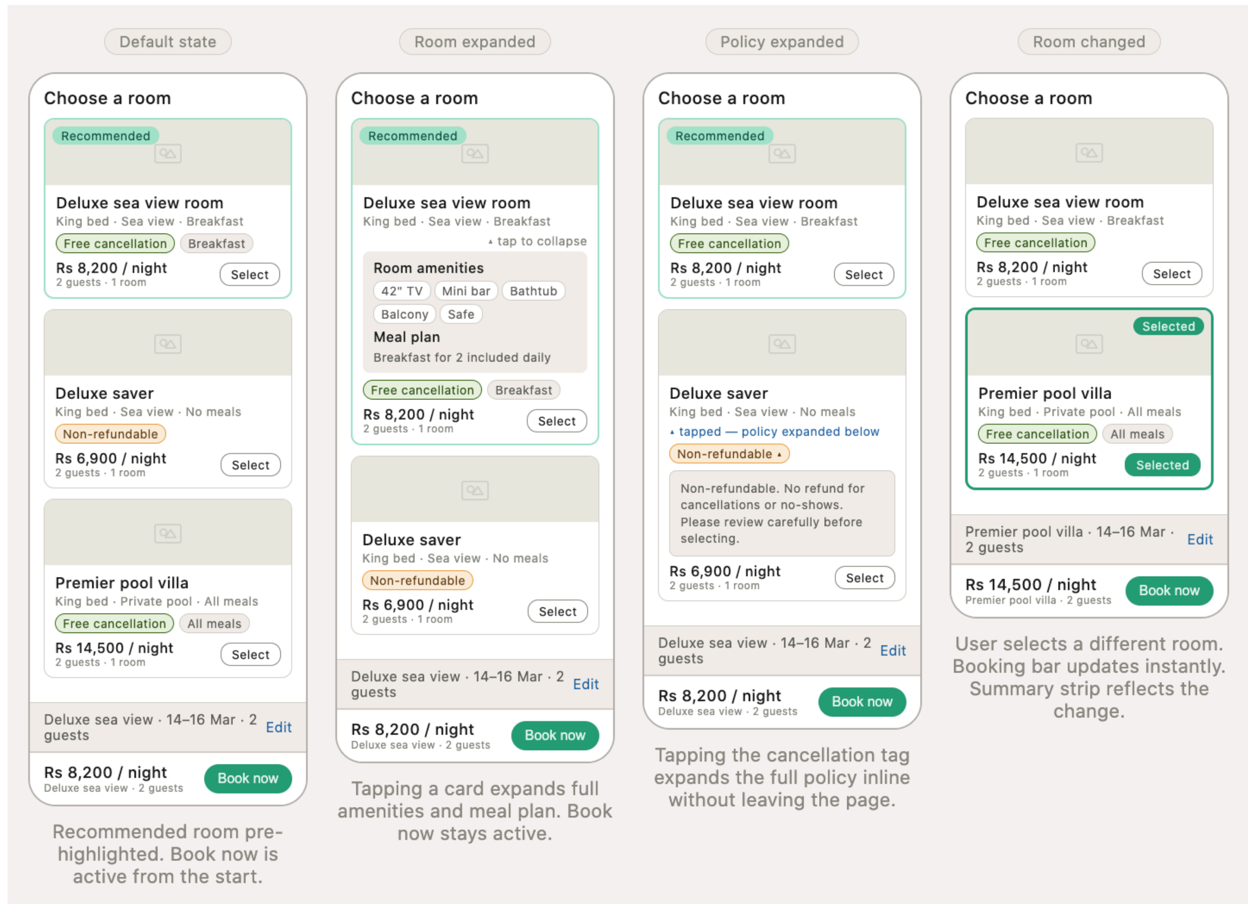
The persistent booking bar

On mobile the booking decision should never be more than one tap away. A persistent bar sits fixed at the bottom of the screen throughout the user's scroll, showing the price per night for the selected configuration and a single Book now button. This bar is visible from the moment the user arrives on the details page to the moment they leave it.

The price shown on this bar is the same price shown on the search results card. If the engine re-validates the price with the supplier in the background and finds it has changed, a clear banner appears at the top of the page immediately, not at checkout. Something like "Price updated since your search, now Rs 8,600 per night." This is honest, non-alarming, and prevents the trust erosion event of a price surprise at the payment step.

Room selection

Room selection happens inline on the hotel details page, not on a separate screen. The user has already committed enough attention to reach this section. Adding a navigation step between evaluating a room and selecting it creates unnecessary friction on mobile. The room cards sit within the details page and the selection happens in place.



*One screen shown in 4 here, showing the room selection journey. (Full view)

The engine has already identified the best supplier version for this property and the recommended room for this user's context. When the user arrives on the details page the persistent booking bar at the bottom already shows the recommended room's price and name, "Deluxe sea view · Rs 8,200 / night", and the "Book Now" button is immediately active. The user can proceed to book the recommended room without interacting with the room selection section at all. Room selection is a choice the user can make, not a step they are required to complete.

Each room is presented as a card with its own image, a one-line summary of key amenities, price per night for the selected configuration, and a cancellation policy tag. The cancellation tag is the most important trust signal at this stage, it tells the user what they are committing to before they commit. A green tag signals free cancellation. An amber tag signals non-refundable or partially refundable terms. The difference is scannable at a glance without requiring the user to read anything. For users who want the full cancellation detail, the tag is tappable and expands to show the complete policy inline within the card; one tap, no new page, no interruption to the flow.

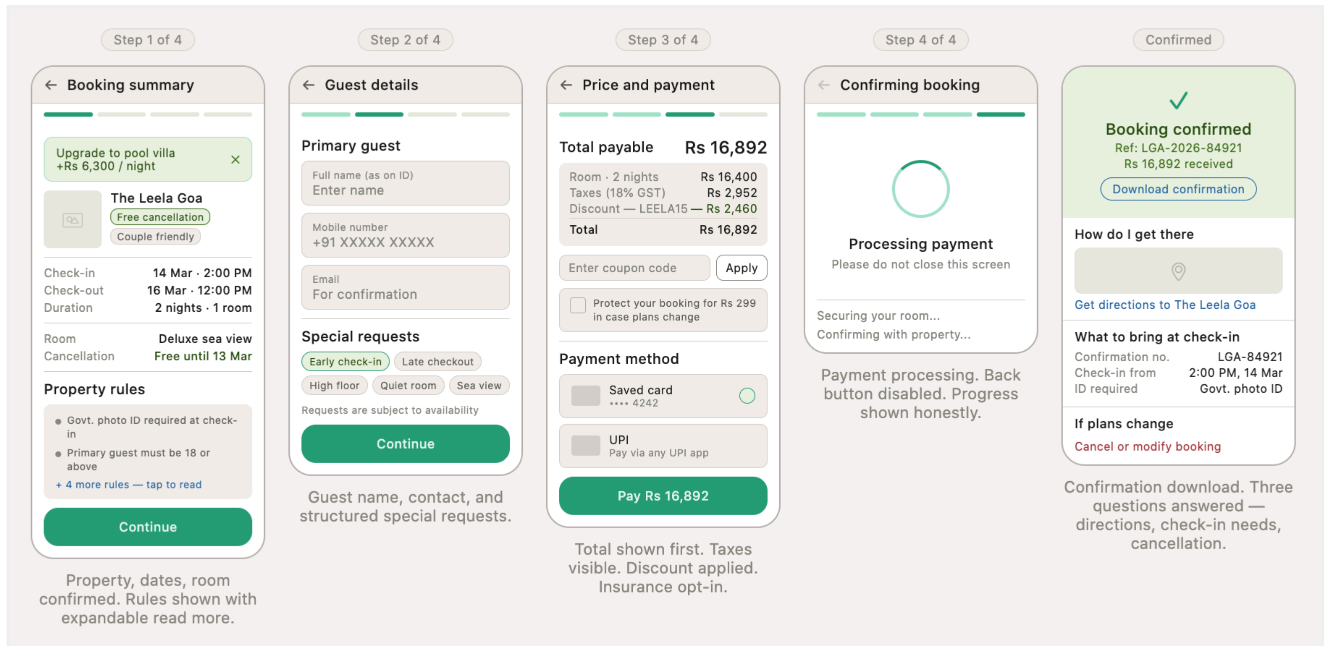
Each room card is also expandable beyond the cancellation terms, tapping the card reveals the full room detail including all amenities, meal plan inclusions, room rules, and additional photos. This keeps the default card view clean and scannable while giving curious users everything they need without leaving the page.

The recommended room card carries a subtle pre-selected visual state when the user first arrives, a lighter highlighted border indicating this is the engine's suggestion for their context. If the user wants a different room they tap Select on any other card and the selection updates immediately. The booking bar updates to reflect the newly selected room's price and name.

A summary strip sits just above the booking bar showing the current booking configuration - room type, dates, and guest count. An Edit option sits inline on this strip. Tapping Edit opens a lightweight modal where the user can change dates, number of guests, or meal plan preference without leaving the page. The room selection and pricing update automatically when configuration changes affect availability.

Booking flow

Tapping Book Now moves the user out of the browsing and evaluation mode and into commitment mode. The experience from this point needs to be fast, transparent, and anxiety-free. Every unnecessary step is a potential abandonment. Every price surprise is a trust erosion event. Every piece of missing information at check-in is a support ticket waiting to happen.



*The complete booking flow from summary to confirmation.(Full view)

Step 1 - Booking summary and confirmation

The first screen after tapping Book Now is not a form. It is a confirmation of what the user is about to book, giving them one final moment to verify everything before entering their details.

This screen shows the property name and key tags - couple friendly, free cancellation, carried over from the details page so context is never lost. Check-in and check-out dates, check-in and check-out times, number of nights, and number of rooms appear clearly. The selected room is restated with its key amenities and cancellation policy, so the user is reminded of what they committed to without having to go back. Property rules and document requirements appear here - what ID is needed at check-in, age requirements, pet policies, so there are no surprises at the property. A subtle room upgrade prompt sits at the top of the screen, not a modal, not an interruption, just a dismissible card offering the next room tier with the price difference clearly stated. The user can ignore it completely without it disrupting their flow.

Step 2 - Guest details

The guest details screen collects the minimum information needed to complete the booking. Name as it appears on a government ID, contact details for the primary guest who must be 18 or older, and any special requests the user wants to communicate to the property - early check-in, late check-out, high floor preference, quiet room. Special requests are presented as

structured options rather than a free text field, structured options are more likely to be acted on by the property and more likely to be completed by the user.

Step 3 - Price breakdown and offers

Before payment the user sees the complete price breakdown - base room rate, taxes, any platform fees, with the total shown prominently at the top. The breakdown is expandable not hidden. Total first, detail on demand. Critically the taxes are not a surprise at this step because they have been included in the price shown throughout the search and details journey. The user has seen the tax-inclusive price from the first search result card. The breakdown at this step is a confirmation not a revelation.

An offer or coupon code entry sits below the breakdown. If a discount was already applied from the search results, “the LEELA15 offer” for example, it appears as already applied with the saving clearly shown. The user can also enter a new code here. The total updates immediately when a valid code is entered.

An insurance option appears below the offers section, opt-in, clearly explained in one line, easy to skip.

Step 4 - Payment

The payment screen is minimal. Saved payment methods appear at the top for returning users. New users enter card details or choose UPI, net banking, or wallet options depending on what the enterprise customer's product supports. The total is restated clearly above the pay button, no ambiguity about what is being charged. The pay button is a single tap. No confirmation modal on top of a confirmation screen.

Step 5 - Booking confirmation

The confirmation screen does not just confirm the booking. It anticipates the three questions every user has the moment a booking is made.

The top of the screen shows a clear payment confirmed message alongside the booking reference number. A booking confirmation is available to download immediately, a shareable document the user can keep offline, print out, or show at check-in if internet connectivity is unavailable at the property. This is particularly useful when travelling to remote locations where

the app may not be accessible.

Below the confirmation, the screen answers the three questions proactively:

How do I get there? - A map thumbnail with a one tap directions link. The user does not need to go back to the property page to find the address.

What do I need at check-in? - The confirmation number displayed prominently, the ID requirement restated, check-in time clearly shown. Everything the user needs to walk into the property confidently.

What if something changes? - A clearly labelled cancellation or modification link. Not buried in a menu. Visible on the confirmation screen itself. A user who needs to cancel should never have to search for how to do it.

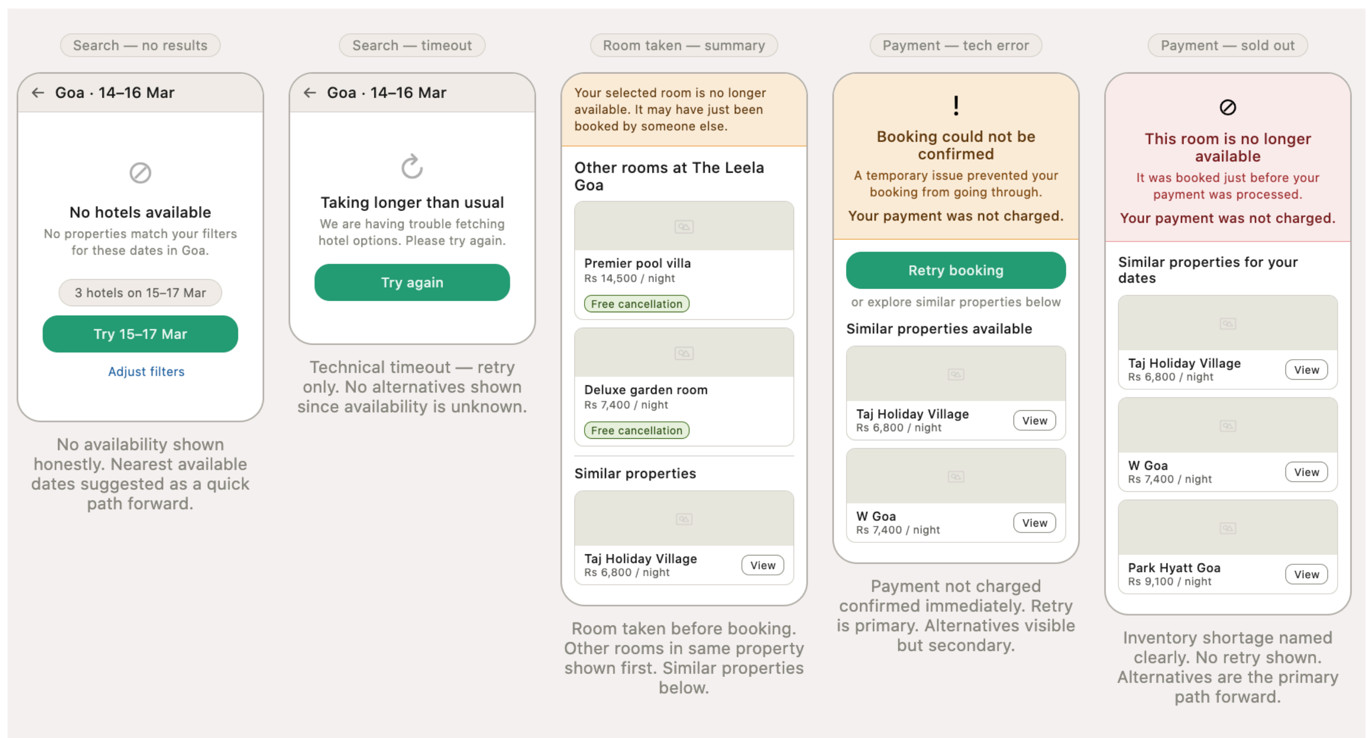
The confirmation is also saved to the user's bookings section within the app automatically, no action required from the user to save their booking details.

Error and fallback experience

Errors in a booking flow are not just technical events. They are emotional ones. A user who has spent time searching, evaluating, and committing to a booking, and then encounters a failure, is in a vulnerable moment. The experience design at this point determines whether they try again or abandon entirely. The goal is not just to communicate what went wrong. It is to remove anxiety, restore confidence, and keep the user moving forward.

The principle that runs through every error scenario

Every error message in this flow follows three rules. It confirms what did not happen, specifically that no payment was taken if the failure occurred at payment. It explains what went wrong in plain language, not a technical error code, not a generic "something went wrong," but a specific honest message the user can understand. And it gives the user a clear path forward - retry, alternative, or change, so they are never left stranded on a dead end screen.



*The key error states across the search and booking journey.(Full View)

Errors at search

Two distinct failures can occur at the search stage and they require different responses:

The first is genuine no availability, the destination has no properties meeting the user's criteria for the selected dates. The experience shows this honestly - "No hotels available in Goa for 14-16 March with free cancellation", and immediately offers two paths: change the dates or adjust the filters. The engine surfaces the nearest available dates as a suggestion - "3 hotels available if you shift to 15-17 March", making the path forward as frictionless as possible.

The second is a retrieval timeout, the engine could not fetch results from suppliers fast enough. This is a technical issue not a genuine availability problem. The experience shows a single retry option with an honest non-alarming message - "Taking longer than usual to find hotels. Please try again." No alternatives are shown here because the engine does not yet know what is available. A retry is the right and only path.

Errors at the details page

Two failures can occur here:

A price change between search and the details page is handled by the banner we established earlier, surfaced immediately at the top of the page before the user sees any other content, honest about the new price, non-alarming in tone.

A content unavailability failure, where the supplier's content API cannot be reached, is handled by showing cached content with a subtle note: "Some details may not be fully updated. We recommend confirming amenities directly with the property." The user can still proceed. They are not blocked by a partial technical failure. But they are told honestly that the information may be incomplete.

Errors between Book Now and the booking summary

When the user taps Book Now the engine does a final availability check. If the selected room has been booked by someone else in the seconds since the user was on the details page, the booking summary screen does not load. Instead the user is returned to the details page with a clear message at the top - "Your selected room is no longer available. It may have just been booked by someone else." The room selection section scrolls into view automatically showing other available rooms within the same property first. If no other rooms are available in the property, similar properties are shown below.

Errors at the payment step - Temporary technical failure

If payment fails due to a temporary technical error, supplier API timeout, connectivity issue, a system failure unrelated to actual availability, the first and most important thing the experience communicates is that no payment was taken. This happens within the first three seconds of the failure appearing on screen.

The message is honest and specific - "Your booking could not be confirmed. Your payment was not charged." A prominent retry button sits below this message. Alternative similar properties appear further down the screen, visible but not the primary focus. The user's intention was to book this specific property and that intention should be honoured first. The retry is the natural first action. Alternatives are there if the user chooses to explore them without being pushed toward them.

Errors at the payment step - Inventory shortage

If the failure is an inventory shortage, the room was genuinely taken between the user initiating payment and the supplier confirming, retrying will not help. Showing a retry option would only frustrate the user further.

The message changes - "This room is no longer available. It was booked just before your payment was processed. Your payment was not charged." The inventory shortage is named clearly and honestly. The apology is implicit in the explanation, the platform is not hiding what happened. Alternative similar properties become the primary focus of the screen, ranked by the same signals and weights that produced the original result so they feel like genuine recommendations. They are labelled "Similar properties available for your dates", informational framing, not a sales pitch.

Errors at the payment step - Unidentified failure

When the engine cannot determine the failure type, it allows one retry with an honest message - "Something went wrong. Please try again." If the second attempt also fails, the retry option is removed. Alternative properties become the primary focus. An internal investigation is triggered on the platform side, invisible to the user but important for supplier reliability scoring.

The design principle that makes alternatives feel helpful not pushy

In every failure scenario where alternatives are shown, they appear after the primary path has been offered or closed. The engine respects the user's original intention, to book this specific property, before redirecting. When alternatives do appear they are always labelled as similar properties available for the same dates, not as promoted options or personalised recommendations. The distinction is subtle but meaningful. One feels like the platform is helping the user achieve their goal a different way. The other feels like the platform is taking advantage of a difficult moment to sell something.

Support experience

When something goes wrong with a hotel booking the end user contacts the enterprise customer's support team. The user does not know OnArrival exists. From their perspective they are dealing with the travel product they trust. The support agent is the bridge between the user's problem and the platform data that can resolve it.

A good support experience is built on three things - Speed of information retrieval, Clarity of what happened, and Confidence in what can be done. OnArrival's platform makes all three possible by giving the support agent a single unified view of every booking event from search to stay.

Link to the Dashboard Prototype(Low fidelity wireframe)

The Agent's three jobs and what OnArrival makes possible

The first job is acknowledging the complaint and pulling the booking instantly. The agent searches by Booking reference number, phone number, or email. One lookup returns the complete booking record - property name, room type, dates, price confirmed, supplier used, cancellation policy shown at the time of booking, and payment status. No manual searching across systems. No asking the user to repeat information they have already provided.

The second job is understanding what happened. This is where OnArrival's platform provides something the enterprise customer could not build themselves, a complete booking event log. Not just what the user booked but how the booking was processed at every step. What price was displayed at search and what price was confirmed at payment. What the supplier returned at each stage of the booking flow. What error codes were generated if something failed. This log gives the agent the full picture without having to piece it together from multiple sources.

The third job is knowing what can be done. The support dashboard surfaces available actions directly, not a set of manual procedures the agent has to figure out independently. Can this booking be cancelled without penalty given the circumstances? Is a refund available and through which channel? Is there an alternative property available at a comparable price for the same dates? These options are presented based on the booking's current state and the applicable cancellation policy, reducing the time between a complaint and a resolution.

The three most common support scenarios and how the dashboard handles each -

Payment debited but booking not confirmed - the most anxiety-inducing scenario. The user's money has left their account and they have no confirmed booking. The agent pulls the booking record and immediately sees the payment status alongside the supplier response. If payment was captured but the supplier failed to confirm, the dashboard flags this as an unresolved booking event. The agent can see exactly where the process broke down, whether it was a

supplier timeout, a technical failure, or a payment gateway issue. Depending on the cause the agent either initiates a refund immediately or triggers a re-confirmation attempt with the supplier. The user is kept informed throughout without needing to know which system is being interrogated.

The hotel does not acknowledge the booking at check-in - the highest urgency scenario. The user is standing at the hotel reception with a confirmation number the property cannot find. The agent pulls the booking and sees the supplier's confirmation reference, the PNR number issued by the supplier when the booking was confirmed. This is the reference the hotel needs, not the platform's own booking reference. The agent shares this with the user immediately. If the supplier's reference is also unrecognised by the hotel, the agent escalates to OnArrival's platform team who contacts the supplier directly. The user stays on the line with the enterprise customer's agent throughout, the escalation is internal and invisible to them.

Content mismatch at the property - the user arrives and finds the hotel does not match what was shown. Photos were outdated. Amenities were missing. The room type was different from what was booked. The agent pulls the content quality record for the supplier version that was shown to the user - what photos were displayed, what amenities were listed, what room type was described. This becomes the evidence basis for any refund or compensation claim. The mismatch is also logged as a negative signal against the supplier's content quality score on the platform, contributing to future ranking decisions so the same issue is less likely to affect future users.

Escalation to OnArrival

Some issues cannot be resolved by the enterprise customer's support agent alone. Three categories require escalation to OnArrival's platform team:

Supplier disputes - If the hotel or supplier is refusing to honour a booking or process a refund that the cancellation policy entitles the user to, OnArrival intervenes directly with the supplier on the enterprise customer's behalf.

Technical failures - OnArrival's engineering team investigates and resolves if the booking event log shows a platform-level failure rather than a supplier or user error.

Refund processing failures - A refund has been approved but has not reached the user within the expected window.

In every escalation scenario the enterprise customer's agent remains the single point of contact for the end user. The user never speaks to OnArrival directly. The agent handles all communication and updates the user as resolution progresses. The escalation is a change in internal stakeholder, invisible and seamless from the user's perspective.

Partner admin controls

The partner admin dashboard is the enterprise customer's control centre for their hotel booking product. It is used by the product manager, commercial lead, or operations team, not by developers or end users. Its job is to give the partner visibility into how their product is performing and control over how the ranking logic behaves within OnArrival's platform guardrails.

Link to the Dashboard Prototype(Low fidelity wireframe)

What the partner monitors

The dashboard opens with a performance overview showing the metrics that matter most to the partner's business team on a daily and weekly basis.

Headline metrics give an immediate health check, weekly active users on the travel product, total bookings in the period, guest retention rate measured as repeat booking rate, and total support ticket volume. These four numbers tell the partner whether their hotel product is growing, stable, or deteriorating without requiring any drill-down.

Drill-down metrics sit below the headline and explain the story behind the numbers, booking conversion rate showing what percentage of searches result in a booking, average time from search to booking indicating how much friction exists in the flow, and support ticket breakdown by category showing whether tickets are driven by booking failures, content mismatches, or cancellation disputes. The breakdown is the most actionable metric in the dashboard, it tells the partner specifically what is going wrong and points directly to which configuration lever might address it.

Two signals specifically trigger a partner to review their configuration, a sustained drop in booking volume and a rise in support ticket volume. Both are surfaced with trend indicators on the dashboard so the partner can see direction not just current state.

Configuration controls

The partner can adjust four categories of configuration through self-serve controls in the dashboard. All adjustments operate within the guardrails OnArrival has set, the partner sees the permitted range for each control and cannot exceed it without OnArrival involvement.

Ranking weights - the partner sets baseline weights for reliability, content quality, price competitiveness, and margin for their overall product. Weights must sum to 1.0 and each weight has a permitted minimum and maximum range defined by OnArrival. A reliability-first partner sets reliability high. A price-first partner sets price competitiveness high. Both operate within the permitted range, reliability always carries a minimum weight that cannot be reduced below the guardrail level.

Cohort rules - the partner defines different weight configurations for different user segments within their product. A premium cohort might have reliability weighted at 0.45 while a standard cohort has price competitiveness at 0.35. Each cohort rule is a separate configuration that the engine applies when the developer passes the corresponding user segment signal at runtime.

Margin targets - the partner sets their target margin range per booking. This influences the margin signal in the composite score. Changes to margin targets take effect immediately for new searches, existing bookings are not affected.

Supplier preferences - the partner can indicate preferred supplier ordering within the list of approved suppliers on the platform. They can also flag suppliers they want deprioritised based on their own commercial or operational experience. They cannot remove suppliers from the platform entirely, that requires OnArrival involvement, but they can influence ordering within the approved set.

Two changes require OnArrival involvement rather than self-serve configuration - requesting changes to the permitted weight ranges themselves, and adding new suppliers to the approved list. These changes affect the platform's integrity and require review before taking effect.

Guardrails visibility

Every guardrail is visible in the dashboard and explained in plain language. The partner sees the minimum reliability threshold, the permitted weight ranges for each signal, and the approved supplier list. Each guardrail is accompanied by a brief explanation of why it exists, not just a number but the reason behind the number.

For example the minimum reliability threshold is displayed as - "Properties scoring below 70 on reliability will not appear in your results regardless of other configuration. This protects your users from booking failures and your product from support escalations." The guardrail becomes a feature the partner can understand and explain to their own stakeholders rather than an opaque constraint they bump into unexpectedly.

This transparency is intentional. A partner who understands their operating boundaries makes better configuration decisions and raises fewer disputes about why certain properties are not appearing. Guardrail visibility is also how OnArrival builds a collaborative relationship with its enterprise customers.

Supplier dashboard

The supplier dashboard is where OnArrival makes good on its transparency commitment to suppliers. A supplier who loses booking volume without understanding why has no reason to improve, they simply look for other distribution channels. A supplier who can see exactly why their scores have moved and what specific actions will improve them has both the information and the motivation to fix the right things.

The dashboard serves two primary supplier interests - understanding their booking volume and conversion performance on OnArrival's platform, and understanding what is driving their scores so they can act on the right problems.

Link to Dashboard Prototype(Low fidelity wireframe)

Performance visibility

The headline view shows the supplier their booking performance in the current period, total bookings routed through them, total revenue generated, and the ratio of times their properties were shown in search results against the number of times those properties converted to a

booking. This shown-to-booked ratio is the most commercially useful signal for the supplier. A supplier shown frequently but converting poorly has a pricing or content problem. A supplier shown infrequently has a reliability or score problem that is causing the engine to deprioritise them before the user even sees their inventory.

Both situations require different remediation. The dashboard distinguishes between them clearly so the supplier knows which problem they are solving.

Score visibility

Each supplier sees their own four scores - reliability, content quality, price competitiveness, and margin, on a 0 to 100 scale with a label indicating whether they are in the High, Moderate, or Low range. The minimum reliability threshold is shown explicitly, the supplier can see exactly where the floor is and how far above or below it their current score sits.

Below each score the supplier sees the sub-signals that contribute to it. For reliability this means booking success rate, post check-in complaint rate, support ticket rate, API uptime rate, and cancellation dispute rate, each shown individually so the supplier can see precisely which sub-signal is pulling their overall score down. This is specific enough to act on. A supplier whose booking success rate is 94 but whose API uptime is 68 knows the problem is technical, their engineering team needs to fix API reliability, not their operations team.

A score trend line sits alongside each metric showing movement over the last 30 and 90 days. This gives the supplier visibility into whether their remediation actions are working before they cross the threshold, the trend moving upward is the motivating signal during recovery, not just the final threshold crossing.

The notification and improvement process

When a supplier's reliability score drops below a defined warning level, set above the minimum threshold to give the supplier time to act before being eliminated from ranking, OnArrival sends a proactive notification. The notification names the specific sub-signal causing the drop with the precise numbers, not "your reliability score has dropped" but "your API uptime rate has fallen from 89 to 67 over the last 30 days, generating 4 timeout errors per 100 booking attempts. This is pulling your overall reliability score toward the minimum threshold."

The supplier can respond to the notification directly from the dashboard, acknowledging the issue and providing an estimated resolution timeline. This creates a formal record of the improvement commitment and gives OnArrival's supplier management team visibility into which suppliers are actively remediating versus ignoring the signal.

Once the reliability score crosses back above the minimum threshold the supplier is automatically returned to full ranking eligibility. No manual reinstatement required. The improvement is measured by the platform's own observed data, not by the supplier's self-report.

What the supplier does not see

Three categories of information are withheld from the supplier dashboard.

Other suppliers' scores and performance data. Each supplier sees only their own data. Exposing competitor performance would turn the dashboard into a benchmarking tool that reveals commercially sensitive information OnArrival has no right to share.

Partner configuration details. The supplier cannot see which enterprise customers have deprioritised them, what weight configurations partners have applied, or what margin targets partners are working toward. This information belongs to the partner relationship and is not the supplier's to access.

The exact weights applied to each sub-signal in the composite formula. The supplier sees their sub-signal scores, booking success rate, API uptime, and so on, and can see which ones are weak. But they do not see the precise weight each sub-signal carries in the composite calculation. The reason is not that suppliers would fabricate performance, real performance is measured by OnArrival from actual platform events and cannot be gamed. The concern is more subtle, a supplier who knows that booking success rate is weighted at 0.35 and cancellation dispute rate at 0.05 might choose to invest all their remediation effort in booking confirmation and ignore cancellation disputes entirely, since disputes contribute minimally to the score. This leaves a user-affecting problem unresolved. Showing what to fix without showing the exact formula for how each fix is weighted keeps supplier remediation focused on genuine improvement rather than selective score optimisation.

METRICS

Metric	What it measures and why it matters	How it is measured	At launch(proxy used)
Repeat booking rate - North Star	Percentage of users who make a second booking within a defined window. A user who returns has trusted the product enough to commit money to it again. The purest signal that the engine is surfacing the right properties and delivering an experience that matches expectations.	Observed from booking history once sufficient volume exists, typically 6 or more months after launch.	Search to booking conversion rate, a user who converts on first search is more likely to return than one who abandoned. *Proxy data - measurable from day one*
Support ticket rate per completed booking - Primary Metric	Number of support contacts generated per completed booking. A clean booking generates no contact. A failed, disappointing, or confusing booking generates at least one. Captures problems across the entire journey in a single number that moves faster than repeat booking rate. Broken down by ticket type - booking failure, content mismatch, cancellation dispute, to pinpoint where the engine is failing.	Support ticket volume divided by completed booking count in the same period. Segmented by ticket category.	No proxy needed, measurable from the first booking. *Direct data - available immediately*

Booking conversion rate - Primary Metric	Percentage of searches that result in a completed booking. Signals whether the ranked list is producing results the user finds relevant and trustworthy enough to commit to. The most actionable metric for partners reviewing their ranking configuration, a partner whose conversion rate is dropping has a direct signal to investigate.	Completed bookings divided by total search sessions in the same period. Measurable at partner and cohort level.	Also used as the proxy for repeat booking rate at launch. *Direct data - available immediately*
Booking failure rate - Guardrail metric	Percentage of payment attempts that fail to produce a confirmed booking. Users are losing money and trust in real time. The highest severity guardrail, requires immediate investigation when it rises, regardless of what primary metrics show. Segmented by failure type to distinguish technical errors from inventory shortages.	Failed booking confirmations divided by total payment attempts. Monitored in near real time, not reviewed weekly.	*Direct data - measurable from the first live booking.*
Post check-in complaint rate - Guardrail metric	Percentage of completed bookings generating a content mismatch or property dispute after the user has arrived. Captures the gap between what the engine showed and what the user found in reality. Moves slowly, complaints accumulate over days after stays complete, but gives the platform time to act before trust erosion becomes permanent damage to repeat booking rate.	Post check-in complaint tickets divided by completed bookings in the same period. Requires stay date to elapse before it can be calculated.	Supplier content quality scores across the active supplier set, high content quality scores structurally reduce the risk of post check-in disappointment. *Proxy data - observable from scoring engine*

Low reliability booking share - Guardrail metric	Percentage of total bookings flowing through suppliers whose reliability score is close to the minimum threshold. If this share is growing the supply side is quietly deteriorating before it becomes visible in the booking failure rate, the platform is at risk of a supply gap before any individual booking has visibly failed.	Bookings routed through near-threshold suppliers as a proportion of total bookings in the same period.	Supplier reliability scores assessed during integration testing before any live bookings are made. Suppliers near the threshold at onboarding are flagged for closer monitoring from day one. *Proxy data - measurable pre-launch*

How we know the engine is improving

The six metrics above measure whether the product is working for users. A separate but equally important question is whether the engine's ranking decisions are getting better or worse over time, before those improvements or deteriorations show up in user outcomes.

The leading signal for this is the average reliability score of completed bookings. If this number is rising over time it means the engine is routing an increasing proportion of traffic through higher quality suppliers, its decisions are improving. If it is falling the engine is surfacing deteriorating inventory even if booking failure rate has not yet spiked. This signal is observable from within the engine itself without waiting for user outcomes to materialise, which makes it the earliest available indicator of whether the system is self-correcting in the right direction.

The same logic applies to the average content quality score of top-ranked properties, a rising average means the engine is surfacing more accurately represented inventory over time. Together these two internal signals form the feedback loop that tells the platform team whether the engine is getting smarter or drifting.

How we know when the engine has caused a failure

Not every metric deterioration is caused by the engine. A spike in booking failure rate could be a supplier going offline. A rise in support tickets could be a content problem with one property. A drop in conversion rate could be a seasonal demand shift. Distinguishing engine failures from external failures matters because the fix is different in each case.

Three patterns indicate the engine specifically is the cause. First, if booking failure rate spikes across multiple suppliers simultaneously it is more likely an engine routing or fallback logic failure than a single supplier issue. Second, if conversion rate drops while supplier scores and content quality remain stable the ranked list itself is the problem, the engine is surfacing properties that are technically reliable but poorly matched to user intent. Third, if support ticket rate rises uniformly across ticket types rather than spiking in one category it suggests a systemic engine problem rather than a localised supplier or content issue.

These patterns are the difference between knowing something went wrong and knowing what to fix.

Rollout Strategy

Before launch - Prerequisites

1. Supply Level - Minimum two to three suppliers fully integrated and passing the API reliability gate. Each supplier must have content quality scores above a minimum floor across a sufficient number of properties and a signed commercial agreement defining the margin structure. Single supplier go-live is not permitted, one supplier going offline takes the entire platform down.

2. Partner Level - At least one enterprise customer has completed their API integration through the sandbox successfully. Baseline ranking weights are configured. The support team is trained on the support dashboard. End user experience is functional in the

partner's app before the first real user is directed to it.

3. Platform Level - Engine running with conservative defaults - Property deduplication operational; Booking event log capturing data from the first transaction; Supplier notification system live; Fallback logic tested exhaustively, a booking failure in the first week of a new partner's product is a trust event that can end the partnership before it has started.

What gets built when -

Capability	Phase	Reasoning
Configurable weighted scoring	V1	Works without historical data. Conservative defaults at launch. Partners can configure within guardrails from day one.
Property deduplication	V1	High confidence matches auto-merged. Uncertain matches flagged for manual review. Conservative approach at launch, do not automate what cannot yet be calibrated.
Booking routing and fallback logic	V1	Core to the product's reliability promise. Must be live from day one with all three failure classifications - technical error, inventory shortage, and unidentified, handled correctly.
Static runtime signals	V1	User segment and price sensitivity passed by the developer at search time. No user history required. Available from the first API call.
Support, partner, and supplier dashboards	V1	All three stakeholder dashboards must be live at launch. Support agents cannot resolve failures without the booking event log. Partners cannot configure without the admin

		dashboard. Suppliers cannot improve without score visibility.
Behaviour-based runtime signals	V2	Stay history and search behaviour require accumulated user data to be meaningful. Cannot be built until the platform has sufficient observed booking history per user.
Automated content reconciliation at scale	V2	Fully automated content merging across a large supplier set requires enough observed data to calibrate the matching logic accurately. V1 handles uncertain matches manually.
Complex multi-dimensional cohort rules	V2	Exhaustively configurable cohort rules require booking data by cohort to validate whether the rules are producing the intended outcomes. V1 establishes the cohort framework with basic segments.
Dynamic ML ranking	V3	Requires months of observed booking data across properties, suppliers, and user segments before it can make meaningful predictions. Building it at launch wastes engineering capacity on something that cannot function yet.

What we explicitly will not do yet -

1. Go live with a single supplier
2. Allow partners to configure without guardrails
3. Launch to a partner's full user base immediately
4. Route full booking volume through new suppliers from day one

Biggest dependencies -

1. Supplier API integrations passing the reliability gate.
2. Partner sandbox integration completion
3. Fallback logic tested under simulated failure conditions
4. Support team trained on booking lookup dashboard
5. Sufficient booking volume for Score calibration

Biggest risks -

1. Supplier quality worse than expected in production.
2. Partner misconfiguration causing poor user experience
3. First booking failure on a new partner's platform
4. Slow booking volume growth delaying V2 capabilities

V1 establishes the foundation. As observed booking data accumulates, the platform moves toward V2 and V3 capabilities. Each capability is unlocked when the data required to make it work reliably exists, not before. The rollout is complete when repeat booking rate is measurable, supplier reliability scores are updated from observed platform data rather than integration testing proxies, and partner configuration is informed by cohort-level outcome data rather than best-guess defaults. At that point the engine is no longer being operated on assumptions.

Edge Cases and Failure Modes

The Inventory Intelligence Engine is designed to handle ambiguity and failure deliberately, not just the happy path. Below are nine scenarios where the system could break or behave unexpectedly, and how the design responds to each.

The handling principle that runs through every case

Every failure mode is handled at the earliest possible point in the engine's flow, before the user sees a result, before a booking is attempted, or before a bad signal compounds into a worse outcome. Where the engine cannot prevent a failure, it contains it and feeds it back into platform intelligence so the same failure is less likely to recur.

Edge case	What goes wrong	Why it matters	How the design handles it
Best-priced supplier fails often after payment	A supplier offers the lowest price and ranks well on price competitiveness, but their booking confirmation rate is poor. The user selects them, payment is attempted, and the booking fails.	The user has already committed emotionally and financially. A failure at this stage is the highest severity trust erosion event, payment has been initiated and nothing has been confirmed.	Booking success rate is the highest weighted sub-signal in the reliability score (0.35). A supplier with frequent post-payment failures will see their reliability score fall quickly. If it drops below the platform floor, they are eliminated from ranking entirely before any user sees them. The fallback logic classifies the failure, confirms no charge was made, and surfaces alternatives ranked by the same engine logic.
Supplier has great price but terrible cancellations	A supplier prices aggressively and scores high on price competitiveness. But they generate frequent cancellation disputes, guests find the cancellation terms are not honoured or are more restrictive than shown.	The user books based on a competitive price and later cannot cancel as expected. This creates a support escalation and often a refund dispute, costly for the enterprise customer and damaging to end user trust.	Cancellation dispute rate is a sub-signal within the reliability score. A pattern of disputes pulls down the composite score regardless of how competitive the price is. If disputes are frequent enough, the reliability score falls below the platform floor and the supplier is eliminated. Price competitiveness cannot compensate for reliability failures, the scoring structure prevents it.

Premium discounts distort ranking logic	A supplier offers a steep discount that boosts their price competitiveness score, pushing them up the ranked list. But the discount applies only to a non-refundable rate, expires before check-in, or is being used to drive volume despite poor service quality.	The user sees a highly ranked, apparently cheap option that is not as good a deal as it appears. If they book without understanding the terms, the post-stay or cancellation experience may disappoint, a trust erosion event that the ranking logic contributed to.	Two engine-side defences exist. First, the reliability floor eliminates suppliers who are discounting to drive volume despite poor performance, reliability catches up with them regardless of price. Second, the price competitiveness score rewards consistent competitive pricing, not anomalous one-off discounts. The experience design adds a third layer, cancellation terms are surfaced prominently early on the details page before the user commits, so a non-refundable discount is visible before booking, not after.
Hotel content merged incorrectly	The deduplication process incorrectly merges two different properties into one listing. The user sees content from Property A but the booking is routed through a supplier whose version represents Property B, a different physical hotel.	The user arrives at a hotel that does not match what they saw. This is the most severe content failure the platform can produce, a physical mismatch at check-in that no amount of support can immediately resolve.	Deduplication uses a multi-signal confidence score - lat-long coordinates, property name similarity, standardised property IDs, and confirming signals like phone number. Only high confidence matches are auto-merged. Moderate confidence matches are held for manual review. The content quality score includes a cross-supplier consistency check, if two merged listings show significantly different amenities or descriptions, that discrepancy is flagged for re-review. Incorrect merges that do reach users are logged as severe content mismatch events and feed back into the deduplication logic to prevent recurrence.

Partner overrides reduce performance	An enterprise customer configures ranking weights to heavily prioritise margin, perhaps due to a commercial deal with a specific supplier, surfacing options that are commercially beneficial to them but not the best options for their users.	The user consistently sees results that feel slightly off, not terrible enough to complain, but not good enough to build trust. Over time conversion drops and repeat booking rate suffers. The partner does not immediately connect their configuration choices to the performance deterioration.	Platform guardrails cap how heavily any single signal can be weighted. A partner cannot configure margin to dominate ranking to the point where reliability or content quality are meaningfully sacrificed. The partner dashboard surfaces performance metrics with trend indicators alongside their configuration settings, a drop in conversion rate or rise in support ticket volume is visible in the same view as their ranking weights, creating a direct connection between configuration choices and product performance.
Recommended tag loses user credibility	The top-ranked property carries the Recommended tag but users consistently do not select it, scrolling past it to choose lower-ranked options. The tag starts to feel like a commercial label rather than a genuine recommendation.	If users do not trust the engine's top recommendation, they lose the primary trust signal the platform offers. Over time this erodes confidence in the product and reduces the value of the ranking system entirely.	The metrics section defines this as a detectable failure pattern, conversion rate drops while supplier scores remain stable, indicating the ranked list is the problem. This triggers investigation into whether ranking weights are correctly calibrated for that partner's user cohort. The experience design also limits where the tag appears, only on the top result in the default sort, disappearing when the user changes sort order. This keeps the tag honest. It belongs to the engine's genuine recommendation, not to a result the user has manually repositioned.

All supplier versions eliminated by reliability floor	For a given property, every available supplier version falls below the minimum reliability threshold. The property has no eligible supplier versions and cannot appear in results.	If this happens across multiple properties in a destination, the user sees a severely limited result set without understanding why. This feels like a platform failure even though the guardrail is working correctly.	The engine does not surface the property, showing an unreliable option is worse than showing fewer options. The search results page handles thin result sets honestly, suggesting nearby dates or adjusted filters rather than leaving the user on an empty screen. OnArrival's supplier management team is notified when multiple supplier versions for a property are simultaneously near or below the threshold, triggering outreach before the floor is breached entirely.
New supplier with no reliability history	A newly integrated supplier has no observed booking data on OnArrival's platform. They receive a conservative neutral score by default. But if their actual reliability is significantly worse than the neutral score implies, they may be ranked higher than they deserve in early bookings.	Early bookings with a new supplier carry inherently more risk. If the supplier turns out to be unreliable, the first users to book through them absorb the cost of that discovery, a trust erosion event that could have been avoided with more caution.	New suppliers start at a conservative neutral score and receive a limited initial traffic allocation, they do not receive full booking volume from day one. Early booking outcomes update their scores quickly, so reliability problems are detected and reflected within the first few hundred bookings. New suppliers are also flagged for closer monitoring from the moment they go live, so OnArrival's supplier management team can act early if signals deteriorate.

Price change between search and booking	The supplier's price changes between the moment the user sees it on the search results page and the moment they proceed to book. The user commits based on one price and encounters a different one.	A price surprise at any point in the booking journey is a trust erosion event. The later it appears, the more damaging it is, a surprise at checkout feels like a bait-and-switch even if the platform had no control over the supplier's pricing.	The engine re-validates the supplier's price in the background when the user opens the hotel details page. If the price has changed, a clear non-alarming banner appears immediately at the top of the page, before the user sees room selection or taps Book Now. The price shown throughout the booking flow reflects the re-validated price, not the original search price. A price change is also logged as a supplier pricing volatility signal, contributing to that supplier's ranking assessment over time.

How the engine distinguishes its own failures from external ones

Not every performance deterioration is caused by the engine. Identifying the source of a failure matters because the fix is different in each case.

A spike in a single metric category - booking failures, content mismatches, or cancellation disputes, may point to a specific supplier or content issue, but it does not rule out the engine. If the engine's routing logic is directing traffic disproportionately through one supplier or one property, the failure will still show up as a category spike even though the root cause is the engine. Every category spike warrants investigation into both the supplier and the engine's routing behaviour before drawing a conclusion.

A uniform rise across all ticket types simultaneously is a stronger signal that the engine itself is the problem. Different suppliers tend to cause different types of failures, a supplier with poor booking confirmation generates booking failure tickets, while a supplier with inaccurate content generates mismatch tickets. When every category rises together, no single supplier or content issue is a plausible explanation. Something is affecting all bookings regardless of supplier, which points to the engine's core ranking or routing logic.

A drop in conversion rate while supplier scores and content quality remain stable is the third pattern. Here the supplier side looks healthy but users are not converting. The ranked list itself is the most likely cause, the engine is surfacing technically reliable options that are poorly matched to user intent.

Together these three patterns are the difference between knowing something went wrong and knowing what to fix.