

CUBES Unified Approval Workflow

How the existing Van Sales -> Enterprise API -> Enterprise App discount approval works, and how to design a reusable Resto Service endpoint for discount, item void, bill cancel, and refund.

Central fact The readable value DISC-1-1784122302191 is not used to join product lines or poll status. SQL Server creates an integer header Id. Detail rows store that integer in hdr_key, Enterprise App approves that same integer, and Van Sales polls that integer.

Document scope	Value
Current systems	CUBES Van Sales, CUBES Enterprise API, CUBES Enterprise App
Proposed system	Resto Service Online as a validated approval gateway
Actions	Discount, item void, bill cancel, refund
Status	Architecture explanation and draft recommendation; no code changes
Prepared	2026-08-24

Quick answer

- One header can contain many product/detail rows.
- The header-detail relationship is header.Id = detail.hdr_key.
- The push notification goes to the Enterprise App approver, not back to Van Sales.
- Van Sales learns the decision by polling GetApprovalStatus every five seconds.
- The same approval tables can be reused, but they need action/source/reason fields and stable bill-line identifiers.

Contents

Section	Topic
1	The three identifiers and why they are different
2	Current end-to-end discount workflow
3	How multiple product lines are stored and retrieved
4	How Enterprise App receives and decides the request
5	How Van Sales learns Approved or Disapproved
6	Database connection boundaries
7	What should be reused for four action types
8	Recommended generic API and payload
9	Database transaction and notification sequence
10	How each action is executed after approval
11	Resto Service deployment approaches
12	Required changes by project
13	Security, reliability, tests, and rollout
14	Source evidence

Recommended direction

Resto Service should validate the originating bill and expose a generic submission/status API, then forward the request to Enterprise API. Enterprise API should remain the central owner of Dashboard approval tables, Enterprise App integration, device tokens, and push delivery.

1. The three identifiers and why they are different

Identifier	Example	Purpose	Where used
hdr_id / ClientRequestId	DISC-1-1784122302191	Readable business request number and retry/idempotency key	Shown in Enterprise App; sent by Van Sales
Header Id	527	SQL identity and canonical approval ID	Returned by API; used for approve, status, and details
Detail hdr_key	527	Logical parent link from every detail row	Used to retrieve all product lines for header 527

How DISC-1-1784122302191 is created

```
String terminalId = settings.getString("TerminalID", "T1");
String hdrId = "DISC-" + terminalId + "-" + System.currentTimeMillis();
```

For terminal 1, the result is DISC-1-{current Unix time in milliseconds}. It is readable and usually unique, but the current database does not enforce uniqueness. The new design should treat it as ClientRequestId and add a unique index.

Do not confuse keys

Van Sales does not later ask for status using DISC-1-1784122302191. It stores the API response Id, for example 527, as pendingApprovalId and, for a held bill, as discount_approval_id in its local SQLite bill header.

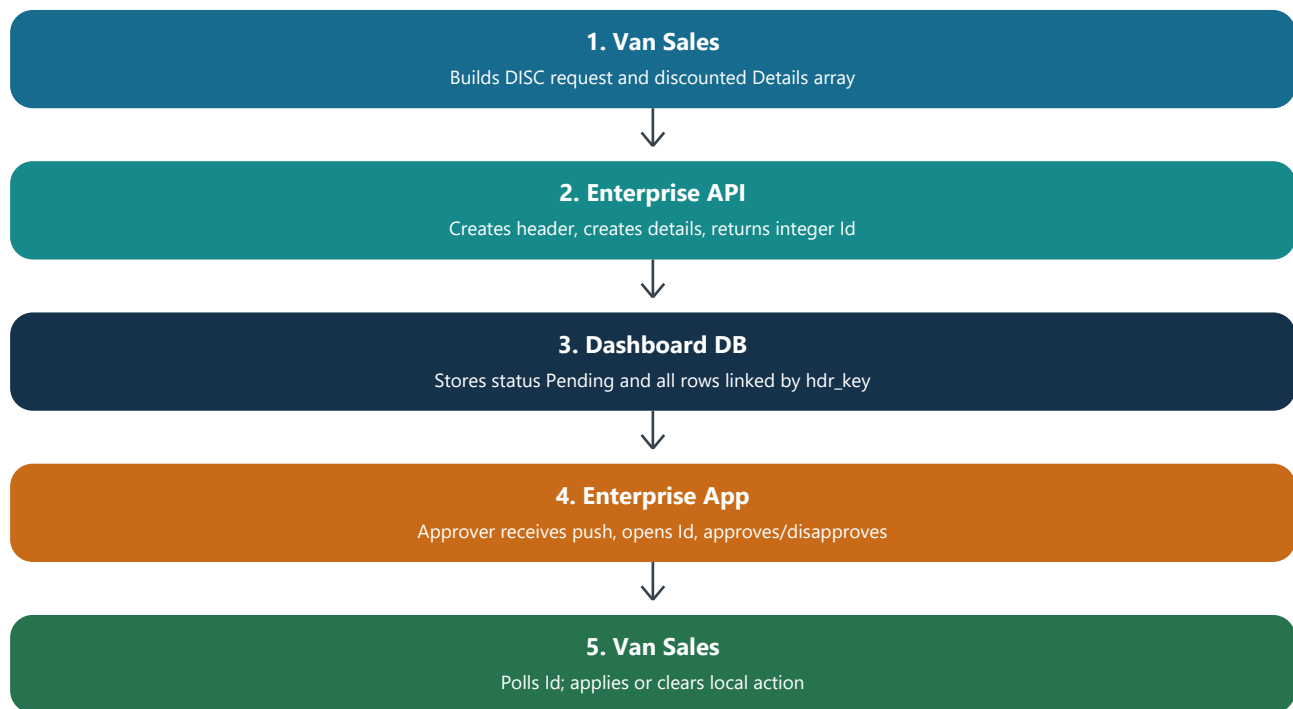
Example relationship

```
Dashboard header table
  Id = 527
  hdr_id = DISC-1-1784122302191
  approval_type = 3
  status = Pending

Dashboard detail table
  row 9001: hdr_key = 527, product_key = 100, disc_amt = 20
  row 9002: hdr_key = 527, product_key = 200, disc_amt = 5

Logical join
  header.Id (527) = detail.hdr_key (527)
```

2. Current end-to-end discount workflow



Detailed request sequence

- Van Sales reads DiscountApproverEmail from settings.
- Van Sales calls GetCompanyPrefix with that email. The API looks up mst_user joined to mst_company and returns a prefix such as ABC.
- Van Sales builds a Details array containing each non-void line where DiscountAmount > 0.
- Van Sales posts the header plus Details to /Approval/CreateDiscountApproval.
- Enterprise API connects to the central Dashboard DB using its DBConnection setting.
- Enterprise API inserts the header with status Pending and approval_type 3, then obtains SCOPE_IDENTITY().
- It inserts every Details item with hdr_key equal to that identity.
- It sends an Expo push to tokens registered for approver_email.
- It returns the integer identity to Van Sales as data.id.

3. How multiple product lines are stored and retrieved

Yes, one header can contain many detail rows. The request model contains List Details. The API loops over that list and performs one detail insert per item, always using the same headerId.

Header example: `trn_approval_hdr_ABC`

Id	hdr_id	approval_type	total_amt	status	approver_email
527	DISC-1-1784122302191	3	225.00	Pending	manager@example.com

Details example: `trn_approval_dtl_ABC`

Id	hdr_key	product_key	product_name	qty	price	disc_amt
9001	527	100	Product A 70.00	2	100	20
9002	527	200	Product B 40.00	1	50	5

Retrieval

```
SELECT *
FROM dbo.trn_approval_hdr_ABC
WHERE Id = 527;

SELECT *
FROM dbo.trn_approval_dtl_ABC
WHERE hdr_key = 527;
```

The current schema has an index on detail.hdr_key but no physical foreign-key constraint. A future migration can add an FK only if the per-prefix table generation and historical data quality allow it.

Current detail scope

Van Sales sends only discounted lines. Non-discounted invoice products are omitted. For a bill-level discount with no discounted product rows, Van Sales sends one synthetic line with product_key 0 and product_name Bill Discount|||0.00.

4. How Enterprise App receives and decides the request

Why approval_type 3 means discount

- CreateDiscountApprovalAsync explicitly writes the literal value 3 into approval_type.
- Enterprise API exposes customer_name only when approval_type = 3.
- Enterprise App has a special approval_type == 3 branch for discount display.
- The detail screen also recognizes hdr_id containing DISC.

Current type	Meaning	Evidence/behavior
1	LPO	Verifier then approver workflow
2 / existing other	Stock/request	Request approval workflow
3	Discount	Hardcoded by discount creation and rendered as discount

Push delivery

Enterprise API queries UserDeviceTokens by approver email, then calls the Expo Push API. The notification data contains notificationType = DISCOUNT_APPROVAL and approvalId = 527. When tapped, Enterprise App navigates directly to the approval detail screen.

```
{
  "to": "ExponentPushToken[...]",
  "title": "Discount Approval Request",
  "body": "New discount approval ...",
  "data": {
    "notificationType": "DISCOUNT_APPROVAL",
    "approvalId": "527"
  }
}
```

Decision

Enterprise App calls /Approval/approve/527 or /Approval/disapprove/527 with companyPrefix. The API updates the same header Id. Approved sets approved_by, approved_by_email, approved_datetime, and status = Approved. Disapproved sets the equivalent audit fields and status = Disapproved.

5. How Van Sales learns Approved or Disapproved

No return push

The current implementation sends a push only to the Enterprise App approver. Van Sales is not registered as a notification target for approval results.

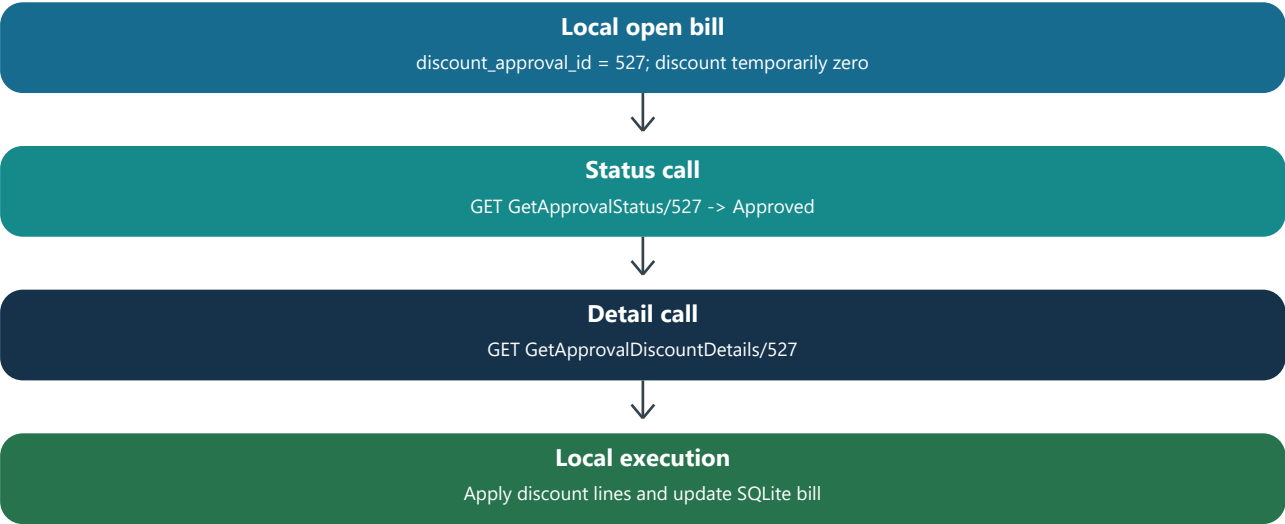
Active discount screen

Van Sales stores data.id as pendingApprovalId. A Handler runs every 5,000 ms and calls GetApprovalStatus/{approvalId}. The status query reads the central Dashboard header row by Id.

Returned status	Van Sales behavior
Pending	Keep polling and display pending state
Approved	Stop polling, clear approval link, mark discountApproved, call SaveDiscount()
Disapproved	Stop polling, clear approval link, call ClearDiscount()

Held open bill

When Request Approval and Hold is used, Van Sales saves the open bill without discount and stores approval Id in local trn_bill_header.discount_approval_id. The Open Bills screen checks status. Once approved, it calls GetApprovalDiscountDetails, retrieves detail rows by hdr_key, applies disc_amt locally, recalculates VAT and bill totals, and persists the updated bill.



6. Database connection boundaries

Component	Connection	Data responsibility
Van Sales	No direct Dashboard SQL connection	Local SQLite bill plus HTTP API calls
Enterprise API	ConnectionStrings:DBConnection	Central Dashboard users, tokens, approval headers/details
Enterprise App	No direct SQL connection	Calls Enterprise API with bearer token
Resto Service	DBPath / RESTO_DB_PATH	Operational Resto bills, products, stock, reasons, payments

The existing Resto Service DBPath is not automatically the Dashboard database. Writing `trn_approval` tables through DBPath would place them in the operational outlet database. Enterprise App would not see those rows unless Enterprise API uses the exact same database.

Connection rule

If Resto Service writes Dashboard tables directly, configure a separate DashboardDBPath. Do not reuse DBPath unless deployment evidence proves both names point to the same database.

Preferred boundary

Resto Service reads and validates operational bill data using DBPath, then forwards a normalized approval request over HTTP to Enterprise API. Enterprise API alone writes Dashboard approval tables and sends the Enterprise App push. This prevents duplicated SQL logic and Dashboard credentials in outlet services.

7. What should be reused for four action types

The existing per-company `trn_approval_hdr_` and `trn_approval_dtl_` tables can be reused for compatibility. Do not overload `disc_amt` or `qty` to mean different things without explicit action fields.

approval_type	Action	Header meaning	Detail meaning
3	Discount	Original/requested total and bill reference	Line discount amount
4	Item void	Bill reference and void reason	Stable bill-line ID and requested void qty
5	Bill cancel	Bill reference, total, cancel reason	Optional full bill snapshot
6	Refund	Original bill and requested refund total	Stable line ID, refund qty and amount

Recommended header additions

```

source_system      nvarchar(50)
source_bill_key    int
source_bill_no     nvarchar(100)
source_terminal_id nvarchar(50)
source_location_key int
client_request_id  nvarchar(100) -- unique, e.g. VOID-1-...
reason_key         int
reason_text        nvarchar(500)
original_amount    decimal(18,2)
requested_amount   decimal(18,2)
request_payload_json nvarchar(max) -- optional audit snapshot

```

Recommended detail additions

```

source_line_id      nvarchar(100) -- required for line actions
action_type         nvarchar(30)
original_qty        decimal(18,3)
requested_qty       decimal(18,3)
original_amount     decimal(18,2)
requested_amount    decimal(18,2)
refund_amount       decimal(18,2)
reason_key          int
reason_text         nvarchar(500)
original_values_json nvarchar(max)
requested_values_json nvarchar(max)

```

Why source_line_id matters

Matching only `product_key` is unsafe because the same product can appear on multiple bill rows. Every void, refund, and discount detail should carry a stable local bill-line identifier.

8. Recommended generic API and payload

Use one generic submission endpoint for all request types, plus common read and decision endpoints. A single contract is useful; one method that mixes create/get/approve/status operations is not.

Endpoint	Responsibility
POST /Service.svc/CreateApprovalRequest	Validate and submit Discount/Void/Cancel/Refund
POST /Service.svc/GetApprovalStatus	Return status for one approval Id
POST /Service.svc/GetApprovalRequest	Return header and all details
POST /Service.svc/DecideApprovalRequest	Approve or disapprove with authorization and audit

Generic submission example

```
{
  "AuthKey": "encrypted-auth-key",
  "CompanyPrefix": "ABC",
  "RequestType": "VOID",
  "ClientRequestId": "VOID-1-1784122302192",
  "SourceSystem": "VAN_SALES",
  "SourceBillKey": 5001,
  "SourceBillNo": "INV-10025",
  "TerminalId": "1",
  "LocationKey": 1,
  "RequestedBy": "salesperson",
  "ApproverEmail": "manager@example.com",
  "OriginalAmount": 250.00,
  "RequestedAmount": 100.00,
  "ReasonKey": 7,
  "ReasonText": "Item entered incorrectly",
  "Details": [
    {
      "SourceLineId": "5001-3",
      "ProductKey": 100,
      "ProductName": "Product A",
      "OriginalQty": 3,
      "RequestedQty": 1,
      "Price": 100.00,
      "DiscountAmount": 0,
      "RefundAmount": 0
    }
  ]
}
```

Generic submission response

```
{
  "Status": {
    "ResultCode": "1",
    "ResultStatus": "Success",
    "Error": ""
  },
  "ApprovalId": 527,
  "RequestNumber": "VOID-1-1784122302192",
  "ApprovalStatus": "Pending",
  "NotificationSent": true
}
```

Status request and response

```
Request
{
  "AuthKey": "encrypted-auth-key",
  "CompanyPrefix": "ABC",
  "ApprovalId": 527
}

Response
{
  "ApprovalId": 527,
  "RequestType": "VOID",
  "ApprovalStatus": "Approved",
  "DecisionDateTime": "2026-08-24T10:30:00Z"
}
```

Decision request

```
{
  "AuthKey": "encrypted-auth-key",
  "CompanyPrefix": "ABC",
  "ApprovalId": 527,
  "Decision": "Approved",
  "DecisionBy": "manager@example.com",
  "Remarks": "Approved"
}
```

Idempotency

CreateApprovalRequest must first search by CompanyPrefix + ClientRequestId. If the same request is retried, return the existing ApprovalId rather than create a duplicate.

9. Database transaction and notification sequence

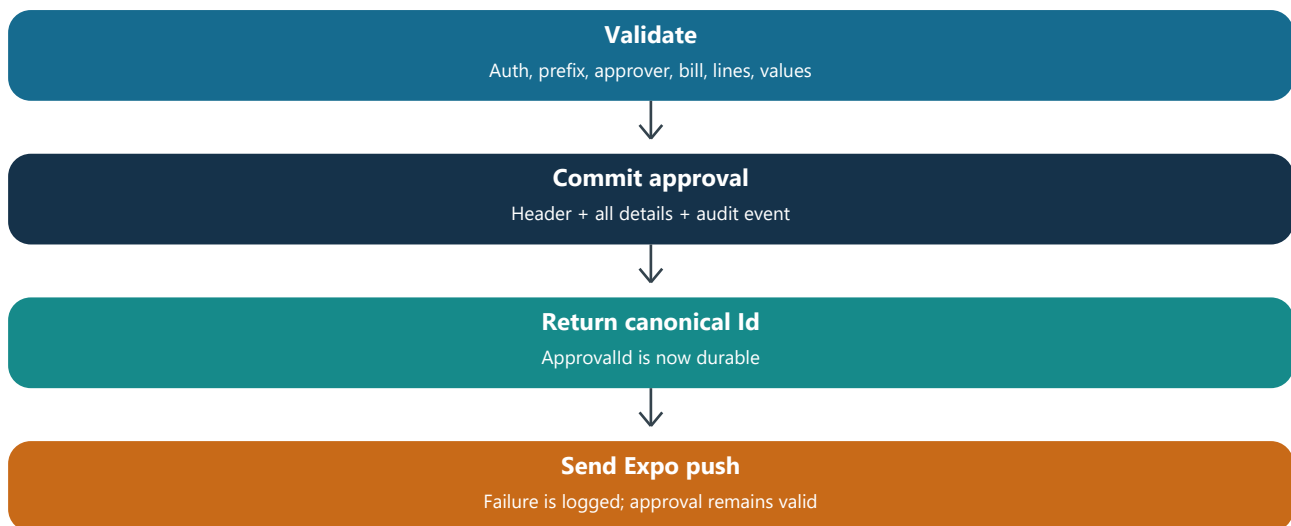
The current Enterprise API inserts header and detail rows separately. The unified implementation should use one stored procedure, one transaction, and a table-valued parameter or validated JSON detail input.

```
BEGIN TRY
  BEGIN TRANSACTION;

  -- 1. Reject duplicate ClientRequestId or return existing Id.
  -- 2. Insert header and capture SCOPE_IDENTITY().
  -- 3. Insert all details with hdr_key = new header Id.
  -- 4. Insert an audit/event record: Submitted.

  COMMIT TRANSACTION;
END TRY
BEGIN CATCH
  IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
  THROW;
END CATCH;
```

Notification must occur after commit



Notification payload should be generic: notificationType = APPROVAL_REQUEST, requestType = DISCOUNT/VOID/CANCEL/REFUND, and approvalId. Enterprise App routes all four types to the same approval detail screen, which renders type-specific sections.

10. How each action is executed after approval

Approval data is an immutable request snapshot and decision record. The approval service should not directly modify POS stock, accounting, or bill tables. After Van Sales receives Approved, Van Sales executes the local domain operation using its existing business rules.

Type	Before approval	After Approved	After Disapproved
Discount	Keep proposed discount pending; optionally hold bill at original amount	Apply line disc_amt and recalculate totals/VAT	Clear proposed discount
Item void	Keep line active and store requested void qty	Run local void logic for source_line_id	Keep line unchanged
Bill cancel	Keep bill active/held	Run local cancel workflow and reason	Keep bill active
Refund	Do not create final refund posting	Run local refund transaction workflow	Do not post refund

Van Sales needs a generic local pending table

```

pending_approval_action
  local_id
  approval_id
  client_request_id
  request_type
  source_bill_key
  source_line_id
  request_payload_json
  status          -- Pending/Approved/Disapproved/Applied/Failed
  created_at
  completed_at
  last_error

```

This replaces the discount-only assumption that one bill header column is sufficient. It also allows retries if an approval is granted while the device is offline or the local action fails.

11. Resto Service deployment approaches

Approach A - validated gateway to Enterprise API (recommended)

- Van Sales calls the new Resto Service CreateApprovalRequest endpoint.
- Resto Service authenticates AuthKey and validates the source bill/lines through DBPath.
- Resto Service forwards a normalized request to Enterprise API over HTTP.
- Enterprise API writes Dashboard tables, sends push, and returns ApprovalId.
- Resto status/detail endpoints proxy Enterprise API, keeping Van Sales integration simple.

Benefits: one central approval owner, existing Enterprise App compatibility, no Dashboard credentials at every outlet, and one notification implementation.

Approach B - Resto Service writes Dashboard DB directly

- Add a separate DashboardDBPath connection setting.
- Resto DBPath remains the operational bill database.
- Duplicate header/detail/notification logic inside Resto Service.

Trade-off: fewer HTTP hops, but duplicated ownership, greater credential exposure, and higher risk that Enterprise API and Resto Service behave differently.

Approach C - save approvals only in local Resto DB

Not recommended for the current architecture. Enterprise App reads the central Dashboard approval tables; it would not see outlet-local approvals without new multi-outlet discovery and routing.

Recommendation

Choose Approach A. It satisfies the requirement for a Resto Service endpoint while preserving Enterprise API as the central approval system and Enterprise App integration point.

12. Required changes by project

Project	Changes
Resto Service Online	Add WCF request/response contracts; Create/Get/Status proxy methods; AuthKey validation; operational bill validation; Enterprise API client; structured logging and timeouts.
Enterprise API	Add generic create/status/detail/decision contract; extend schema; transaction-based stored procedure; type enum/constants; idempotency; generic push payload.
Enterprise App	Render approval types 3-6; show relevant details/reason/source bill; generic notification routing; authorize approve/disapprove; prevent editing immutable source facts.
Van Sales	Call Resto endpoint; generate ClientRequestId; store generic pending action; poll ApprovalId; execute type-specific local action; retry and recovery UI.

Suggested server-side type definition

```
public enum ApprovalType
{
    Lpo = 1,
    StockRequest = 2,
    Discount = 3,
    ItemVoid = 4,
    BillCancel = 5,
    Refund = 6
}
```

The enum value must be shared in documentation and tests across services and clients. Enterprise App can use a TypeScript enum with the same numeric values.

13. Security, reliability, tests, and rollout

Security

- Require AuthKey or a signed service credential on every Resto submission/status endpoint.
- Do not accept an arbitrary companyPrefix and concatenate it into SQL. Resolve/whitelist the prefix after authentication.
- Verify that approver_email belongs to the company and has approval permission.
- Verify source bill, source line, quantity, amount, and reason from the operational DB rather than trusting the client payload.
- Do not include DB credentials, auth keys, or personal device tokens in logs.

Reliability

- Use a unique ClientRequestId to make retries safe.
- Commit header and all details atomically before notification.
- Treat notification failure as retryable and independent from approval creation.
- Use compare-and-set decision logic so a request cannot be approved and disapproved twice.
- Keep an approval event/audit history instead of only the latest status.
- Let Van Sales mark local action Applied only after its domain transaction succeeds.

Essential test cases

Area	Test
Header/detail	Three submitted lines produce one header and three details with the same hdr_key
Idempotency	Retrying ClientRequestId returns the original ApprovalId without duplicate rows
Authorization	Wrong approver/company cannot list or decide the request
Atomicity	Detail insertion failure rolls back the header
Notification	Push failure leaves approval Pending and records retryable delivery status
Decision	Only Pending can become Approved/Disapproved; second decision is rejected
Van execution	Each type applies only its source bill/line and remains retryable after local failure

14. Source evidence

Evidence	File	Lines
Van request number generation	D:\ANDROID_APPS\CUBES_VAN_SALES\CUBES_VAN_SALES\app\src\main\java\com\example\abdul\cubes_resto\Discount.java	1423-1426
Van request payload and Details array	D:\ANDROID_APPS\CUBES_VAN_SALES\CUBES_VAN_SALES\app\src\main\java\com\example\abdul\cubes_resto\Discount.java	1428-1464
Van stores API data.id	D:\ANDROID_APPS\CUBES_VAN_SALES\CUBES_VAN_SALES\app\src\main\java\com\example\abdul\cubes_resto\Discount.java	1480-1490
Van polling and status handling	D:\ANDROID_APPS\CUBES_VAN_SALES\CUBES_VAN_SALES\app\src\main\java\com\example\abdul\cubes_resto\Discount.java	1846-1909
Van held bill applies approved details	D:\ANDROID_APPS\CUBES_VAN_SALES\CUBES_VAN_SALES\app\src\main\java\com\example\abdul\cubes_resto\OpenBills.java	1083-1208
Enterprise discount controller	D:\ANDROID_APPS\CUBES_ENTERPRISE_API\CUBES_ENTERPRISE_API\Controllers\ApprovalController.cs	171-260
Enterprise header/detail insertion	D:\ANDROID_APPS\CUBES_ENTERPRISE_API\CUBES_ENTERPRISE_API\Repository\Service\ApprovalService.cs	391-456
Enterprise status query	D:\ANDROID_APPS\CUBES_ENTERPRISE_API\CUBES_ENTERPRISE_API\Repository\Service\ApprovalService.cs	479-500
Dashboard approval table schema	D:\ANDROID_APPS\CUBES_ENTERPRISE_API\CUBES_ENTERPRISE_API\Scripts\setup_dashboard_db.sql	125-200
Enterprise push tokens and Expo delivery	D:\ANDROID_APPS\CUBES_ENTERPRISE_API\CUBES_ENTERPRISE_API\Repository\Service\PushNotificationService.cs	29-115
Enterprise App approval API calls	D:\ANDROID_APPS\CUBES_ENTERPRISE_APP\CUBES_ENTERPRISE_APP\hooks\Zustand\useApproval.ts	30-270
Enterprise App discount UI calculations	D:\ANDROID_APPS\CUBES_ENTERPRISE_APP\CUBES_ENTERPRISE_APP\app\tabs\approvals\approvalDetails.tsx	31-162, 331-445
Resto WCF service contracts	D:\TOOLS\Resto_Service_Online\Resto_Service_Online\App_Code\IService.cs	1 onward
Resto DB access	D:\TOOLS\Resto_Service_Online\Resto_Service_Online\App_Code\DBHandler.cs	188 onward
Resto existing Expo push method	D:\TOOLS\Resto_Service_Online\Resto_Service_Online\App_Code\Service.cs	6300-6402

Design approval gate

This document explains the verified current behavior and a recommended architecture. Implementation should begin only after confirming whether Resto Service will act as the gateway to Enterprise API or write the Dashboard database directly.