**Kuickpay apis**

**Logs api**

**Public api to see vocher details like it is approved or no or which things included in this vocher. Qr code scan system.**

**Group and permission api**

**Customer side web page handling like taking booking then delivering order of booking follow up that client.**

**COMMISSION OF BRACH FROM AGENT AND AREA AGENT AND CUSTOMERS\_Branch commissions from umrah packages,hotels and tickets see everything of commissions**

**Hotel outside sourcing (like where from i purchase this hotel hotel name,room no,room type,room price, room quantity etc.)**

**All agent can see movement of there pex (for example in makkah,madina or jeddah)**

**Get all prices apis some updates**

**🧾 LEDGER AUTO CREATION SYSTEM – FINAL STRUCTURE**

**⚙️ Auto Create Rule**

Whenever a **booking** is marked as “Paid” →  
System will **auto-create a ledger transaction** between the respective parties,  
based on booking ownership and payment flow.

**🔹 LEDGER ENTRY FORMAT (for every auto/post/manual record)**

| **Field** | **Description** |
| --- | --- |
| **creation\_datetime** | Auto set (timezone aware) |
| **booking\_no** | Booking reference (auto from booking table) |
| **service\_type** | ticket / umrah / hotel / transport / package / payment / refund |
| **narration** | Text summary (e.g., “Advance payment for Umrah Booking #SK1234”) |
| **transaction\_type** | debit or credit |
| **seller\_organization\_id** | Organization who owns the inventory |
| **inventory\_owner\_organization\_id** | Owner org of that inventory item (auto detect from item) |
| **area\_agent\_id** | if booking linked with area agent |
| **agency\_id** | if created by an agent |
| **branch\_id** | if created under branch |
| **payment\_ids** | list of all linked payment records |
| **group\_ticket\_: total number** | if multiple tickets are in group booking |
| **umrah\_visa : total number** | if Umrah visa included |
| **hotel\_nights : total number** | all hotels involved with this booking |
| **final\_balance :** | auto-calc from (total paid - total due) |
| **internal\_notes ids** | array of internal notes text only (example below) |

**Internal Notes Example (text format):**

[2025-10-17 11:24] Payment received via Bank Alfalah.

[2025-10-17 11:25] Commission auto-posted to agent.

[2025-10-17 11:26] Linked with Umrah package #U245.

**🔹 GET ENDPOINTS (5 LEDGER LEVELS)**

1️⃣ **Organization Ledger (with all its branches & linked orgs)**  
GET /api/ledger/organization/<organization\_id>/  
→ shows all transactions related to that organization and its branches.

2️⃣ **Branch Ledger**  
GET /api/ledger/branch/<branch\_id>/  
→ shows all transactions between branch ↔ organization / agents.

3️⃣ **Agency Ledger**  
GET /api/ledger/agency/<agency\_id>/  
→ shows all transactions between agent ↔ branch / organization.

4️⃣ **Area Agency Ledger**  
GET /api/ledger/area-agency/<area\_agency\_id>/  
→ shows all transactions between area agency ↔ organization.

5️⃣ **Organization-to-Organization Ledger**  
GET /api/ledger/org-to-org/<org1\_id>/<org2\_id>/  
→ shows receivable/payable summary and full transaction history between two companies.

**🧮 Auto Posting Logic**

| **Condition** | **Debit** | **Credit** | **Narration Example** |
| --- | --- | --- | --- |
| Agent booked inventory owned by Saer.pk | Agent | Saer.pk | “Agent payment for ticket booking” |
| Branch booked inventory owned by Saer.pk | Branch | Saer.pk | “Branch booking settlement” |
| Area Agent got commission | Saer.pk | Area Agent | “Area commission for booking” |
| Organization A using inventory of Organization B | Org A | Org B | “Inventory share settlement” |
| Refund issued | Saer.pk | Agent / Customer | “Refund for cancelled booking” |

**🏨 HOTEL & INVENTORY LINK RULE**

Each booking item (hotel, ticket, transport, etc.) must have:

"inventory\_owner\_organization\_id": 12

📘 **How it works:**

* System auto-detects who owns that item.
* When booking is paid, **ledger auto creates between buyer org and owner org**.
* If multiple hotels belong to different owners, separate ledger entries auto-created for each.

🧠 **Example:**

Booking has 2 hotels —  
Hotel A (owned by Saer.pk) and Hotel B (owned by Al Madina Group).

When booking is paid →  
One ledger entry creates Saer.pk ↔ Agent  
Another ledger entry creates Al Madina ↔ Saer.pk (as reseller flow).

**🚫 What NOT to do (important rules)**

❌ Don’t allow **manual change** in:

* inventory\_owner\_organization\_id
* booking\_no
* linked payment IDs  
  Because this breaks the auto-balance logic and audit trail.

❌ Don’t allow **double posting**:

* Once booking marked “Paid”, one ledger entry per service type.
* Any later changes should be **adjustment entries**, not overwrite.

❌ Don’t allow **cross-ledger updates** manually:

* If one org → another org, both sides should sync auto, not manually edited.

**🧩 Summary for Developer**

* Ledger table auto-created on booking “paid” status.
* Must handle multiple parties (Org, Branch, Agent, Area Agent).
* 5 GET endpoints for viewing by level.
* Internal notes stored as text array.
* inventory\_owner\_organization\_id is key to all cross-company calculations.
* Currency (SAR/PKR) handled at auto conversion level.
* Receivable/payable summary auto-calculated using debit-credit.

**API: GET /api/agents/pending-balances**

**Purpose:**  
Yeh API un sab agents ki list deti hai jinka **final balance minus mein hai** (i.e. company ke against outstanding amount hai).  
System organization-wise data fetch karega.(all agency and area agency)

**Request:**

GET /api/agents/pending-balances?organization\_id={organization\_id}

**Query Parameters:**

| **Field** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| organization\_id | string | ✅ | Jis organization ke agents ka data chahiye |

**Response (200 OK):**

{

"organization\_id": "ORG12345",

"organization\_name": "Saer.pk",

"total\_pending\_agents": 5,

"agents": [

{

"agent\_id": "AGT001",

"agency\_name": "Star Travel",

"agent\_name": "Ahmed Raza",

"contact\_no": "+92 300 1234567",

"pending\_balance": -25000,

"internal\_note\_ids": MULTIPLE IDS },

{

"agent\_id": "AGT002",

"agency\_name": "Umrah Express",

"agent\_name": "Ali Khan",

"contact\_no": "+92 333 9876543",

"pending\_balance": -12000,

"internal\_note\_ids": ["NOTE125"]

}

]

}

**Logic:**

* System **ledger table** se saare agents ka final\_balance check karega.
* Jo agents ka final\_balance < 0 hai unko list karega.
* Data **organization\_id** ke basis par filter hoga.
* internal\_note\_ids optional array hai (agar koi internal note linked hai).

**API: GET /api/final-balance**

**Purpose:**  
Yeh API kisi bhi **agent**, **area agent**, **organization**, ya **branch** ka **final balance** return karti hai (ledger ke base par total debit-credit summary).

**Request:**

GET /api/final-balance?type={type}&id={id}

**Query Parameters:**

| **Field** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| type | string | ✅ | "agent", "area\_agent", "organization", or "branch" |
| id | string | ✅ | ID of the respective entity (e.g. agent\_id, branch\_id, etc.) |

**Response (200 OK):**

{

"type": "agent",

"id": "AGT001",

"name": "Ahmed Travels",

"total\_debit": 250000,

"total\_credit": 230000,

"final\_balance": 20000,

"currency": "PKR",

"last\_updated": "2025-10-17T10:30:00Z"

}

**Logic:**

* System ledger entries se total **debit** aur **credit** calculate karega.
* final\_balance = total\_debit - total\_credit
* Positive balance → organization ke favour mein.
* Negative balance → agent/branch ne pay karna hai.
* Data real-time ledger ke according auto update hoga.

**🧩 1️⃣ Universal Register API**

**API Name:**

POST /universal/register

**Request Body:**

{

"id": "auto-generate",

"type": "organization | branch | agent | employee",

"parent\_id": "ID of parent (organization/branch)",

"name": "string",

"email": "string",

"phone": "string",

"cnic\_front": "file/url",

"cnic\_back": "file/url",

"address": "string",

"city": "string",

"visiting\_card": "file/url",

"dts\_license": "file/url", (FILES ATTACHEDMENT IS NOT REQUIRED)

"created\_at": "auto",

"updated\_at": "auto"

}

**Required Endpoints:**

| **Method** | **Endpoint** | **Purpose** |
| --- | --- | --- |
| POST | /universal/register | Create new record (organization / branch / agent / employee) |
| GET | /universal/list?type=agent | Get all agents |
| GET | /universal/list?type=branch | Get all branches |
| GET | /universal/list?type=employee | Get all employees |
| PUT | /universal/update/{id} | Update record details |
| DELETE | /universal/delete/{id} | Delete record |

**Logic & Relationship Rules:**

* type and parent\_id define hierarchy.
* If type = branch, → parent\_id must be an **organization**.
* If type = agent, → parent\_id must be a **branch**.
* If type = employee, → parent\_id can be either **organization** or **branch**.

**2️⃣ Registration Rules Table (Dynamic Guidelines)**

**Table Fields:**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | string | Auto-generated |
| type | string | "agent", "employee", "branch" |
| requirement\_text | string | Registration requirements |
| benefit\_text | string | Benefits of registration |
| city\_needed | string | Required only for branch (optional) |
| service\_allowed | string | Allowed services (for agent, optional) |
| post\_available | string | Post options (for employee, optional) |
| created\_at | datetime | Auto |
| updated\_at | datetime | Auto |

Intimation api exit and entry report of umrah pex

• Jab bhi agent booking API se koi passenger add hota hai/ BOOKING PAID HOTI HAI → system usko automatically “checkpoint” bana dega.

• Uske baad agar flight update hoti hai to system entry/exit track karega.

• Har pax ka status hamesha clear rahega:

• In Pakistan

• Entered KSA

• In KSA

• Exited KSA (verified / not verified)

In Makkah

Madina

Jeddah

OR ANY OTHER CITY IF DATA IS AVAILABLE

⸻

🔹 Database Design (short version)

pax\_movements

1. id

Flight ki ful details Kb ja raha AUR kb wapis a raha hai

2. pax\_id (linked to booking)

Pex passport info

3. flight\_no

4. departure\_airport

5. arrival\_airport

6. departure\_time AND DATE

7. arrival\_time AND DATE

8. status (in\_pakistan, entered\_ksa, in\_ksa, exited\_ksa, exit\_pending)

9. verified\_exit (true/false)

Organisation

10. agent\_id

11. created\_at

12. updated\_at

13. Is this reported to shirka yes or no (this will change to auto no if agnet update data of flight of any dep or return flight.)

⸻

🔹 API Endpoints

1. POST /booking/create

• Create booking & auto generate pax movement checkpoint.

2. PUT /pax-movement/update/{id}

• Update flight info (entry/exit).

• If new flight → update status accordingly.

3. GET /pax-movement/status/{id}

• Return pax status (in Pakistan, in KSA, exited, etc).

4. GET /pax-movement/summary

• Returns total counts:

• how many in Pakistan

• how many in KSA

. HOW MANY IN MAKKAH AND MADINA AND OR ANY OTHER CITY IF DATA AVAILABLE

• how many exited

5. POST /pax-movement/verify-exit/{id}

• Admin manually checks system (PNR/exit report) → confirm exit or reject.

6. POST /pax-movement/notify-agent

• If pax not exited as claimed → send auto notification to agent to update flight info.

⸻

🔹 Example Workflow

1. Agent books → POST /booking/create → pax created in pax\_movements with status = in\_pakistan.

2. Flight update entry KSA → status = entered\_ksa then in\_ksa.

3. Return flight update → status = exit\_pending.

4. Admin verifies → if yes → status = exited\_ksa + verified\_exit=true.

5. If no → system auto → notify\_agent.

**📊 REPORTS MODULE API STRUCTURE (DJANGO REST FRAMEWORK)**

**1️⃣ Sales Counting Report API**

**Endpoint:**  
GET /api/v1/reports/sales-summary/

**Query Params:**

* date\_from → Start date (YYYY-MM-DD)
* date\_to → End date (YYYY-MM-DD)
* organization\_id → Required (for which company report is needed)
* agent\_id → Optional (filter)
* branch\_id → Optional (filter)

**Output Fields:**

{

"total\_bookings": 0,

"total\_group\_bookings": 0,

"total\_ticket\_bookings": 0,

"total\_umrah\_bookings": 0,

"total\_visa\_bookings": 0,

"total\_hotel\_nights": 0,

"total\_transport\_bookings": 0,

"total\_food\_bookings": 0,

"total\_ziyarat\_bookings": 0,

"total\_paid\_orders": 0,

"total\_unpaid\_orders": 0,

"total\_expired\_orders": 0,

"total\_amount": 0,

"total\_paid\_amount": 0,

"total\_unpaid\_amount": 0,

"total\_expired\_amount": 0,

"agent\_wise\_summary": [

{

"agent\_id": 101,

"agent\_name": "Saer.pk Islamabad Agent",

"total\_orders": 45,

"paid\_orders": 32,

"unpaid\_orders": 13,

"total\_sales\_amount": 1200000,

"paid\_sales\_amount": 850000,

"service\_breakdown": {

"umrah": { "count": 10, "amount": 400000 },

"visa": { "count": 8, "amount": 150000 },

"tickets": { "count": 20, "amount": 350000 },

"hotel": { "nights": 45, "amount": 200000 }

}

}

]

}

**Calculation Rules:**

* From Booking table:
  + total\_bookings = COUNT(all bookings)
  + total\_ticket\_bookings, etc. = COUNT filtered by category
  + total\_amount = SUM(total\_amount)
  + total\_paid\_amount = SUM(total\_amount WHERE payment\_status = "paid")
* From Ledger (optional for precision):
  + Adjust final\_amount where ledger entries exist for same booking\_id
* For total\_hotel\_nights:
  + SUM of hotel\_booking.total\_nights in same date range
* Apply date\_from, date\_to on booking\_date / created\_at
* Filter by org, agent, branch accordingly.

**2️⃣ Financial / Ledger Summary Report API**

**Endpoint:**  
GET /api/v1/reports/financial-summary/

**Query Params:**

* organization\_id → Required

**Output Fields:**

{

"organization\_id": 202,

"total\_receivable\_amount": 540000,

"total\_payable\_amount": 380000,

"receivable\_settled\_amount": 350000,

"receivable\_unsettled\_amount": 190000,

"payable\_settled\_amount": 250000,

"payable\_unsettled\_amount": 130000,

"net\_balance": 160000,

"by\_counterparty": [

{

"organization\_id": 303,

"organization\_name": "FlyWorld Travels",

"receivable": 90000,

"payable": 40000

}

],

"by\_agent": [

{

"agent\_id": 101,

"agent\_name": "Ahmad Tours",

"receivable": 150000,

"payable": 30000

}

]

}

**Calculation Rules (short):**

* **Receivable:**  
  SUM(amount) from Ledger where to\_company\_id = organization\_id
* **Payable:**  
  SUM(amount) from Ledger where from\_company\_id = organization\_id
* **Net Balance:**  
  total\_receivable\_amount - total\_payable\_amount
* **Settled / Unsettled split:**  
  Based on status = settled / pending
* **By Counterparty:**  
  Group by organization\_id and aggregate receivable/payable
* **By Agent:**  
  Group by agent\_id and aggregate receivable/payable

**3️⃣ Top Seller (Agent-Wise) Report API**

**Endpoint:**  
GET /api/v1/reports/top-sellers/

**Query Params:**

* date\_from → optional
* date\_to → optional
* organization\_id → required
* limit → optional (default = 10)
* sort\_by → total\_amount or total\_bookings

**Output Fields:**

[

{

"agent\_id": 101,

"agent\_name": "Ahmad Tours",

"total\_bookings": 48,

"total\_amount": 1250000,

"categories": [

{ "category": "ticket", "count": 20, "amount": 400000 },

{ "category": "umrah", "count": 12, "amount": 500000 },

{ "category": "visa", "count": 6, "amount": 150000 },

{ "category": "hotel", "count": 10, "amount": 200000 }

],

"ranking": 1

}

]

**Calculation Logic:**

* From Booking:
  + Group by agent\_id
  + Count total bookings per agent
  + Sum total\_amount per agent
* Group again by (agent\_id, category) for category breakdown.
* Ranking logic:
  + Sort by total\_amount (or total\_bookings) descending.
  + Assign ranking field sequentially.
* Filter optional: by organization, by date range, by booking status (only paid).

**Common Rules for All Reports**

✅ **General**

* Must support filters by organization\_id, branch\_id, agent\_id, date\_from, date\_to
* Default date range → current month
* Return data in consistent structure with status, message, and data

✅ **Performance**

* Use .annotate() and Sum(), Count() for aggregation.
* Avoid for-loops; aggregate at query level.
* Always select\_related('agent', 'organization') to reduce DB hits.

✅ **Pagination**

* Only for large list APIs like top sellers (limit, offset)

✅ **Permissions**

* Org admin → see all their agents + branches
* Branch → see their own data only
* Agent → see self only

✅ **Optional Caching**

* Cache heavy reports (e.g. sales-summary, financial-summary) for 10–15 min with Redis.

✅ **Auto Update**

* Whenever booking/ledger changes, update computed fields next cron or on-demand (via recalculation function).

**🔄 Example Model References**

* Booking → id, agent\_id, organization\_id, category, total\_amount, payment\_status, created\_at
* Ledger → from\_company\_id, to\_company\_id, amount, status, transaction\_type
* Agent → id, name, branch\_id, organization\_id
* Organization → id, name, type

**🧭 DAILY OPERATIONS MANAGEMENT APIS**

**🔹 Common Rules:**

* Sab APIs date filter ke sath kaam karein: ?date=YYYY-MM-DD
* Har service mein status update ho sakta hai: pending / started / completed / canceled
* Har record mein **pax\_id**, **first\_name**, **last\_name**, **booking\_id** mandatory.
* **Clicking on a pax name** → GET full pax details (separate endpoint at the end).

**1️⃣ HOTEL CHECK-IN / CHECK-OUT API**

**GET /daily/hotels**

Get all today’s hotel check-ins or check-outs.

**Query Params:**  
?date=YYYY-MM-DD&type=checkin / checkout

**Response Example:**

{

"date": "2025-10-17",

"hotels": [

{

"booking\_id": "BKG-101",

"Contact no of family head": "+92300-0709017",

"hotel\_name": "Hilton Makkah",

"city": "Makkah",

"check\_in": "2025-10-17",

"check\_out": "2025-10-20",

"status": "checked\_in / pending / checked\_out",

"pax\_list": [

{

"pax\_id": "PAX001",

"first\_name": "Ali",

"last\_name": "Raza",

"Contact no of pex": "+92300-0709017",

"room\_no": "204",

"bed\_no": "B1"

}

]

}

]

}

**PUT /daily/hotel/update-status**

Update check-in / check-out status.

{

"booking\_id": "BKG-101",

"pax\_id": "PAX001",

"status": "checked\_in / checked\_out / pending",

"updated\_by": "EMP-12"

}

**2️⃣ ZIYARAT MANAGEMENT API**

**GET /daily/ziyarats**

Get today’s scheduled ziyarats list.

{

"date": "2025-10-17",

"ziyarats": [

{

"booking\_id": "BKG-101",

"location": "Uhud Mountain",

"pickup\_time": "08:00 AM",

"status": "pending / started / completed / canceled",

"pax\_list": [

{ "pax\_id": "PAX001", "first\_name": "Ali", "last\_name": "Raza", "contact no": "+923000709017" }

]

}

]

}

**PUT /daily/ziyarats/update**

{

"booking\_id": "BKG-101",

"pax\_id": "PAX001",

"status": "completed / pending / canceled / not\_picked",

"updated\_by": "EMP-12"

}

**3️⃣ TRANSPORT MANAGEMENT API (City or Intercity Transfers)**

**GET /daily/transport**

Get all today’s transport jobs (pickup/drop between hotels or cities).

{

"date": "2025-10-17",

"transports": [

{

"booking\_id": "BKG-101",

"pickup": "Makkah Hotel",

"drop": "Madinah Hotel",

"vehicle": "Hiace",

"driver\_name": "Abdullah",

"status": "departed / arrived / pending",

"pax\_list": [

{ "pax\_id": "PAX001", "first\_name": "Ali", "last\_name": "Raza", "contact no": "+923000709017" }

]

}

]

}

**PUT /daily/transport/update**

{

"booking\_id": "BKG-101",

"pax\_id": "PAX001",

"status": "departed / arrived / pending / canceled",

"updated\_by": "EMP-12"

}

**4️⃣ AIRPORT PICKUP / DROP API**

**GET /daily/airport**

For all pickups/drops (based on flight timings).

{

"date": "2025-10-17",

"airport\_transfers": [

{

"booking\_id": "BKG-101",

"transfer\_type": "pickup / drop",

"flight\_number": "SV802",

"flight\_time": "15:30",

"pickup\_point": "Jeddah Airport",

"drop\_point": "Makkah Hotel",

"status": "waiting / departed / arrived",

"pax\_list": [

{ "pax\_id": "PAX001", "first\_name": "Ali", "last\_name": "Raza", "contact no": "+923000709017" }

]

}

]

}

**PUT /daily/airport/update**

{

"booking\_id": "BKG-101",

"pax\_id": "PAX001",

"status": "waiting / departed / arrived / not\_picked",

"updated\_by": "EMP-12"

}

**5️⃣ FOOD MANAGEMENT API**

**GET /daily/food**

Get all meals for today.

{

"date": "2025-10-17",

"meals": [

{

"booking\_id": "BKG-101",

"meal\_type": "Dinner",

"time": "08:00 PM",

"menu": "Biryani + Raita",

"location": "Makkah Hotel",

"status": "served / pending",

"pax\_list": [

{ "PAX001", "first\_name": "Ali", "last\_name": "Raza", "contact no": "+923000709017" }

]

}

]

}

**PUT /daily/food/update**

{

"booking\_id": "BKG-101",

"pax\_id": "PAX001",

"status": "served / pending / canceled",

"updated\_by": "EMP-12"

}

**6️⃣ GET PAX FULL DETAILS**

**GET /pax/details/{pax\_id}**

Get full details of one pax when clicked.

{

"pax\_id": "PAX001",

"first\_name": "Ali",

"last\_name": "Raza",

"passport\_no": "AB123456",

"family\_no": "FAM-20",

"booking\_id": "BKG-101",

"package\_type": "Umrah",

"flight": {

"departure": "LHE",

"arrival": "JED",

"flight\_time": "2025-10-17 15:30"

},

"hotel": [

{ "name": "Hilton Makkah", "check\_in": "2025-10-17", "check\_out": "2025-10-20" }

],

"transport": [

{ "pickup": "Airport", "drop": "Hotel", "status": "completed" }

],

"ziyarats": [

{ "location": "Uhud", "status": "completed" }

],

"food": [

{ "meal\_type": "Dinner", "status": "served" }

]

}

**🧠 Backend Logic Notes**

* Date filter is must → default = today.
* Every update auto-syncs to the main booking table.
* Pax details show combined data from all modules.
* All modules independent (hotel / ziyarats / transport / airport / food) → but connected by booking\_id + pax\_id.
* Airport transfers use **flight\_time** for coordination.
* Must show family head contact number when we get data

**API: Get All Unpaid Orders**

**Endpoint:**  
GET /api/bookings/unpaid

**Description:**  
Fetches **all unpaid bookings** (of all agents or clients under that organization) that are still **active (not expired)**.  
it simply returns **all unpaid orders** with pending balance > 0.

**Response Example**

{

"total\_unpaid": 2,

"unpaid\_bookings": [

{

"booking\_id": 101,

"booking\_no": "INV-101",

"customer\_name": "Ali Raza",

"contact\_number": "+92-300000000",

"total\_amount": 250000,

"paid\_payment": 50000,

"pending\_payment": 200000,

"expiry\_time": "2025-09-30T23:59:00Z",

"agent\_id": 12,

"status": "unpaid",

"call\_status": false,

"client\_note": null

},

{

"booking\_id": 102,

"booking\_no": "INV-102",

"customer\_name": "Fatima",

"contact\_number": "+92-300111111",

"total\_amount": 180000,

"paid\_payment": 0,

"pending\_payment": 180000,

"expiry\_time": "2025-09-28T23:59:00Z",

"agent\_id": 15,

"status": "unpaid",

"call\_status": true,

"client\_note": "Customer will pay tomorrow"

}

]

}

Unpaid order api

🔹 API 1: Get Unpaid Orders

GET /api/bookings/unpaid/org id

Response Example

{

"total\_unpaid": 2,

"unpaid\_bookings": [

{

"booking\_id": 101,

"booking\_no": "INV-101",

"customer\_name": "Ali Raza",

"contact\_number": "+92-300000000",

"total\_amount": 250000,

"paid\_payment": 50000,

"pending\_payment": 200000,

"expiry\_time": "2025-09-30T23:59:00Z",

"agent\_id": 12,

"status": "unpaid",

"call\_status": false,

"client\_note": null

},

{

"booking\_id": 102,

"booking\_no": "INV-102",

"customer\_name": "Fatima",

"contact\_number": "+92-300111111",

"total\_amount": 180000,

"paid\_payment": 0,

"pending\_payment": 180000,

"expiry\_time": "2025-09-28T23:59:00Z",

"agent\_id": 15,

"status": "unpaid",

"call\_status": true,

"client\_note": "Customer will pay tomorrow"

}

]

}

**Logic / Filters**

* ✅ status = unpaid
* ✅ pending\_payment > 0
* ✅ expiry\_time >= current\_date (exclude expired)
* ✅ or\_id → filter by organization
* ✅ Include all related agents & clients under the given organization\_id

Unpaid order api

🔹 API 1: Get Unpaid Orders

GET /api/bookings/unpaid/org id

Response Example

{

"total\_unpaid": 2,

"unpaid\_bookings": [

{

"booking\_id": 101,

"booking\_no": "INV-101",

"customer\_name": "Ali Raza",

"contact\_number": "+92-300000000",

"total\_amount": 250000,

"paid\_payment": 50000,

"pending\_payment": 200000,

"expiry\_time": "2025-09-30T23:59:00Z",

"agent\_id": 12,

"status": "unpaid",

"call\_status": false,

"client\_note": null

},

{

"booking\_id": 102,

"booking\_no": "INV-102",

"customer\_name": "Fatima",

"contact\_number": "+92-300111111",

"total\_amount": 180000,

"paid\_payment": 0,

"pending\_payment": 180000,

"expiry\_time": "2025-09-28T23:59:00Z",

"agent\_id": 15,

"status": "unpaid",

"call\_status": true,

"client\_note": "Customer will pay tomorrow"

}

]

}

🔹 API 2: Add Call Remarks (Update Booking Call Status)

POST /api/bookings/unpaid/remarks

Request Body

{

"booking\_id": 101,

"call\_status": true,

"Internal remarks (multiples ids

)

"created\_by": 7

}

🔹 Kaise Kaam Karega

1. GET /unpaid → system booking table me se payment\_status = unpaid ya pending\_payment > 0 filter karega.

2. POST /unpaid/remarks → call agent remarks add karega aur call\_status update karega.

3. Baad me reports me aap dekh sakte ho kis customer ko kitni baar call hui aur kya reply mila.

**Agency Profile API (Relationship & Work Overview)**

Endpoint:  
GET /api/agency/profile?agency\_id=123

**Notes for Dev:**  
This API will return the **complete relationship profile** of any agency, including behavior, work status, and history.

**Response Structure Example:**

{

"agency\_id": 123,

"agency\_name": "Al Saer Travels",

"contact\_person": "Ahmed Khan",

"contact\_number": "+92-300000000",

"relationship\_status": "active",

"relation\_history": [

{

"date": "2025-10-15",

"type": "discussion",

"note": "Talked about new Umrah rates"

},

{

"date": "2025-09-20",

"type": "conflict",

"note": "Delayed payment for 2 weeks"

}

],

"working\_with\_companies": [

{

"organization\_id": 1,

"organization\_name": "Saer.pk",

"work\_type": ["Umrah Packages", "Tickets"]

},

{

"organization\_id": 2,

"organization\_name": "Al Noor Travels",

"work\_type": ["Hotels", "Visa"]

}

],

"performance\_summary": {

"total\_bookings": 85,

"on\_time\_payments": 79,

"late\_payments": 6,

"disputes": 1,

"remarks": "Overall good performance, some delay in payments."

},

"recent\_communication": [

{

"date": "2025-10-10",

"by": "Admin",

"message": "Confirmed next Umrah batch."

}

],

"conflict\_history": [

{

"date": "2025-08-25",

"reason": "Misunderstanding over refund",

"resolved": true

}

]

}

**Purpose:**  
– To view **agency behavior, performance, and communication history**.  
– To check **which organizations** the agency works with and **how the relationship is going**.  
– Helps admin/team know if the agency is **reliable, problematic, or improving**.

**POST API — Add / Update Agency Profile Details**

Endpoint:  
POST /api/agency/profile

**Purpose:**  
To add or update complete relationship information of an agency — including their work history, communication, conflicts, and associated companies.

### 🧩 **Request Body Example**

{

"agency\_id": 123,

"relationship\_status": "active",

"relation\_history": [

{

"date": "2025-10-17",

"type": "discussion",

"note": "Talked about upcoming Umrah package commission"

},

{

"date": "2025-09-28",

"type": "meeting",

"note": "Met in office, discussed hotel rates"

}

],

"working\_with\_companies": [

{

"organization\_id": 1,

"organization\_name": "Saer.pk",

"work\_type": ["Tickets", "Hotels"]

},

{

"organization\_id": 3,

"organization\_name": "FlySmart Travels",

"work\_type": ["Visa", "Umrah Packages"]

}

],

"performance\_summary": {

"total\_bookings": 90,

"on\_time\_payments": 85,

"late\_payments": 5,

"disputes": 0,

"remarks": "Active and responsive agent with good market relationship"

},

"recent\_communication": [

{

"date": "2025-10-16",

"by": "Admin",

"message": "Shared new Umrah package details"

}

],

"conflict\_history": [

{

"date": "2025-08-12",

"reason": "Late commission clearance",

"resolved": true,

"resolution\_note": "Payment cleared within 3 days"

}

]

}

### ✅ **Response Example**

{

"success": true,

"message": "Agency profile updated successfully",

"updated\_profile": {

"agency\_id": 123,

"relationship\_status": "active"

}

}

### ⚙️ **Dev Notes**

* If agency\_id exists → update existing record.
* If not → create a new agency profile.
* Auto-track **created\_by**, **updated\_by**, and **timestamps**.
* Future scope:
  + Add auto-sync with booking and payment APIs to auto-update performance.
  + Add filter for relationship\_status (active, inactive, risky, dispute, etc.).
  + Option to upload documents or screenshots as **evidence for disputes or meetings**.

**1. API — Hotel Availability with Map**

**Endpoint:**  
GET /api/hotels/availability?hotel\_id=123&date\_from=YYYY-MM-DD&date\_to=YYYY-MM-DD

**Purpose:**  
To fetch hotel availability (rooms, beds, floors, maps) with occupied/available status based on bookings.

**Response Example**

{

"hotel\_id": 123,

"hotel\_name": "Hilton Makkah",

"total\_rooms": 120,

"total\_quint-rooms": 120,

"total\_quad-rooms": 120,

"total\_Triple-rooms": 120,

"total\_double-rooms": 120,

"available\_rooms": 35,

"available\_beds": 35,

"available\_sharing-beds": 35,

"available\_quint-rooms": 35,

"available\_quad-rooms": 35,

"available\_Triple-rooms": 35,

"available\_double-rooms": 35,

"occupied\_rooms": 85,

"floors": [

{

"floor\_no": 1,

"floor\_map\_url": "https://cdn.saer.pk/maps/floor\_1.png",

"rooms": [

{

"room\_id": 101,

"room\_no": "101",

"room\_type": "Double",

"capacity": 2,

"available\_beds": 0,

"status": "occupied",

"current\_booking\_id": 5023,

"guest\_names": ["Ali Raza", "Ahmed Khan"],

"checkin\_date": "2025-10-17",

"checkout\_date": "2025-10-20"

},

{

"room\_id": 102,

"room\_no": "102",

"room\_type": "Triple",

"capacity": 3,

"available\_beds": 1,

"status": "partially\_occupied",

"current\_booking\_id": 5024,

"guest\_names": ["Usman"],

"checkin\_date": "2025-10-17",

"checkout\_date": "2025-10-21"

}

]

}

]

}

**🧩 2. API — Assign Room/Bed to Pax**

**Endpoint:**  
POST /api/hotels/assign-room

**Purpose:**  
To assign a pax (passenger) to a room or specific bed and update hotel map availability instantly.

**Request Body Example**

{

"booking\_id": 5024,

"hotel\_id": 123,

"pax\_id": 987,

"room\_id": 102,

"bed\_no": 2,

"assigned\_by": "admin\_001",

"checkin\_date": "2025-10-17",

"checkout\_date": "2025-10-21"

}

**Response Example**

{

"success": true,

"message": "Room assigned successfully",

"assigned\_details": {

"room\_id": 102,

"room\_no": "102",

"bed\_no": 2,

"pax\_id": 987,

"hotel\_id": 123,

"status": "occupied"

}

}

**🧩 3. API — Room & Bed Map Management (Admin Panel)**

**Endpoint:**  
POST /api/hotels/room-map

**Purpose:**  
To create or update floor/room/bed layout with map image and coordinates.

**Request Body Example**

{

"hotel\_id": 123,

"floor\_no": 2,

"floor\_map\_url": "https://cdn.saer.pk/maps/floor\_2.png",

"rooms": [

{

"room\_no": "201",

"room\_type": "Quad",

"capacity": 4,

"beds": [

{ "bed\_no": 1, "status": "available" },

{ "bed\_no": 2, "status": "available" },

{ "bed\_no": 3, "status": "available" },

{ "bed\_no": 4, "status": "available" }

]

},

{

"room\_no": "202",

"room\_type": "Double",

"capacity": 2,

"beds": [

{ "bed\_no": 1, "status": "available" },

{ "bed\_no": 2, "status": "available" }

]

}

]

}

**🧩 4. Auto Availability Rules (Backend Logic)**

| **Trigger** | **Action** |
| --- | --- |
| **New Booking Confirmed** | Mark assigned rooms/beds as occupied |
| **Checkout Completed** | Auto change status to cleaning\_pending |
| **Cleaning Done** | Change room/beds to available again |
| **Booking Cancelled** | Free the assigned room/beds instantly |
| **Manual Override** | Admin can set custom status manually (e.g., maintenance, reserved, etc.) |

**🧩 5. Booking Integration**

* When a booking is created with hotel\_id, system auto-checks:
  + Room type availability
  + Dates overlap
  + Current room map
* Assigns available rooms automatically if not assigned manually.
* Keeps room\_assignment under booking JSON:

"hotel\_details": {

"hotel\_id": 123,

"room\_assignments": [

{ "pax\_id": 987, "room\_id": 102, "bed\_no": 2 }

]

}

**🧩 6. Auto Sync Points**

* /api/hotels/availability auto-syncs with:
  + /api/bookings (for active and future bookings)
  + /api/hotels/assign-room
* Cron job updates status daily (check-ins / checkouts auto-refresh).

Jin orders ka balance payable hai un orders ki list har agnecy ki load ho one api.

**🧩 1. API — Create / Update Rules (Terms & Conditions)**

**Endpoint:**  
POST /api/rules/create

**Purpose:**  
Admin can create new rules or update existing ones, and define **where** they will be displayed (e.g. Booking Page, Agent Dashboard, Hotel Info Page, etc.)

**Request Body Example**

{

"id": null,

"title": "Umrah Booking Terms",

"description": "All Umrah bookings are subject to advance payment and visa approval.",

"rule\_type": "terms\_and\_conditions",

"pages\_to\_display": ["booking\_page", "agent\_portal"],

"is\_active": true,

"language": "en",

"created\_by": "admin\_001"

}

**Response Example**

{

"success": true,

"message": "Rule created successfully",

"rule\_id": 23

}

**🧩 2. API — Get Rules**

**Endpoint:**  
GET /api/rules/list?type=terms\_and\_conditions&page=booking\_page

**Purpose:**  
Fetch rules dynamically based on where they need to be shown.

**Response Example**

{

"rules": [

{

"id": 23,

"title": "Umrah Booking Terms",

"description": "All Umrah bookings are subject to advance payment and visa approval.",

"pages\_to\_display": ["booking\_page", "agent\_portal"],

"is\_active": true

},

{

"id": 24,

"title": "Hotel Cancellation Policy",

"description": "Cancellation within 24 hours of check-in is non-refundable.",

"pages\_to\_display": ["hotel\_page"],

"is\_active": true

}

]

}

**🧩 3. API — Update / Delete Rule**

**Endpoint:**  
PUT /api/rules/update/{id}  
DELETE /api/rules/delete/{id}

**PUT Request Example**

{

"title": "Updated Umrah Terms",

"description": "Advance payment must be cleared within 48 hours after confirmation.",

"pages\_to\_display": ["booking\_page", "agent\_portal"],

"is\_active": true

}

**🧩 4. Dynamic Display Rules**

| **Page / Section** | **Example Displayed Rules** |
| --- | --- |
| booking\_page | Payment, cancellation, change policies |
| agent\_portal | Agent commissions, usage rights, lead policies |
| hotel\_page | Check-in/out, refund, room policy |
| transport\_page | Pickup, delay, liability rules |
| visa\_page | Document, rejection, refund rules |

**🧩 5. Auto Behavior**

* When is\_active = true, the rule **automatically appears** on assigned pages.
* Frontend dynamically pulls from /api/rules/list?page={page\_name}
* Rules are **versioned** (keep history for compliance).
* language field allows multilingual display (Urdu / English both).

Jin orders ka balance payable hai un orders ki list har agnecy ki load ho one api.

**🏨 WALK-IN CUSTOMERS (HOTEL MANAGEMENT MODULE)**

**🔹 Purpose:**

To manage customers who come directly to the hotel (not through online booking or agent).  
System should handle:

* Booking / Check-in / Check-out
* Room assignment
* Price, discount, and profit tracking
* Ledger + Organization earning

**🔸 1. POST /api/walkin/create**

**Purpose:** Create a new walk-in booking directly from the hotel panel.

**Request Example:**

{

"hotel\_id": 201,

"organization\_id": 10,

"booking\_type": "walk\_in",

"customer": {

"name": "Ahmed Khan",

"cnic": "35202-1234567-8",

"phone": "+92-3121234567",

"address": "Lahore, Pakistan"

},

"room\_details": [

{

"room\_id": 101,

"room\_no": "A-102",

"bed\_type": "double",

"price\_per\_night": 8000,

"discount": 500,

"check\_in": "2025-10-17T14:00:00Z",

"check\_out": "2025-10-18T12:00:00Z"

}

],

"advance\_paid": 4000,

"payment\_mode": "cash",

"remarks": "Late night check-in"

}

**Auto Actions:**

* System auto-generates booking number (WALKIN-###).
* Room status → changes to “Occupied”.
* On check-out → room status auto → “Cleaning Pending”.
* Ledger entry auto-created under that organization (debit: customer, credit: organization revenue).

**🔸 2. GET /api/walkin/list**

**Purpose:** Show all walk-in customers (active + completed).

**Response Example:**

{

"total\_walkin\_bookings": 3,

"walkins": [

{

"booking\_no": "WALKIN-001",

"customer\_name": "Ahmed Khan",

"room\_no": "A-102",

"check\_in": "2025-10-17T14:00:00Z",

"check\_out": "2025-10-18T12:00:00Z",

"status": "checked\_in",

"total\_amount": 8000,

"paid": 4000,

"balance": 4000

},

{

"booking\_no": "WALKIN-002",

"customer\_name": "Fatima",

"room\_no": "B-201",

"status": "checked\_out",

"total\_amount": 6000,

"paid": 6000,

"profit": 1500

}

]

}

**🔸 3. PUT /api/walkin/update-status/{booking\_id}**

**Purpose:** Update check-in / check-out / cleaning status.

**Request Example:**

{

"status": "checked\_out",

"checkout\_time": "2025-10-18T12:30:00Z",

"final\_amount": 8200,

"remarks": "Extra 200 charged for late checkout"

}

**Auto Actions:**

* Ledger updates → credit organization, close booking balance.
* Room auto becomes “Cleaning Pending”.
* After cleaning confirmation → “Available” again.

**🔸 4. GET /api/walkin/summary**

**Purpose:** Show hotel’s profit/loss summary.

**Response Example:**

{

"hotel\_id": 201,

"date": "2025-10-17",

"total\_rooms": 25,

"occupied\_rooms": 10,

"available\_rooms": 15,

"total\_sales": 85000,

"total\_expense": 20000,

"profit": 65000

}

**🔸 5. Auto Ledger Entry Logic**

| **Event** | **Debit** | **Credit** | **Description** |
| --- | --- | --- | --- |
| Booking Created | Customer | Organization | Booking payment receivable |
| Advance Paid | Cash | Customer | Advance payment received |
| Check-out | Customer | Organization | Balance cleared |
| Expense (cleaning, service) | Expense Account | Cash | Recorded as cost |
| Profit Summary | - | - | Auto from total sales - total expense |

**🔸 6. Optional Add-ons**

✅ Add “walk-in” toggle in hotel availability API  
✅ Auto print mini invoice on check-out  
✅ Add filter: “Today Check-in / Today Check-out / In-house Guests”  
✅ Daily email or dashboard summary

**🔒 7. Security & Ownership Rules**

* Walk-in booking always belongs to hotel’s inventory\_owner\_organization\_id.
* Cannot assign walk-in booking to external agency.
* All payments stay internal under that organization’s ledger only.

**📘 Profit, Loss & Expense Management (Full Module Overview)**

**🎯 Goal**

To calculate and track **profit, loss, and expenses** for every service type — including **Hotels, Visas, Transports, Tickets, and Umrah Packages** — per organization, branch, and agent.

**1️⃣ Data Structure (Database Design / JSON Format)**

**Main Table: financial\_records**

Each record represents a single transaction entry related to any module.

{

"id": "auto\_generated",

"organization\_id": "uuid",

"branch\_id": "uuid",

"agent\_id": "uuid",

"module\_type": "hotel | visa | transport | ticket | umrah\_package",

"booking\_id": "uuid",

"reference\_no": "SAER-HTL-00125",

"income\_amount": 120000,

"expense\_amount": 85000,

"profit\_amount": 35000,

"loss\_amount": 0,

"description": "Profit from Umrah hotel booking (Makkah Hilton)",

"record\_date": "2025-10-17",

"created\_by": "user\_id",

"last\_updated\_by": "user\_id",

"status": "active | archived"

}

**2️⃣ Expense Management**

**Endpoint: POST /api/finance/expense/add**

Used to add any type of expense linked to module or independent expense (like salary, maintenance, etc.)

{

"organization\_id": "uuid",

"branch\_id": "uuid",

"expense\_type": "hotel\_cleaning | staff\_salary | fuel | visa\_fee | maintenance | other",

"module\_type": "hotel | visa | transport | ticket | umrah\_package | general",

"booking\_id": "optional\_uuid",

"description": "Fuel cost for airport transfer vehicle",

"amount": 4000,

"payment\_mode": "cash | bank | pending",

"paid\_to": "vendor\_name",

"expense\_date": "2025-10-17"

}

**3️⃣ Profit/Loss Auto Calculation (Backend Logic)**

* System will **auto-record** profit/loss whenever a **booking is confirmed or updated.**
* Formula:
* Profit = Total Selling Price - Total Purchase Cost - Total Expenses
* Loss = If (Profit < 0) then abs(Profit)
* All modules (Hotel, Visa, Transport, Ticket, Umrah) will send transaction data to financial\_records table through internal API calls.

**4️⃣ API Endpoints**

**✅ GET /api/finance/summary/all**

Return full financial summary by organization, branch, or agent.

**Query Params:**

organization\_id

branch\_id

agent\_id

module\_type (optional)

**Response:**

{

"organization\_id": "uuid",

"total\_income": 12400000,

"total\_expense": 8700000,

"total\_profit": 3700000,

"total\_loss": 0,

"breakdown\_by\_module": {

"hotel": { "income": 5000000, "expense": 3000000, "profit": 2000000 },

"visa": { "income": 2000000, "expense": 1600000, "profit": 400000 },

"transport": { "income": 1000000, "expense": 700000, "profit": 300000 },

"ticket": { "income": 4400000, "expense": 3400000, "profit": 1000000 }

}

}

**✅ GET /api/finance/ledger/by-service**

Returns detailed transaction list for one module.

**Query:**

module\_type=hotel&organization\_id=uuid

**Response:**

{

"records": [

{

"booking\_id": "uuid",

"reference\_no": "SAER-HTL-00125",

"income\_amount": 120000,

"expense\_amount": 85000,

"profit": 35000,

"record\_date": "2025-10-17",

"agent\_name": "Ahsan Travels"

}

]

}

**✅ GET /api/finance/expense/list**

List all expenses by type or date.

**Query Params:**

organization\_id

expense\_type (optional)

start\_date

end\_date

**5️⃣ Walk-in + Linked Booking**

* For walk-in customers (especially for hotel bookings):
  + Expense and income will auto-sync when checkout is marked “Done.”
  + Staff can manually adjust costs, taxes, or extra charges.
  + API will post data to financial\_records.

**6️⃣ Dashboard Summary**

Show real-time:

* Today’s Profit/Loss
* This Week / This Month
* By Module (Hotel, Ticket, etc.)
* By Branch / Agent

**7️⃣ Audit Trail**

Every change in financial record (update, delete) should store:

{

"action": "update",

"old\_value": { ... },

"new\_value": { ... },

"updated\_by": "user\_id",

"updated\_at": "timestamp"

}

**🧾 System Name:**

**Financial Bookkeeping & Tax Reporting System**

**🔹 Main Purpose:**

Track **every rupee movement** (income, expense, transfer) for:

* Each **organization**
* Each **branch**
* Each **agent**
* Each **booking type** (ticket, hotel, Umrah, visa, transport)
* And generate **FBR-ready profit/loss + tax return reports**

**🔹 Core Modules**

**1. Chart of Accounts (COA)**

All money movements are categorized:

| **Account Type** | **Examples** |
| --- | --- |
| Assets | Cash in hand, bank account, receivables |
| Liabilities | Payables, customer advances |
| Income | Ticket sales, Umrah packages, hotel income |
| Expenses | Salaries, rent, transport fuel, commission |
| Equity | Owner capital, retained earnings |

✅ *Auto created per organization and branch.*

**2. Transactions Journal**

Every event creates a journal entry (auto + manual allowed):

**Structure**

{

"id": 1,

"date": "2025-10-17",

"organization\_id": 101,

"branch\_id": 12,

"description": "Umrah package booking by Agent X",

"entries": [

{"account": "Cash", "type": "debit", "amount": 200000},

{"account": "Umrah Income", "type": "credit", "amount": 200000}

],

"source\_type": "booking/expense/manual",

"source\_id": 15001,

"created\_by": 2

}

✅ *Auto entries generated from bookings, invoices, and payments.*  
🧾 *Manual entries allowed via /manual/posting for adjustments.*

**3. Branch Ledger**

Tracks branch-wise transactions:

* Auto-linked to organization COA.
* Every branch has separate **cashbook**, **bank ledger**, and **expense ledger**.
* Can filter by:
  + Date
  + Transaction type
  + Booking type
  + Profit/loss report view

**4. Profit & Loss Engine (Auto)**

System auto-calculates P&L per branch, per product, and per period.

**Auto-calculation rule:**

Profit = (Total Income + Adjustments) - (Total Expenses + Discounts + Commissions)

**Endpoints**

GET /reports/profit-loss?branch\_id=12&month=2025-09

GET /reports/profit-loss?organization\_id=101&year=2025

✅ Calculates:

* Hotel profit/loss
* Ticket profit/loss
* Visa profit/loss
* Umrah profit/loss
* Transport profit/loss

**5. Expense Management**

* Manual & recurring expenses supported.
* Expense categories linked to chart of accounts.
* Example:
* {
* "type": "monthly\_rent",
* "branch\_id": 5,
* "amount": 45000,
* "date": "2025-10-01",
* "remarks": "October rent",
* "approved\_by": 1
* }

**6. FBR Return & Tax Report**

System auto-generates:

* Sales Tax Summary
* Income Tax Return Summary
* Withholding Tax on payments
* Yearly Profit Statement (for FBR)

**Endpoint**

GET /reports/fbr/summary?organization\_id=101&year=2025

**Auto Data Sources**

* All invoices (with tax fields)
* Expenses (with tax %)
* Payments (cross org or branch)

**7. Audit Trail**

Every entry → timestamp + user ID  
Full traceability for FBR audit.

**API:**

GET /audit/transactions?branch\_id=12&date\_from=...&date\_to=...

**8. Manual Posting & Adjustments**

When something isn’t auto (like office renovation, manual cash transfer):

POST /manual/posting

{

"date": "2025-10-15",

"branch\_id": 3,

"debit\_account": "Office Renovation Expense",

"credit\_account": "Cash",

"amount": 120000,

"description": "Renovation of Islamabad branch"

}

✅ *Goes directly to ledger + auto reflected in reports.*

**🔹 Reporting APIs Summary**

| **API** | **Purpose** |
| --- | --- |
| **GET /ledger/branch** | View branch-wise transactions |
| **GET /ledger/organization** | Consolidated financial report |
| **GET /reports/profit-loss** | Profit & loss (branch/org level) |
| **GET /reports/fbr/summary** | Yearly FBR summary |
| **GET /audit/transactions** | Complete history for audit |
| **POST /manual/posting** | Add manual entries |
| **GET /reports/balance-sheet** | Assets, liabilities, equity report |

**🔹 Automation Summary**

| **Task** | **Auto / Manual** |
| --- | --- |
| Booking income posting | ✅ Auto |
| Supplier payment | ✅ Auto |
| Expense entry | ⚙️ Manual |
| Bank deposit / withdrawal | ⚙️ Manual |
| Profit calculation | ✅ Auto |
| FBR tax summary | ✅ Auto |
| Adjustments | ⚙️ Manual (for corrections) |

**🧭 Module Name:**

**Passport Leads & Follow-up Management API**

**🔹 1️⃣ POST /passport-leads/create**

➡️ Create a new passport lead (for branch or customer).

**Request Body:**

{

"branch\_id": 12,

"organization\_id": 101,

"lead\_source": "walk-in / facebook / agent",

"customer\_name": "Ali Raza",

"customer\_phone": "+92-3001234567",

"cnic": "35202-1234567-1",

"passport\_number": "AB1234567",

"city": "Lahore",

"remarks": "Asked for Umrah package, said will decide in 2 days",

"followup\_status": "pending",

"next\_followup\_date": "2025-10-18",

"assigned\_to": 5,

"pending\_balance": 50000,

"pax\_details": [

{

"first\_name": "Ali",

"last\_name": "Raza",

"age": 34,

"gender": "male",

"passport\_number": "AB1234567",

"nationality": "Pakistani"

}

]

}

✅ *Auto create PAX record linked with lead.*

**🔹 2️⃣ GET /passport-leads/list**

➡️ Get all passport leads (with filters).

**Query Params:**

/passport-leads/list?branch\_id=12&status=pending&date\_from=2025-10-01&date\_to=2025-10-31

**Response Example:**

{

"total\_leads": 3,

"leads": [

{

"lead\_id": 201,

"customer\_name": "Ali Raza",

"customer\_phone": "+92-3001234567",

"passport\_number": "AB1234567",

"pending\_balance": 50000,

"followup\_status": "pending",

"next\_followup\_date": "2025-10-18",

"remarks": "Interested in Umrah package",

"branch\_id": 12,

"assigned\_to\_name": "Ahmed"

}

]

}

**🔹 3️⃣ GET /passport-leads/{lead\_id}**

➡️ Get full details of one lead + all PAX under it.

**Response Example:**

{

"lead\_id": 201,

"customer\_name": "Ali Raza",

"customer\_phone": "+92-3001234567",

"pending\_balance": 50000,

"followup\_status": "pending",

"next\_followup\_date": "2025-10-18",

"remarks": "Waiting for customer response",

"pax\_details": [

{

"pax\_id": 1,

"first\_name": "Ali",

"last\_name": "Raza",

"passport\_number": "AB1234567",

"gender": "male",

"nationality": "Pakistani",

"previous\_bookings": [

{"booking\_id": 101, "type": "Umrah", "status": "completed"}

]

}

]

}

✅ *Each pax’s old booking & payment history auto loads.*

**🔹 4️⃣ PUT /passport-leads/update/{lead\_id}**

➡️ Update lead status, remarks, follow-up, or pending balance.

**Request Example:**

{

"followup\_status": "completed",

"remarks": "Customer booked Umrah",

"pending\_balance": 0,

"next\_followup\_date": null

}

✅ Auto linked with branch ledger — if pending balance cleared → transaction auto closes.

**🔹 5️⃣ DELETE /passport-leads/{lead\_id}**

➡️ Soft delete a lead (kept in archive for audit).

**🔹 6️⃣ GET /passport-leads/followups/today**

➡️ Get today’s all pending or due follow-ups for call team.

**Response:**

{

"total\_due": 5,

"followups": [

{

"lead\_id": 201,

"customer\_name": "Ali Raza",

"phone": "+92-3001234567",

"remarks": "Will pay today",

"next\_followup\_date": "2025-10-17"

}

]

}

**🔹 7️⃣ POST /pax/update/{pax\_id}**

➡️ Update or edit any PAX record (for re-use in next booking).

{

"first\_name": "Ali",

"last\_name": "Raza",

"passport\_number": "AB1234567",

"phone": "+92-3001234567",

"notes": "Frequent Umrah traveller"

}

✅ *PAX reused automatically in next booking forms.*

**🔹 8️⃣ GET /pax/list**

➡️ Show all saved PAX of organization or branch with search filter.

**Query Example:**

/pax/list?branch\_id=12&search=Ali

**🔹 Automation Summary**

| **Function** | **Auto / Manual** | **Description** |
| --- | --- | --- |
| Link lead to booking | ✅ Auto | Once booking created, lead marked as converted |
| Follow-up reminder | ✅ Auto | Shows in dashboard on due date |
| Pending balance link to ledger | ✅ Auto | Updates branch balance |
| PAX record save/update | ✅ Auto | Once passport added, stored globally |
| Manual remark entry | ⚙️ Manual | By agent or branch operator |

**✅ Core Benefit**

* All **passport leads + follow-ups + pending balances + pax records** in one unified API.
* Agent or branch can easily manage calls, re-book customers, or check who still owes balance.

**🧩 Module Name:**

**Customer Data Auto-Collection API (Branch + Area + Leads + Bookings)**

**🔹 Objective:**

Automatically collect and merge all **customer contact details** (name, phone, email, city, source, etc.) from all existing APIs — **Passport Leads**, **Booking**, and **Area Branch customers** — into one centralized list for marketing, follow-ups, and data analysis.

**Endpoints**

**1️⃣ GET /customers/auto-collection**

➡️ Get all customers automatically collected from all sources (Bookings, Leads, Area Customers).

**Query Params:**

/customers/auto-collection?branch\_id=12&organization\_id=5

**Response Example:**

{

"total\_customers": 5,

"customers": [

{

"customer\_id": 101,

"full\_name": "Ali Raza",

"phone": "+92-3001234567",

"email": "ali.raza@gmail.com",

"city": "Lahore",

"source": "Booking",

"last\_activity": "2025-10-14",

"service\_type": "Umrah Package",

"branch\_id": 12,

"organization\_id": 5

},

{

"customer\_id": 102,

"full\_name": "Fatima",

"phone": "+92-3338889999",

"email": "fatima@gmail.com",

"city": "Karachi",

"source": "Passport Lead",

"last\_activity": "2025-10-12",

"service\_type": "Visa Inquiry",

"branch\_id": 12,

"organization\_id": 5

}

]

}

✅ Automatically merges data from:

* Bookings (customer info)
* Passport Leads (contact info)
* Area Branch records (clients who shared contact numbers)

**2️⃣ GET /customers/{id}**

➡️ Get full details of one customer from any source (lead or booking).

**Response Example:**

{

"customer\_id": 101,

"full\_name": "Ali Raza",

"phone": "+92-3001234567",

"email": "ali.raza@gmail.com",

"city": "Lahore",

"source": "Booking",

"total\_bookings": 3,

"last\_service": "Ticket",

"last\_contacted\_on": "2025-10-14",

"notes": [

{"text": "Interested in Hajj 2026", "date": "2025-10-12"}

]

}

**3️⃣ POST /customers/manual-add**

➡️ Add a new walk-in or untracked customer manually.

**Request Example:**

{

"name": "Ahmed Khan",

"phone": "+92-3009876543",

"email": "ahmed@gmail.com",

"city": "Faisalabad",

"source": "Walk-in",

"branch\_id": 12,

"organization\_id": 5

}

**4️⃣ DELETE /customers/{id}**

➡️ Delete a customer from the collection list (if duplicate or incorrect).

**5️⃣ AUTO SYNC**

Whenever a **new booking** or **passport lead** is created, or when an **area agent/branch** saves a customer number or email →  
system automatically checks:

* If same number/email already exists → update activity
* If new → create new customer record

**🧠 Developer Notes**

* Auto-collect data from:
  + /api/bookings
  + /api/passport/leads
  + /api/area/customers
* Merge duplicates by phone/email
* Always attach branch\_id, organization\_id, and source
* No ledger or payment logic required

### 💡 **Goal:**

Split any booking safely (e.g. 1 group booking → 2 smaller bookings)  
Auto-manage ledgers, totals, journal items, and dependencies.

### 🔧 **POST API (Booking Split API)**

**POST /api/bookings/split**

#### Request Body Example:

{

"original\_booking\_id": "BKG-1023",

"split\_reason": "Customer group divided into 2 separate travel plans",

"split\_by\_user\_id": "USR-2301",

"new\_booking\_structure": [

{

"pax\_ids": ["PAX-1", "PAX-2"],

"hotel\_ids": ["HOTEL-101"],

"transport\_ids": ["TRN-201"],

"ziyarat\_ids": ["ZIY-15", "ZIY-16"],

"food\_ids": ["FD-12"],

"payment\_adjustment": 25000

},

{

"pax\_ids": ["PAX-3"],

"hotel\_ids": ["HOTEL-102"],

"transport\_ids": ["TRN-202"],

"ziyarat\_ids": [],

"food\_ids": ["FD-13"],

"payment\_adjustment": 18000

}

],

"auto\_ledger\_update": true,

"notes": "Split due to different return dates"

}

#### Response:

{

"status": "success",

"message": "Booking successfully split into 2 new bookings.",

"new\_booking\_ids": ["BKG-2024-A", "BKG-2024-B"],

"ledger\_updates": [

{

"booking\_id": "BKG-2024-A",

"ledger\_id": "LEDGER-8891",

"auto\_entry": true

},

{

"booking\_id": "BKG-2024-B",

"ledger\_id": "LEDGER-8892",

"auto\_entry": true

}

]

}

### ⚙️ **How It Works (System Flow)**

1. **Clone original booking** → Duplicate structure & references (hotels, pax, payments, etc.).
2. **Assign selected pax and items** to each new booking.
3. **Auto-create new booking IDs** and link them back to original\_booking\_id.
4. **Recalculate totals** (per pax, per service, tax, commissions, discounts).
5. **Ledger auto-update:**
   * Create a new ledger entry per booking.
   * Update journal\_items.
   * Keep audit trail (who split, when, and why).
6. **Auto close original booking** or mark as partially\_split.
7. **Auto generate history logs** for FBR/tax and booking audit.

### 🧾 **Supporting APIs (Optional Helpers)**

**1. Get booking details before splitting**

GET /api/bookings/{booking\_id}

**2. Update ledger after split**

POST /api/ledger/update-after-split

**3. Merge back (if needed)**

POST /api/bookings/merge

### 🧠 **Important Notes**

* Always keep original\_booking\_id in both new bookings for traceability.
* Use soft-delete (mark is\_active: false) for removed items — don’t hard delete.
* Ledger must be generated automatically from split data, no manual entry needed.
* Auto-calculate commission or profit changes based on new totals.
* History and audit logs must record each split and data change.

# 📞 LEAD GENERATION & AREA CUSTOMER MANAGEMENT API (For Branch Customers Only)

## 🔹 1. PURPOSE

This API is used to:

* Collect **passport leads** and customer data when a walk-in or local customer visits the office.
* Automatically save customer details for future bookings and follow-ups.
* Manage **lead follow-ups**, **loan commitments**, and **booking conversions**.
* Maintain full lead history, remarks, and conversion tracking (linked with PEX details and bookings).
* **Not** for agents — this is strictly for **branch area customers**.

## 🔹 2. API STRUCTURE

### 🧾 Main Table: Leads

{

"id": 1,

"customer\_full\_name": "Ahmed Ali",

"passport\_number": "AB1234567",

"passport\_expiry": "2028-03-01",

"contact\_number": "+923001234567",

"email": "ahmed@example.com",

"cnic\_number": "35201-1234567-8",

"address": "Lahore, Pakistan",

"branch\_id": 3,

"organization\_id": 1,

"lead\_source": "walk-in / call / whatsapp / facebook / referral",

"lead\_status": "new / followup / confirmed / lost",

"interested\_in": "ticket / umrah\_package / visa / transport / hotel",

"interested\_travel\_date": "2025-12-01",

"next\_followup\_date": "2025-11-20",

"next\_followup\_time": "14:00",

"remarks": "Customer wants to travel in December, waiting for salary.",

"loan\_promise\_date": "2025-11-15",

"loan\_status": "pending / cleared / overdue",

"last\_contacted\_date": "2025-10-18",

"conversion\_status": "not\_converted / converted\_to\_booking / lost",

"booking\_id": null,

"pex\_id": null,

"created\_by\_user\_id": 22,

"created\_at": "2025-10-19T10:00:00Z",

"updated\_at": "2025-10-19T12:00:00Z"

}

## 🔹 3. FUNCTIONALITY FLOW

### 🟢 (A) Auto Lead Creation from Booking

Whenever a **booking** is created:

* System checks if passport\_number or contact\_number already exists in Leads.
* If not found → auto create new lead record.
* If found → auto link booking with that lead.

**Result:**  
All customer data becomes reusable by passport number or phone number.

### 🟢 (B) Manual Lead Creation

Branch user can manually create a new lead from:

* Walk-in customer
* WhatsApp inquiry
* Cold call

Required fields: name, passport\_number, contact\_number

### 🟢 (C) Lead Follow-up Management

Each lead will have a **follow-up log**:

#### Follow-up API Example:

{

"lead\_id": 1,

"followup\_date": "2025-11-20",

"followup\_time": "14:00",

"contacted\_via": "call / whatsapp / in-person",

"remarks": "Customer said will confirm next week.",

"next\_followup\_date": "2025-11-27",

"followup\_result": "pending / confirmed / lost"

}

➡ Each entry auto-saves in **FollowUpHistory** table.  
➡ Once booking is confirmed, status auto-updates to confirmed.

### 🟢 (D) Loan or Payment Commitment Tracking

If any customer has **payment due before travel**, store their commitment:

{

"lead\_id": 1,

"booking\_id": 20,

"promised\_clear\_date": "2025-11-15",

"status": "pending / cleared / overdue",

"remarks": "Customer said will clear remaining 50,000 before 15 Nov."

}

System can auto alert branch user when:

* Date = promised\_clear\_date
* Status = still pending → mark as **overdue**

### 🟢 (E) Lead Conversion Tracking

When lead converts to booking:

* conversion\_status = converted\_to\_booking
* Auto link booking\_id + pex\_id
* Auto create ledger if applicable

If not converted:

* Mark as lost
* Add reason in remarks.

### 🟢 (F) Auto Search by Passport or Contact Number

When creating a booking, system checks:

* If passport/contact found in Leads → auto fetch full customer details  
  (name, contact, CNIC, email, etc.)
* Prefill booking form with that data.
* Save any new updates back to the lead record.

## 🔹 4. RELATED TABLES

### 1️⃣ Leads

Stores main lead data (passport, contact, etc.)

### 2️⃣ FollowUpHistory

Stores all communication records for each lead.

### 3️⃣ LoanCommitments

Tracks payment promise data.

## 🔹 5. RULES & VALIDATIONS

✅ **Auto-create** lead on first contact or booking.  
✅ Passport or contact number must be **unique per organization**.  
✅ Only **branch users** can create/update these leads (not agents).  
✅ All dates stored in UTC.  
✅ Use **pagination + filter**:

* by branch\_id
* by lead\_status
* by next\_followup\_date
* by created\_date range  
  ✅ Always store created\_by\_user\_id and updated\_at.  
  ✅ Maintain full conversion history in one thread.

## 🔹 6. API ENDPOINTS (Suggested)

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/leads/create/ | Create new lead |
| GET | /api/leads/list/ | Get all leads (pagination + filters) |
| GET | /api/leads/detail/<id>/ | View single lead |
| PUT | /api/leads/update/<id>/ | Update lead info |
| POST | /api/leads/followup/ | Add follow-up record |
| POST | /api/leads/loan-promise/ | Add or update loan commitment |
| GET | /api/leads/search/?passport=AB1234567 | Auto search for existing lead |
| PUT | /api/leads/convert/<id>/ | Mark lead as converted to booking |
| PUT | /api/leads/lost/<id>/ | Mark lead as lost |

## 🔹 7. AUTO ACTIONS (Triggers)

* **Auto-create Lead:** When new booking created without existing lead.
* **Auto-update Conversion:** When booking linked → lead auto updates.
* **Auto Reminder:** If next\_followup\_date == today → reminder shown to branch user.
* **Auto Overdue Loan:** If promised\_clear\_date < today and not cleared → mark overdue.

## 🧩 **1. Customer Lead API (Passport / Contact Based)**

**Purpose:**  
To store every new walk-in or call-in customer’s passport & contact info once — and auto-fetch it when creating bookings later.

### **Endpoints**

#### ➤ POST /area-leads/create

**Use:** Save new customer lead (passport + contact + personal info).  
**Body:**

{

"branch\_id": "BR123",

"lead\_type": "walkin",

"customer\_name": "Ahmed Khan",

"passport\_number": "AB1234567",

"contact\_number": "+923001234567",

"cnic": "35202-1234567-1",

"email": "ahmed@gmail.com",

"address": "Lahore",

"notes": "Visited office for Umrah info",

"lead\_status": "pending",

"source": "office\_walkin",

"created\_by": "employee\_id\_45"

}

#### ➤ GET /area-leads/search

**Use:** Auto-fill booking form by searching passport or contact.  
**Query params:**  
?passport\_number=AB1234567 or ?contact\_number=+923001234567

✅ Returns full stored customer data for quick auto-fill.

## 🧭 **2. Lead Follow-up Management API**

**Purpose:**  
Track communication and next follow-up reminders for unconfirmed leads.

### **Endpoints**

#### ➤ POST /area-leads/followup/create

**Use:** Log next call/meeting reminder or customer promise date.  
**Body:**

{

"lead\_id": "LID123",

"next\_followup\_date": "2025-10-25",

"next\_followup\_time": "15:30",

"remarks": "Customer said will confirm Umrah package after salary",

"followup\_status": "waiting\_response"

}

#### ➤ GET /area-leads/followup/today

**Use:** Get all leads that need contact today for reminders.

## 🗣️ **3. Lead Communication History API**

**Purpose:**  
Save every conversation or action on the lead (like CRM timeline).

#### ➤ POST /area-leads/conversation/add

{

"lead\_id": "LID123",

"message\_type": "call",

"summary": "Customer said will clear payment next week",

"recorded\_by": "employee\_id\_45",

"timestamp": "2025-10-19T16:30:00"

}

#### ➤ GET /area-leads/conversation/history?lead\_id=LID123

Returns all call/text/note history in timeline format.

## 💰 **4. Lead Loan / Payment Promise Tracker**

**Purpose:**  
If a customer has pending payments or loan-type booking, store promise date & enforce contact.

#### ➤ POST /area-leads/payment-promise/add

{

"lead\_id": "LID123",

"booking\_id": "BKG567",

"amount\_due": 35000,

"due\_date": "2025-10-27",

"customer\_promise": "Will clear before travel date",

"status": "pending"

}

#### ➤ GET /area-leads/payment-promise/upcoming

Returns customers who must be contacted today or before due date.

## ✅ **5. Lead Conversion / Status Change API**

**Purpose:**  
When lead turns into booking or is lost.

#### ➤ PATCH /area-leads/update-status

{

"lead\_id": "LID123",

"status": "converted",

"converted\_booking\_id": "BKG567",

"closed\_reason": null

}

or

{

"lead\_id": "LID123",

"status": "lost",

"closed\_reason": "Customer bought from other agency"

}

## 🔗 **6. System Behavior**

* Every **booking API** auto-checks passport\_number or contact\_number in leads table.  
  ➜ If found → auto-fill customer data.  
  ➜ If not found → auto-save as a new lead record.
* Every **area branch** has separate leads dataset (only visible to that branch).
* Agents **cannot** access or modify branch leads.

**🧩 1. Table: area\_leads**

Stores all customer passport/contact data + main lead info.

CREATE TABLE area\_leads (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

branch\_id VARCHAR(50) NOT NULL,

customer\_name VARCHAR(150),

passport\_number VARCHAR(50) UNIQUE,

contact\_number VARCHAR(50),

cnic VARCHAR(25),

email VARCHAR(120),

address TEXT,

source ENUM('office\_walkin','call','facebook','instagram','website','whatsapp','other') DEFAULT 'office\_walkin',

lead\_status ENUM('pending','waiting\_response','converted','lost') DEFAULT 'pending',

notes TEXT,

created\_by VARCHAR(50),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

**📅 2. Table: lead\_followups**

Stores each reminder or customer follow-up plan.

CREATE TABLE lead\_followups (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

lead\_id BIGINT NOT NULL,

next\_followup\_date DATE,

next\_followup\_time TIME,

remarks TEXT,

followup\_status ENUM('waiting\_response','done','cancelled') DEFAULT 'waiting\_response',

created\_by VARCHAR(50),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (lead\_id) REFERENCES area\_leads(id) ON DELETE CASCADE

);

**🗣️ 3. Table: lead\_conversations**

Logs all conversations, voice call summaries, or WhatsApp notes.

CREATE TABLE lead\_conversations (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

lead\_id BIGINT NOT NULL,

message\_type ENUM('call','whatsapp','text','note') DEFAULT 'note',

summary TEXT,

recorded\_by VARCHAR(50),

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (lead\_id) REFERENCES area\_leads(id) ON DELETE CASCADE

);

**💰 4. Table: lead\_payment\_promises**

Used when a customer owes money or promised a date for payment clearance.

CREATE TABLE lead\_payment\_promises (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

lead\_id BIGINT NOT NULL,

booking\_id VARCHAR(50),

amount\_due DECIMAL(10,2),

due\_date DATE,

customer\_promise TEXT,

status ENUM('pending','cleared','cancelled') DEFAULT 'pending',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (lead\_id) REFERENCES area\_leads(id) ON DELETE CASCADE

);

**🔄 5. Optional Link in Booking Table**

In your main bookings table, add:

ALTER TABLE bookings ADD COLUMN lead\_id BIGINT NULL;

ALTER TABLE bookings ADD FOREIGN KEY (lead\_id) REFERENCES area\_leads(id);

This way:

* When you create a booking → system auto-detects by passport or contact → links lead\_id.
* Booking conversion auto-updates lead\_status → “converted”.

**⚙️ 6. API Auto-Behavior Rules**

| **Action** | **Trigger** | **Effect** |
| --- | --- | --- |
| New booking created | passport/contact match | auto-link lead\_id |
| New booking created (no match) | none found | auto-create new lead |
| Lead marked converted | manual or via booking | lead\_status → converted |
| Lead lost manually | no booking done | lead\_status → lost |
| Follow-up date reached | daily cron | show in reminders dashboard |
| Loan due date reached | daily cron | alert branch team |

# 🧩 INVENTORY PERMISSION & BOOKING SYSTEM RULES (Updated Text for Developer)

## 🔹 1. ORGANIZATION LINK API

**Purpose:**  
Super Admin can link organizations to share inventory.

**Fields:**

{

"main\_organization\_id": 1,

"linked\_organizations": [

{

"linked\_organization": 2,

"request\_status": "accepted"

},

{

"linked\_organization": 3,

"request\_status": "rejected"

},

{

"linked\_organization": 4,

"request\_status": "pending"

}

]

}

**Request Status Types:**

* accepted
* pending
* rejected

**Rules:**

* Only **Super Admin** can perform these actions.
* Once a request is created, both parties see the inventory share request in **Partners Page**.
* Shared inventories are visible only after both sides accept.

## 🔹 2. INVENTORY MODULES UPDATES

### 🟢 TICKETS API

* Add field: reselling\_allowed (boolean)
* Only show tickets:
  + Where **organization is allowed**
  + Not expired (no passed dates)
  + Status = active
  + available\_seats > 0
* Do **not show**:
  + Inactive
  + Zero-seat
  + Not shareable tickets

### 🟢 UMRAH PACKAGES API

* Add field: reselling\_allowed (boolean)
* Add fields for commission:
  + area\_agent\_commission\_per\_adult
  + area\_agent\_commission\_per\_child
  + area\_agent\_commission\_per\_infant
  + branch\_commission\_per\_adult
  + branch\_commission\_per\_child
  + branch\_commission\_per\_infant
* Get filters:
  + Show own + allowed organization packages
  + Active only
  + Future-dated only
  + Exclude non-shareable packages

### 🟢 HOTELS API

* **Remove:** google\_drive\_link
* **Add:**
  + photos → multiple image upload allowed
  + inventory\_owner\_organization\_id
  + reselling\_allowed (boolean)
* **Filters:**
  + Show own + allowed organization hotels
  + Show only active inventories
  + Cannot edit others’ inventory data
  + Exclude non-shareable hotels

### 🟢 DISCOUNTS API

**Structure:**

{

"name": "Winter Offer",

"group\_type": "organization/group/agent",

"organization": 1,

"is\_active": true,

"discounts": {

"group\_ticket\_discount\_amount": 500,

"umrah\_package\_discount\_amount": 1000,

"hotel\_night\_discounts": [

{

"quint\_discount": 300,

"quad\_discount": 250,

"triple\_discount": 200,

"double\_discount": 150,

"sharing\_discount": 100,

"other\_discount": 0,

"discounted\_hotels": [1, 2, 3]

},

{

"quint\_discount": 200,

"quad\_discount": 150,

"triple\_discount": 100,

"double\_discount": 50,

"sharing\_discount": 0,

"other\_discount": 0,

"discounted\_hotels": [4, 5]

}

]

}

}

## 🔹 3. BOOKING API CHANGES

### 🧾 Additions:

"journal\_items": [

{

"name": "Service Fee",

"price": 1000,

"quantity": 1,

"extra": "optional field"

}

],

"payments": [1, 2, 3], // Multiple payment IDs

"pex\_details": {

"pex\_id": 12

}

**New Columns:**

| **Field** | **Description** |
| --- | --- |
| area\_agency\_id | For area agency tracking |
| discount\_id | Linked discount applied |
| area\_agency\_discount\_type | Type of discount used |
| created\_by\_user\_id | Booking created by which user |
| organization\_id | Selling organization |
| branch\_id | Booking branch |
| agency\_id | Linked agency ID |
| package\_owner\_organization\_id | Package owner organization |
| expiry\_time | Auto expiry timestamp |
| booking\_type | ticket / umrah\_package / hotel / transport / visa / combined |
| is\_full\_package | Boolean (True = package owner rule applies) |

### 🧾 Additional Details

**Hotels:**  
Add → owner\_organization\_id, room\_no, bed\_no

**Tickets:**  
Add → owner\_organization\_id, ticket\_type  
(ticket\_type = buy\_from\_us OR data\_only)

**Transport:**  
Add → total\_seats

## 🔹 4. AGENCY / AREA AGENCY CHANGES

| **Field** | **Description** |
| --- | --- |
| credit\_limit | Max purchase limit without payment |
| credit\_limit\_days | Max allowed due days before lock |
| agency\_type | full / area |
| agency\_code | Auto generated unique ID |

**Rule:**  
If due days > credit\_limit\_days → lock new bookings until payment cleared.

## 🔹 5. PAYMENTS API

**Fields:**

{

"method": "cash/bank/kuickpay/other",

"amount": 0,

"remarks": "Payment against booking #123",

"status": "pending/approved/rejected",

"image": "string (optional proof)",

"transaction\_number": "string",

"organization": 1,

"branch": 2,

"agency": 3,

"agent": 4,

"created\_by": 10,

"agent\_bank": 5,

"organization\_bank": 6,

"kuickpay\_trn": 7

}

## 🔹 6. GET FILTER RULES (For All Inventory APIs)

✅ Always show:

* Organization’s own inventories
* Allowed & linked organization inventories

🚫 Never show:

* Inactive or expired items
* Non-shareable inventories
* Zero seat items (in tickets)

## 🔹 7. GENERAL RULES (Developer Standards)

* Always use **pagination** for list endpoints.
* Support **filter by organization\_id, branch\_id, date\_from, date\_to, category, agent\_id** where applicable.
* Always validate permissions before showing linked organization data.
* Maintain consistent JSON response with data, count, next, previous.
* Add signals / triggers for:
  + Auto ledger creation on booking.
  + Auto expiry based on expiry\_time.
  + Auto update of total paid/unpaid amounts.
* Always wrap bulk data in transactions.
* Use loops carefully — avoid N+1 queries (use .select\_related() / .prefetch\_related()).
* Every POST/PUT API must validate:
  + Ownership
  + Credit limit
  + Active status
* Every model must have fields:
  + created\_at, updated\_at, created\_by, modified\_by.
* Write reusable functions for totals / commissions.

**🎯 Auto Seat Management Logic**

**Affected Tables:**

* tickets\_inventory
* umrah\_packages\_inventory
* bookings

Each inventory table already has these columns:

total\_seats INT,

booked\_seats INT,

confirmed\_seats INT,

available\_seats INT

**⚙️ Logic Flow — Booking Lifecycle Automation**

**1️⃣ When a new booking is created:**

**Condition:** booking\_status = "unpaid" or "pending"

**Action:**

booked\_seats += number\_of\_passengers

available\_seats -= number\_of\_passengers

✅ Update in:

* tickets\_inventory or umrah\_packages\_inventory (depending on category)

**2️⃣ When booking is marked as "paid" or "confirmed":**

**Condition:** booking\_status changes from "pending/unpaid" → "paid/confirmed"

**Action:**

booked\_seats -= number\_of\_passengers

confirmed\_seats += number\_of\_passengers

💡 *This moves seats from booked to confirmed once payment is done.*

**3️⃣ When booking expires or cancelled:**

**Condition:** booking\_status = "expired" or "cancelled"

**Action:**

available\_seats += number\_of\_passengers

IF booking\_status WAS "pending" THEN booked\_seats -= number\_of\_passengers;

IF booking\_status WAS "confirmed" THEN confirmed\_seats -= number\_of\_passengers;

💡 *This restores those seats back to availability.*

**4️⃣ When booking is edited (number of passengers changed):**

**Condition:** booking already exists and pax updated.

**Action:**  
System re-calculates difference:

seat\_difference = new\_pax\_count - old\_pax\_count

IF seat\_difference > 0:

available\_seats -= seat\_difference

booked\_seats += seat\_difference

ELSE:

available\_seats += ABS(seat\_difference)

booked\_seats -= ABS(seat\_difference)

**5️⃣ Cron job (daily auto-checker)**

Every midnight (or hourly if needed), run a **CRON JOB**:

UPDATE bookings

SET status = 'expired'

WHERE status = 'pending' AND expiry\_date < CURRENT\_DATE;

Then trigger seat restoration logic for those expired bookings automatically.

**💻 Example API Logic (Pseudocode)**

def update\_inventory\_on\_booking(booking):

inv = get\_inventory(booking.item\_id, booking.category)

if booking.status == "pending":

inv.booked\_seats += booking.pax

inv.available\_seats -= booking.pax

elif booking.status == "confirmed":

inv.confirmed\_seats += booking.pax

inv.available\_seats -= booking.pax

elif booking.status in ["cancelled", "expired"]:

if booking.previous\_status == "pending":

inv.booked\_seats -= booking.pax

elif booking.previous\_status == "confirmed":

inv.confirmed\_seats -= booking.pax

inv.available\_seats += booking.pax

save\_inventory(inv)

**📊 API Summary**

| **API** | **Purpose** | **Triggers** |
| --- | --- | --- |
| /bookings/create | On new booking | Auto update booked & available seats |
| /bookings/update\_status | On payment/confirmation | Move booked → confirmed |
| /bookings/cancel\_or\_expire | On cancel or expiry | Add seats back to available |
| /system/cron/expire\_bookings | Daily auto expire | Auto update expired + seat restore |