perfect — here's a single, end-to-end Swagger walkthrough that matches the system you've built (MongoDB + ASP.NET Core, unified login, BackOffice application + admin review, station ownership, operator creation, owner bookings, approvals, cancellations). I've kept all JSONs copy-pasteable for Swagger's "Try it out".

0) Notes before you start

- Open Swagger at **http://localhost:8085/swagger**.
- When a step says "Authorize", click the **Authorize** button at the top and **paste only the JWT** (your Swagger is configured to accept the raw token — **no Bearer prefix**).
- Schemas below match what you shared (e.g., OwnerRegisterRequest, StationCreateRequest, BookingCreateRequest, etc.), plus the new ones we added (LoginRequest, BackOfficeApplyRequest, AdminCreateRequest, ...).

---

1) Bootstrap: create your very first Admin (one-time, outside Swagger)

Because only an Admin can create more Admins or approve BackOffice applications, you need **one** admin in the DB to begin.

Quick seed option (Mongo shell / Compass): insert a document in ev_owners:


```
{
  "nic": "ADM-SEED-0001",
  "fullName": "Seed Admin",
  "email": "seed.admin@example.com",
  "emailLower": "seed.admin@example.com",
  "phone": "+94770000000",
  "passwordHash": "<bcrypt hash for Admin@123>",
  "isActive": true,
  "roles": ["Admin"],
  "createdAtUtc": { "$date": "2025-01-01T00:00:00Z" }
}
```

- For passwordHash, generate with any bcrypt tool for the password Admin@123 (cost 10–12).
- If you already have an admin, skip this step**. (done skip it)**

---

2) Login (Admin)

POST /api/Auth/login

Body:

{

"username": "seed.admin@example.com",

"password": "Admin@123"

}

Response (AuthLoginResponse):

- accessToken (JWT) — copy it.
- roles should include Admin.

Authorize in Swagger using the token you just received.

---

3) BackOffice applies (public)

POST /api/BackOffice/apply (Anonymous)

Body (BackOfficeApplyRequest):

{

"fullName": "Blue Charge Pvt Ltd",

"email": "owner@bluecharge.lk",

"phone": "+94771234567",

"password": "BlueCharge@123",

"businessName": "Blue Charge",

"contactEmail": "contact@bluecharge.lk",

"contactPhone": "+94112456789"

}

Response: OwnerResponse (role will be BackOffice but their **ApplicationStatus = Pending** in the profile; you'll approve it next).

You'll need the **BackOffice NIC** for approval. If it's not shown in this response, list pending applications in the next step and copy it from there.

---

4) Admin reviews pending BackOffice applications

Still **Authorized** as Admin.

- **GET** /api/Admin/backoffices?status=Pending&page=1&pageSize=20
  Note the nic of the applicant (e.g., "BO-7c9e2184d2f3").
- **Approve**
  PUT /api/Admin/backoffices /{nic}/approve
  Example path: /api/Admin/backoffice-applications/BO-7c9e2184d2f3/approve

(If you want to test rejection once, use /reject first, then re-apply and approve.)

---

5) BackOffice logs in

POST /api/Auth/login (Anonymous)

Body:

{

"username": "owner@bluecharge.lk",

"password": "BlueCharge@123"

}

Response has roles: ["BackOffice"]. **Authorize** using this new token so all subsequent BackOffice endpoints work.

---

6) BackOffice creates a Station (owned by this BackOffice)

POST /api/Station (Auth: BackOffice/Admin)
(When called by BackOffice, the service stamps BackOfficeNic on the station internally.)

Body (StationCreateRequest):


```
{
  "name": "Blue Charge – Union Place",
  "type": "AC",
  "connectors": 2,
  "autoApproveEnabled": false,
  "lat": 6.9187,
  "lng": 79.8573,
  "defaultSlotMinutes": 60,
  "hoursTimezone": "Asia/Colombo",
  "pricing": {
    "model": "flat",
    "base": 250.0,
    "perHour": 0,
    "perKwh": 0,
    "taxPct": 8.0
  }
}
```


Response: StationResponse — copy the id (e.g., 64f1f0…).

---

7) Upsert Station schedule

PUT /api/Station/{id}/schedule (Auth: BackOffice/Admin)

Pick the id from step 6.
Body (StationScheduleUpsertRequest) — non-overlapping ranges, valid times:


```
{
```

```
 "weekly": {
  "mon": [{ "start": "08:00", "end": "20:00" }],
  "tue": [{ "start": "08:00", "end": "20:00" }],
  "wed": [{ "start": "08:00", "end": "20:00" }],
  "thu": [{ "start": "08:00", "end": "20:00" }],
  "fri": [{ "start": "08:00", "end": "20:00" }],
  "sat": [{ "start": "09:00", "end": "18:00" }],
  "sun": []
 },
 "exceptions": [
  { "date": "2025-12-25", "closed": true }
 ],
 "capacityOverrides": [
  { "date": "2025-12-10", "connectors": 1 }
 ]
}
```

---

8) BackOffice creates an Operator & (optionally) assigns stations

POST /api/BackOffice/operators (Auth: BackOffice)

Body (OperatorCreateRequest) — include the station ID from step 6 to assign:

```
{
 "fullName": "Kasun Perera",
 "email": "kasun.operator@bluecharge.lk",
 "phone": "+94775553311",
 "password": "Operator@123",
 "stationIds": ["<YOUR_STATION_ID_FROM_STEP_6>"]
```

}

Response: OwnerResponse (role Operator).

To later change assignments: **PUT** /api/BackOffice/operators/{operatorNic}/stations with:

{

"stationIds": ["<stationId1>", "<stationId2>"]

}

---

9) Operator logs in (optional)

POST /api/Auth/login

{

"username": "kasun.operator@bluecharge.lk",

"password": "Operator@123"

}

Response roles includes Operator.

The JWT includes an operator-stations claim internally so you can authorize operator actions per station in future "Session" flows.

---

10) Public discovery (no auth)

- **GET** /api/Station — public list (filters: type, status, minConnectors, paging).
- **GET** /api/Station/nearby?lat=6.9167&lng=79.8560&radiusKm=5&type=AC — shows stations near a point with a 7-day availability **summary**.
- **GET** /api/Station/{id} — fetch a station.
- **GET** /api/Station/{id}/schedule — public schedule.

---

11) EV Owner registration (public)

POST /api/EvOwner (Anonymous)

Body (OwnerRegisterRequest) — use a valid NIC (12-digit or 9 digits + V/X):

{

"nic": "200012345678",

"fullName": "Nadeesha Fernando",

"email": "nadeesha.owner@example.com",

"phone": "+94771230000",

"password": "Owner@123",

"addressLine1": "12 Flower Road",

"addressLine2": "Colombo 7",

"city": "Colombo"

}

If phone validator complains, try "0771230000".

---

12) EV Owner logs in

POST /api/Auth/login

{

"username": "200012345678",

"password": "Owner@123"

}

Response roles includes Owner. **Authorize** with this token to do bookings.

---

13) Owner creates a Booking

POST /api/Booking (Auth: Owner)

Body (BookingCreateRequest) — choose a future date **that matches the station's working hours** and is not closed by exceptions; start time inside a range; minutes must be one of the allowed slot sizes (30/45/60/90/120, per your validators). Example:

{

"stationId": "<YOUR_STATION_ID_FROM_STEP_6>",

"localDate": "2025-12-10",

"startTime": "10:00",

"minutes": 60,

"notes": "Nissan Leaf"

}

Response: BookingResponse with status usually **Pending** unless your station has autoApproveEnabled=true.

---

14) BackOffice/Admin approves or rejects the booking

(Or if autoApproveEnabled=true at station creation, skip this.)

- **Login** as BackOffice (step 5) or Admin (step 2) and **Authorize**.
- **GET** /api/Booking?stationId=<id>&status=Pending&page=1&pageSize=20 (BackOffice/Admin)
  Copy the booking id.
- **Approve**
  PUT /api/Booking/{id}/approve
  Response: BookingApprovalResponse with a qrToken and qrExpiresAtUtc.
- or **Reject**
  PUT /api/Booking/{id}/reject

---

here's a compact Swagger testing guide just for the pieces we implemented (QR issue/verify + Sessions check-in/finalize), with ready-to-paste mongosh snippets to bump a booking into the "right now" window.

Swagger guide (QR + Sessions)

0) Open & Authorize

- **Open Swagger (e.g., http://localhost:8085/swagger).**
- **Click Authorize (lock icon).**
  Paste only the JWT (your Swagger is configured to not require Bearer prefix).

Roles you'll use

- **Owner: can call POST /api/Qr/issue/{bookingId}.**
- **Operator (or BackOffice/Admin): can call POST /api/Qr/verify, POST /api/Sessions/checkin, POST /api/Sessions/finalize.**

Switch tokens anytime: click Authorize → Logout → paste a different token.

New endpoints (what they do)

1) POST /api/Qr/issue/{bookingId} (roles: Owner, BackOffice, Admin)

- **What: Issue (or re-issue) a raw QR token for an Approved booking.**
- **Response:**

{

"id": "<bookingId>",

"status": "Approved",

"qrToken": "<RAW_TOKEN>",

"qrExpiresAtUtc": "2025-10-04T12:34:56Z"

}

- **Notes: Server stores only SHA-256 hash of the token + an expiry. If you call this again, the old token is implicitly revoked.**

2) POST /api/Qr/verify (roles: Operator, BackOffice, Admin)

- **What: Validate a raw QR token from the scanner before check-in.**
- **Request:**

{ "qrToken": "<RAW_TOKEN>" }

- **Response (200):**

{

"valid": true,

"bookingId": "<id>",

"stationId": "<id>",

"slotStartUtc": "2025-10-04T12:30:00Z",

"expiresAtUtc": "2025-10-04T12:45:00Z",

"status": "Approved"

}

- **If invalid/expired → 401 with reasons (e.g., "QR expired").**

3) POST /api/Sessions/checkin (roles: Operator, BackOffice, Admin)

- **What: Start the charging session; moves booking to CheckedIn.**
- **Request:**

{

"bookingId": "<BookingId>",

"qrToken": "<RAW_TOKEN>"

}

- **Rules:**
  - **Booking must be Approved.**
  - **Time window: slotStartUtc − 15m ≤ now ≤ slotStartUtc + slotMinutes + 15m.**
  - **Operator must be scoped to the station (Admin/BackOffice bypass).**
- **Response: BookingResponse with status: "CheckedIn".**

4) POST /api/Sessions/finalize (roles: Operator, BackOffice, Admin)

- **What: End the session; computes totals; marks booking Completed.**
- **Request:**

{

"bookingId": "<id>",

"energyKwh": 21.4,

"unitPrice": 110.0,

"notes": "Night tariff"

}

- **Response: SessionReceiptResponse (includes total, completedAtUtc, etc.).**

You still have your existing approval endpoint that calls the booking service; it returns a QR on approval (manual mode). That's your alternative to /api/Qr/issue/{bookingId}.

Two happy-path flows (in Swagger)

Flow A — Manual approval path (if booking is Pending)

1. **Approve booking with your staff token using your existing approval endpoint (check the "Bookings" tag in Swagger).**
   Response contains { qrToken, qrExpiresAtUtc }.
2. **Switch to Operator token → POST /api/Qr/verify with that qrToken.**
3. **POST /api/Sessions/checkin with the same qrToken.**
4. **POST /api/Sessions/finalize with { bookingId, energyKwh, unitPrice }.**

Flow B — Already Approved booking (re-issue QR)

1. **Use Owner (or staff) token → POST /api/Qr/issue/{bookingId} → copy qrToken.**
2. **Switch to Operator token → POST /api/Qr/verify → then checkin → finalize as above.**

Mongo shell: bump booking into the "now" window

Paste directly in mongosh. Replace the ObjectId with your booking's _id.
This keeps Status: "Approved" and sets:

- **SlotStartUtc = now + 5m**
- **SlotEndUtc = start + 60m**
- **QrExpiresAtUtc = start + 15m (scan grace)**

// 1) Move a booking to start ~now (and keep it Approved)

const id = ObjectId("YOUR_BOOKING_OBJECTID_HERE"); // e.g., ObjectId("68dfd865b63b62232ae51d0e")

```
const now   = new Date();
const start = new Date(now.getTime() + 5 * 60 * 1000);     // start in 5 minutes
const end   = new Date(start.getTime() + 60 * 60 * 1000);  // 60-minute slot
const exp   = new Date(start.getTime() + 15 * 60 * 1000);  // QR valid until start + 15 min


db.bookings.updateOne(
 { _id: id },
 {
   $set: {
     Status: "Approved",
     SlotStartUtc: start,
     SlotEndUtc: end,
     SlotStartLocal: start.toISOString().slice(0,16),
     SlotMinutes: 60,
     QrExpiresAtUtc: exp,
     UpdatedAtUtc: now,
     UpdatedBy: "swagger-test"
   }
 }
);
(Optional) Extend just the QR expiry if your QR expired while testing:
const id = ObjectId("YOUR_BOOKING_OBJECTID_HERE");
const b  = db.bookings.findOne({ _id: id });
if (b) {
  const exp = new Date(b.SlotStartUtc.getTime() + 15 * 60 * 1000); // start + 15m
```

db.bookings.updateOne({ _id: id }, { $set: { QrExpiresAtUtc: exp, UpdatedAtUtc: new Date(), UpdatedBy: "extend-qr" } });

}

If you need a fresh QR token (and to rotate the stored hash) without re-approving, call: POST /api/Qr/issue/{bookingId} (Owner/BackOffice/Admin token). Use the new qrToken in verify/check-in.

Example Swagger request bodies (copy/paste)

Qr → Verify

{

"qrToken": "PASTE_RAW_QR_TOKEN_HERE"

}

Sessions → CheckIn

{

"qrToken": "PASTE_SAME_QR_TOKEN_HERE"

}

Sessions → Finalize

{

"bookingId": "68dfd865b63b62232ae51d0e",

"energyKwh": 21.4,

"unitPrice": 110.0,

"notes": "Night tariff"

}

Common errors & quick fixes

- **“Check-in outside allowed window.”**
  → Bump times with the mongosh snippet above (make sure now is within start−15m … end+15m).
- **401 from /api/Qr/verify with “QR expired”.**
  → Either extend QrExpiresAtUtc in Mongo or call /api/Qr/issue/{bookingId} to get a fresh token.
- **403 on check-in (Operator token):**
  → Operator isn’t scoped to that station. Test with an Admin token or assign the station to the operator.

---

1) Have the right JWTs handy

- **Operator token (best for realistic tests)**
- **Admin/BackOffice token (can bypass station scoping if you need)**

In Swagger (/swagger):

- **Click Authorize (lock icon), paste only the JWT (no Bearer prefix).**
- **You can switch users anytime via Logout → paste another token.**

2) Make sure the Operator is scoped to the station

If your Operator was created via the BackOffice flows, it’s probably set.
If not, do this in mongosh (replace placeholders):

```
// Link operator to the station by ID (store as strings)

db.ev_owners.updateOne(

 { nic: "OP-XXXXXXX" }, // your operator NIC

 { $set: { operatorStationIds: ["68dfccbede4c66fbffb3a800"] } } // your station's _id as string

);
```

3) Prepare an Approved booking that starts “now”

Pick the booking _id you want to test with (replace below), then paste in mongosh:

```
// Make booking start ~now (UTC), keep it Approved, set QR expiry to start+15m
const id = ObjectId("YOUR_BOOKING_OBJECTID_HERE");
const now   = new Date();
const start = new Date(now.getTime() + 5*60*1000);    // start in 5 minutes
const end   = new Date(start.getTime() + 60*60*1000);  // 60-minute slot
const exp   = new Date(start.getTime() + 15*60*1000);  // scan grace

db.bookings.updateOne(
 { _id: id },
 { $set: {
    Status: "Approved",
    SlotStartUtc: start,
    SlotEndUtc: end,
    SlotStartLocal: start.toISOString().slice(0,16), // okay for testing
    SlotMinutes: 60,
    QrExpiresAtUtc: exp,
    UpdatedAtUtc: now,
    UpdatedBy: "swagger-operator-tests"
 }}
);
```

4) Get a raw QR token for that booking

In Swagger, with Owner (or Admin/BackOffice) token:

- **POST /api/Qr/issue/{bookingId}**
    - **Path param: the booking's _id**
    - **Response contains qrToken. Copy it — you'll need it for scan.**

Alternative: If your booking was just Approved via your back-office approval endpoint, you already received a qrToken there.

---

1) GET /api/Operator/inbox

Use: Operator token.

Query param:

- **date = station's local date in YYYY-MM-DD (e.g., if station is in Asia/Colombo and today is 2025-10-04, use 2025-10-04).**

Swagger → Try it out → Execute

- **Expect an array with today's Approved bookings for the operator's stations, sorted by time.**

Sample result snippet:


```
[
 {
  "id": "68dfd865b63b62232ae51d0e",
  "bookingCode": "BK-BE32",
  "stationId": "68dfccbede4c66fbffb3a800",
  "slotStartUtc": "2025-10-04T12:30:00Z",
  "slotMinutes": 60,
  "status": "Approved",
  "ownerNicMasked": "**********5678",
  "slotStartLocal": "2025-10-04T18:00"
 }
]
```


If you get an empty list:

- **Wrong date (use the station's timezone day),**
- **Operator not linked to the station (operatorStationIds),**

- **Booking not Approved (pending/rejected don’t show),**
- **Booking is on a different day.**

---

2) POST /api/Operator/scan

Use: Operator token.

Body:

{

"qrToken": "PASTE_THE_RAW_QR_TOKEN_HERE",

"bookingId": "YOUR_BOOKING_OBJECTID_HERE"

}

(bookingId is optional; it just cross-checks the token mapping.)

What it does:

- **Delegates to Sessions check-in.**
- **Validates QR (hash match + expiry), checks time-window (start−15m … start+slot+15m) and operator’s station scope.**
- **On success, booking becomes CheckedIn.**

Success response snippet (BookingResponse):

{

"id": "68dfd865b63b62232ae51d0e",

"bookingCode": "BK-BE32",

"ownerNic": "BO-CDEF4CD44",

"stationId": "68dfccbede4c66fbffb3a800",

"status": "CheckedIn",

"slotStartLocal": "2025-10-04T18:00",

"slotStartUtc": "2025-10-04T12:30:00Z",

"slotEndUtc": "2025-10-04T13:30:00Z",

"slotMinutes": 60,

"qrExpiresAtUtc": "2025-10-04T12:45:00Z"

}

Common errors (and fixes):

- **401 Invalid/Expired QR → Re-issue token (/api/Qr/issue/{bookingId}) or extend QrExpiresAtUtc in Mongo.**
- **403 ForbiddenStation → Operator not assigned to that station.**
- **409 CheckInWindow / StateConflict → Move the booking time window near now (mongosh snippet above) or ensure it's still Approved.**

---

3) POST /api/Operator/exception ✅ (not "exceptio")

Use: Operator token.

Body (choose one reason):

{

"bookingId": "YOUR_BOOKING_OBJECTID_HERE",

"reason": "NoShow",

"notes": "Customer never arrived"

}

Allowed reasons and transitions:

- **"NoShow": Approved → NoShow (only after slot end + 15m grace)**
- **"Aborted": CheckedIn → Aborted**
- **"CustomerCancelOnSite": Approved|CheckedIn → Cancelled**

Success response snippet (BookingResponse):

{

"id": "68dfd865b63b62232ae51d0e",

"status": "NoShow",

```
 "stationId": "68dfccbede4c66fbffb3a800",
 "slotStartUtc": "2025-10-04T12:30:00Z",
 "slotMinutes": 60,
 "updatedAtUtc": "2025-10-04T14:00:00Z"
}
```


Fast time adjustments for exception testing

- **Test “NoShow” quickly (make slot end in the past):**

```
const id = ObjectId("YOUR_BOOKING_OBJECTID_HERE");
const now = new Date();
const end = new Date(now.getTime() - 10*60*1000);  // ended 10 mins ago
const start = new Date(end.getTime() - 60*60*1000);

db.bookings.updateOne(
 { _id: id },
 { $set: {
    Status: "Approved",
    SlotStartUtc: start,
    SlotEndUtc: end,
    SlotStartLocal: start.toISOString().slice(0,16),
    SlotMinutes: 60,
    QrExpiresAtUtc: new Date(start.getTime() + 15*60*1000),
    UpdatedAtUtc: now,
    UpdatedBy: "exception-test"
 }}
);
```

// Now POST /api/Operator/exception with reason "NoShow"

- **Test “Aborted” (make sure you’re CheckedIn first):**
    1. **Run POST /api/Operator/scan to check-in**
    2. **Then POST /api/Operator/exception with "Aborted"**
- **Test “CustomerCancelOnSite”:**
    - **Works from Approved (no check-in) or CheckedIn:**
    - **Set status & times like the first snippet (Approved), then call exception with "CustomerCancelOnSite".**

Quick checklist if anything fails

- **Inbox empty? Check date/timezone, booking status, and operator station scope.**
- **Scan 401? Use the raw token from /api/Qr/issue/{bookingId}, not its hash; token not expired.**
- **Scan 403? Assign the station to the operator (operatorStationIds).**
- **Check-in window error? Nudge times with the “start in 5 minutes” mongosh snippet.**
- **Exception 409/InvalidState? Ensure the prior state matches the required transition.**

---

0) Prep (Swagger auth & roles)

1. **Open Swagger at http://localhost:8085/swagger.**
2. **Click Authorize, paste a JWT for the role you’re testing:**
    - **Owner (create bookings)**
    - **BackOffice/Admin (approve/reject + query audits)**
    - **Operator (scan/check-in)**
3. **Make sure your station has open hours today (InventoryRegenerator already created the next ~14 days of slots).**

---

1) Create a booking (Owner)

- **Endpoint: POST /api/Bookings**
- **Body (example):**

{

"stationId": "<YOUR_STATION_ID>",

"localDate": "YYYY-MM-DD",

"startTime": "HH:mm",

"minutes": 60,

"notes": "Test run"

}

- **Result: 200 OK with a booking.id.**

Tip: If you want to immediately test check-in, use the Mongo snippet you already use to shift the slot to "now + 5m".

---

2) Approve the booking (BackOffice/Admin)

- **Endpoint: POST /api/Bookings/{id}/approve**
- **Result: 200 OK with a QrToken and QrExpiresAtUtc.**

What this triggers

- **Audit: action = Approved, entityType = Booking, entityId = <bookingId>**
- **Notification: "BookingApproved" (queued to the stub)**

```
const id = ObjectId("<bookingId>");
const now = new Date();
const start = new Date(now.getTime() + 5*60*1000);
const end = new Date(start.getTime() + 60*60*1000);
const exp = new Date(start.getTime() + 15*60*1000);
```

```
db.bookings.updateOne(
 { _id: id },
 { $set: {
    Status: "Approved",
    SlotStartUtc: start,
    SlotEndUtc: end,
    SlotStartLocal: start.toISOString().slice(0,16),
    SlotMinutes: 60,
    QrExpiresAtUtc: exp,
    UpdatedAtUtc: now,
    UpdatedBy: "swagger-checkin-retry"
 }}
);
```

Mongo (verify audits):

```
db.audits.find({ entityType: "Booking", entityId: "<bookingId>" })
 .sort({ createdAtUtc: -1 }).pretty();
db.audits.find({ action: "Approved" })
 .sort({ createdAtUtc: -1 }).limit(5).pretty();
```

3) Check-in (Operator) — via QR

- **Endpoint: POST /api/Operator/scan**
- **Body:**

{ "qrToken": "<the token from approve>" }

- **Result: 200 OK booking response with status = "CheckedIn".**

What this triggers

- **Audit: action = CheckedIn, entityType = Booking**
- **Notification: “CheckedIn”**

If you prefer the Sessions endpoint instead of the scan:

- **POST /api/Sessions/checkin with { "bookingId":"...", "qrToken":"..." }**

---

4) Finalize (Operator or BackOffice/Admin)

- **Endpoint: POST /api/Sessions/finalize**
- **Body (example):**

{

"bookingId": "<booking-id>",

"energyKwh": 12.5,

"unitPrice": 100.0,

"notes": "OK"

}

- **Result: 200 OK receipt with totals, and booking ends as Completed.**

What this triggers

- **Audit: action = Completed, entityType = Booking**
- **Notification: "Finalized/Completed"**

---

5) Verify audits via API (BackOffice/Admin)

- **Endpoint: POST /api/Audits/search**
- **Body (examples)**

By booking

{

"entityType": "Booking",

"entityId": "<booking-id>",

"page": 1,

"pageSize": 50

}

By action

{

"action": "Approved",

"fromUtc": "2025-01-01T00:00:00Z",

"toUtc": "2030-01-01T00:00:00Z",

"page": 1,

"pageSize": 50

}

Expected items
You should see entries like (shape may vary slightly depending on your DTOs):

- **Approved (from step 2)**
- **CheckedIn (from step 3)**
- **Completed (from step 4)**

payload will contain small context fields (e.g., stationId, times, amounts).

---

6) (Optional) Inspect audits directly in Mongo

If you want to double-check the persistence:

```
// Most recent audits
db.audits.find({}).sort({ createdAtUtc: -1 }).limit(10).pretty()


// For one booking
db.audits.find({ entityType: "Booking", entityId: "<booking-id>" })
    .sort({ createdAtUtc: -1 })
    .pretty()


// By action
db.audits.find({ action: "Approved" })
    .sort({ createdAtUtc: -1 })
    .limit(5)
```

.pretty()

---

7) Verify notifications (stub)

We wired a stub NotificationService that queues events (and usually logs them). You can verify in one (or both) ways, depending on how you wired it:

A) Mongo "notifications" log (if persisted)

db.notifications.find({})

.sort({ createdAtUtc: -1 })

.limit(10)

.pretty()

You should see events such as BookingApproved, CheckedIn, Completed with minimal payload (nic, station, bookingId, etc.).

B) Application logs

Look at your Kestrel console while you perform the actions. You should see log lines like:

info: NotificationService[0]

Notification queued: type=BookingApproved bookingId=<...> owner=<...>

Common gotchas

- **401/403 on /api/Audits/search: you must be Admin or BackOffice.**
- **Check-in window: only from slotStartUtc - 15m to slotStartUtc + slotMinutes + 15m. Use your mongosh helper to shift times for quick tests.**
- **Inventory: If you see "CapacityFull" on create/update, your inventory might be saturated for that slot; try a different time or confirm station_slot_inventory for the slot.**

Quick end-to-end sanity list

1. **Owner creates booking → (optionally) shift time via mongosh.**

2. **BackOffice/Admin approves → Audit: Approved; Notification enqueued.**
3. **Operator scans QR → Audit: CheckedIn; Notification enqueued.**
4. **Operator/Admin finalizes → Audit: Completed; Notification enqueued.**
5. **Query /api/Audits/search and/or db.audits.find(...) to see the trail.**
6. **Check notifications logs/collection.**

---

8. Swagger Testing Playbook

8.1 Authorize

- Open /swagger, click Authorize, paste JWT (without 'Bearer ').

8.2 Issue a QR (if booking is already Approved)

- POST /api/Qr/issue/{bookingId} with Owner/BackOffice/Admin token; copy qrToken.

8.3 Verify + Check-in + Finalize

- POST /api/Qr/verify (Operator/Admin/BackOffice): { qrToken } → expect Valid=true.
- POST /api/Sessions/checkin: { qrToken, bookingId? } → status becomes CheckedIn.
- POST /api/Sessions/finalize: { bookingId, energyKwh, unitPrice, notes? } → receipt.

8.4 Operator flows

- GET /api/Operator/inbox?date=YYYY-MM-DD (Operator token) → day's Approved bookings for assigned stations.
- POST /api/Operator/scan { qrToken, bookingId? } → delegates to check-in.
- POST /api/Operator/exception { bookingId, reason, notes? } → NoShow / Aborted / Cancelled.

8.5 mongosh time helpers

```
// Move a booking to start ~now (for check-in testing)

const id = ObjectId("YOUR_BOOKING_OBJECTID_HERE");

const now   = new Date();

const start = new Date(now.getTime() + 5*60*1000);

const end   = new Date(start.getTime() + 60*60*1000);

const exp   = new Date(start.getTime() + 15*60*1000);
```

```
db.bookings.updateOne(
  { _id: id },
  { $set: {
      Status: "Approved",
      SlotStartUtc: start, SlotEndUtc: end,
      SlotStartLocal: start.toISOString().slice(0,16),
      SlotMinutes: 60,
      QrExpiresAtUtc: exp,
      UpdatedAtUtc: now, UpdatedBy: "swagger-tests"
  }}
);

// NoShow quick test (slot in recent past so grace elapsed)
const endPast = new Date(now.getTime() - 10*60*1000);
const startPast = new Date(endPast.getTime() - 60*60*1000);
db.bookings.updateOne(
  { _id: id },
  { $set: {
      Status: "Approved",
      SlotStartUtc: startPast, SlotEndUtc: endPast,
      SlotStartLocal: startPast.toISOString().slice(0,16),
      SlotMinutes: 60,
      QrExpiresAtUtc: new Date(startPast.getTime() + 15*60*1000),
      UpdatedAtUtc: now, UpdatedBy: "noshow-test"
  }}
);
```

Endpoints

GET /api/reports/summary

fromUtc (optional, ISO 8601) – default now−30d

toUtc (optional, ISO 8601) – default now

stationId (optional)

{"bookingsCreated":0,"approved":0,"rejected":0,"cancelled":0,"checkedIn":0,"completed":0,"approvalRate":0,"checkInRate":0,"completionRate":0,"revenueTotal":0,"energyTotalKwh":0}

GET /api/reports/timeseries/bookings

metric (required): created|approved|rejected|cancelled|checkedin|completed

fromUtc, toUtc (required)

granularity (required): day|week|month (UTC)

stationId (optional)

{"metric":"approved","granularity":"day","points":[{"bucketStartUtc":"2025-10-04T00:00:00Z","value":2}] }

GET /api/reports/timeseries/revenue

fromUtc, toUtc (required)

granularity (required): day|week|month (UTC)

stationId (optional)

{"metric":"revenue","granularity":"day","points":[{"bucketStartUtc":"2025-10-04T00:00:00Z","value":625.0}] }

GET /api/reports/stations/{stationId}/utilization

fromLocalDate (yyyy-MM-dd)

toLocalDate (yyyy-MM-dd)

{"stationId":"abc123","daily":[{"date":"2025-10-04","reserved":6,"capacity":16,"utilizationPct":0.375}] }

GET /api/reports/occupancy-heatmap

stationId (required)

fromUtc (required)

toUtc (required)

{"stationId":"abc123","cells":[{"dow":6,"hour":16,"avgReservedPct":0.5}] }

Testing Guide

Create station + schedule; ensure inventory is generated.

Create, approve, check-in, and finalize a booking within the next hour.

Call endpoints in Swagger: /api/reports/*; verify values in responses.

Use different date windows and granularity to validate bucketing.

15) Owner checks and manages their bookings

GET /api/Booking/mine?status=&fromUtc=&toUtc= (Auth: Owner)
GET /api/Booking/{id} — see a specific booking.
PUT /api/Booking/{id} — reschedule/update (if allowed by rules).
DELETE /api/Booking/{id} — cancel (before the configured cutoff).

Current cancel logic (as implemented): Owners can cancel while booking is **Pending** or **Approved**, **until** your configured cutoff (e.g., 2 hours before slot start). If within the cutoff or already Started/Completed, service returns a conflict (CancelCutoff / InvalidState).

---

16) BackOffice views their operators & stations

(BackOffice token)

- **GET** /api/BackOffice/operators?page=1&pageSize=20
- **GET** /api/BackOffice/stations?status=Active&page=1&pageSize=20

---

17) Admin creates another Admin

(Admin token)

POST /api/Admin/admins (AdminCreateRequest)

{
"fullName": "Second Admin",
"email": "second.admin@example.com",
"phone": "+94770000001",
"password": "Admin2@123"
}

Now the new admin can log in via /api/Auth/login with their **email**.

What each endpoint does (very brief)

- **/api/Auth/login** — unified login (Owner by NIC; staff by email). Embeds role claims and for operators, station scope.
- **/api/BackOffice/apply** — creates a BackOffice account in "Pending" state; admin must review.
- **/api/Admin/backoffice-applications** — list + approve/reject BackOffice applications.
- **/api/Station** — create/list/get/update/activate/deactivate; BackOffice-owned station if created by BO; public discovery endpoints.
- **/api/Station/{id}/schedule** — upsert/fetch opening hours, exceptions, daily capacity.
- **/api/BackOffice/operators** — BO can create operators and manage which of **their** stations each operator can access.
- **/api/EvOwner** — public owner registration + manage owner profile (where permitted).
- **/api/Booking** — owner creates/updates/cancels; staff approves/rejects; admin/backoffice can list by station/date/status.
- **/health** — liveness.

Example JWT usage in Swagger

- After login, click **Authorize** and paste the **raw JWT** from AuthLoginResponse.accessToken (no “Bearer ”).
- For testing different roles, simply log in again with the other account and re-Authorize, replacing the token.

Troubleshooting tips

- **Phone format**: If InvalidPhone, try E.164 (+9477xxxxxxx) or a 10-digit local number (077xxxxxxx).
- **Time windows**: Bookings must fit the station’s schedule (HH:mm within the ranges).
- **Cutoff**: Cancels or updates near the start time may hit CancelCutoff / BookingStartedOrPast.
- **Operator station scope**: When creating/updating operators, station IDs must belong to that BackOffice.

DTOs you’ll paste most often

- LoginRequest for unified login.
- BackOfficeApplyRequest to apply.
- AdminCreateRequest to create admins.
- OperatorCreateRequest + OperatorAttachStationsRequest for operator management.
- StationCreateRequest, StationScheduleUpsertRequest for stations/schedules.
- OwnerRegisterRequest for owner signup.
- BookingCreateRequest, BookingUpdateRequest for bookings.