Contents

EV Charging Station Booking System — Story-Wise End-to-End Workflow (Web + Android + Central Web Service)

Below is a single, coherent "movie" of the system from first setup to everyday operations. It weaves your assignment rules (IIS-hosted .NET Web API, NoSQL DB, fat-service logic, Android with SQLite, web UI for backoffice/operators) with your ideas (shared mobile login for EV Owners & Operators, maps, QR, approvals, deactivation rules). I've been explicit about tiny edge cases, timestamps, and guardrails so it's implementation-ready.

---

1) Cast, Roles, and Channels

Actors

- **System Admin** (superuser; oversees all businesses/back offices, stations, operators, and EV owners)
- **Backoffice (Station Admin)** (manages stations, schedules, operators, bookings governance)
- **Station Operator** (runs on-site ops; can also access limited web screens; uses mobile for scan/finalize)
- **EV Owner** (uses Android app)

Channels

- **Web App (Tailwind/Bootstrap/React allowed)** for system admin, backoffice, and optional operator dashboards

- **Android App (Pure Android + SQLite for local User Mgmt)** for EV Owners and Station Operators
- **Central Web Service (C# .NET Web API on IIS, MongoDB/NoSQL)** — *all business logic lives here (fat service)*

Global Technical Rules

- **All authoritative data** lives on the server (MongoDB).
- **All time in UTC** at the service; clients display in local time.
- **JWT auth** with role claims (SystemAdmin, Backoffice, Operator, EvOwner) and scoped permissions.
- **Constraints to enforce** (service-side):
    - Reservation must be within **7 days** of booking date.
    - Update/Cancel must be **≥ 12 hours** before the reservation start.
    - **Cannot deactivate** a station if it has **future approved/pending** bookings.

---

2) First-Time Setup (System Bootstrap)

2.1 System Admin on Web

- **Backoffice Registration Request (Web):**
    - **Backoffice submits request via public signup form.**
    - **Fields: Business name, contact person, email, phone, billing info, time zone.**
    - **Status defaults to PendingApproval.**
- **System Admin Review:**
    - **System Admin logs into web → reviews all pending requests.**
    - **Can Approve (activates account, sends email invite to set password) or Reject (with reason).**
- **On Approval:**
    - **Backoffice account moves to Active.**

- **Backoffice Admin receives email to set password and can now log in.**

Result: Backoffice must first request an account; System Admin controls final activation.

---

3) Backoffice Onboarding (Web)

3.1 Backoffice creates Stations

1. Click **"Create Station"** → Fill:
    - **Name**, **Geo-location** (picked via map; lat/lng stored; 2dsphere index in Mongo)
    - **Type**: AC / DC
    - **Connectors/Slots Capacity** (e.g., 4 connectors ⇒ 4 concurrent bookings per time slot)
    - **Operating Hours** (e.g., 06:00–22:00)
    - **Default Slot Duration** (e.g., 60 minutes; can differ by station or AC/DC)
    - **Pricing model** (flat/hourly/kWh; taxes)
    - **Status** = Active
2. Service stores station with **scheduling profile** (weekly template + exceptions).

3.2 Define Schedules / Availability Windows

1. In **Station > Schedule**, the Backoffice:
    - Builds a **weekly template** (e.g., Mon–Fri 06:00–22:00, Sat–Sun 08:00–20:00).
    - Adds **exceptions** (holiday closures or extended hours).
    - Optionally defines **per-day capacity overrides** (maintenance reduces capacity from 4 to 2).
2. The service **pre-computes bookable "time buckets"** (e.g., every 60 min) over the next **rolling 14 days** (to easily honor the "book within 7 days" rule while showing a bit more for preview).
    - Each bucket has computed **capacity = active connectors – maintenance holds**.

- Buckets are regenerated/updated by a background job when schedules change.

3.3 Create Station Operators

1. Backoffice **creates Operator accounts** and **assigns stations** they can operate.
2. Operator receives **mobile login** (email/SMS).
3. Operator role claim includes stationIds[].

Result: Stations exist with precise maps/schedules; operators are provisioned.

---

4) EV Owner Journey (Android, Shared Mobile Login)

4.1 Registration & Local SQLite

1. EV Owner opens app → **Register**:
   - **NIC (PK)**, name, email, phone, vehicle plate (optional), password.
   - App writes **owner profile** to **local SQLite** (User Mgmt requirement) and **syncs** to server.
   - Server validates NIC uniqueness and returns OwnerId, status Active.
2. **Login** (owner or operator uses same login screen):
   - App calls /api/Auth/login → gets JWT with role and user claims.
   - App updates local SQLite with **current profile + token expiry**.

If the account is **deactivated** by backoffice, login succeeds but **owner status** returned as Deactivated → app shows a banner "Contact backoffice to reactivate" and disables booking actions.

4.2 Discover Nearby Stations (Map)

1. Owner taps **"Find Stations"**:
   - App provides lat/lng (from device) and **filters** (AC/DC, radius).
   - Service queries Mongo with **$near** (2dsphere index) and returns stations with *next 7 days* availability summaries (counts per day).
2. Owner taps a station → sees:

- **Station details**, **map pin**, **connector capacity**, **price info**, **available time slots** (for next 7 days), per-slot **capacity remaining**.

4.3 Create a Reservation (Pending)

1. Owner selects **date/time slot** and **duration** (one or multiple contiguous buckets depending on policy).
2. App shows **"Review & Confirm"** sheet:
    - Station name, date/time (local), expected price estimate, cancellation policy (≥12h), and **"within 7 days"** reminder.
3. On confirm → App calls /api/Booking:
    - Payload: ownerNIC, stationId, slotStartUtc, slotEndUtc, **idempotencyKey**.
    - Service performs **atomic capacity check** and **creates booking** with Status = Pending.
    - If **auto-approve** is enabled at station/policy level and capacity is still free, service sets Status = Approved immediately.
4. App receives booking and displays **"Reservation Created"** with status badge (Pending/Approved).

Edge Cases handled by service

- **> 7 days**: reject 400 with rule message.
- **< 12h** from now: allow creation if time ≥ now + minimal lead (you didn't require ≥12h to create, only to update/cancel; we'll permit creation any time *within the 7-day window*, but you can optionally enforce a minimum lead, e.g., 1h).
- **Race conditions**: unique constraint + transaction ensures no overbooking.

4.4 Booking Approval & QR Generation

- **Manual Approval Flow** (default):
    - Backoffice or Operator **reviews Pending** bookings in web/app queue.
    - They click **Approve** or **Reject** (with reason).
- **On Approval** the service:
    - Generates a **QR Token**: booking:<BookingId>;owner:<NIC>;ts:<issuedAt>;exp:<start±window>

- Signs it with **HMAC**; returns **PNG QR** payload and short text code.
- Sets Status = Approved, approvedAtUtc = now.

- App updates booking card to show **"Approved"**, with **Download QR** and **Add to Wallet** options.

Security Windows

- **Check-in window**: QR is scannable only from **T-15m to T+30m** relative to slotStartUtc (configurable).
- **Replay protection**: On first successful scan, the token is **consumed**; further scans require **operator override**.

---

5) Station Operator Journey (Mobile + Optional Web)

5.1 Operator Task Inbox

1. Operator logs in → sees **Today's queue** filtered by their stationIds[]:
   - **Upcoming Approvals** (if manual approvals are used)
   - **Arrivals** (bookings in check-in window)
   - **In-Progress** sessions
   - **Exceptions** (no-shows, expired QR, capacity change alerts)

5.2 Check-in via QR Scan

1. Owner arrives, shows QR. Operator taps **Scan**:
   - App decodes QR → sends token to /api/Sessions/checkin (or /api/Booking/checkin).
   - Service validates signature, time window, booking status (Approved), station match.
2. If valid → service sets Status = InProgress and assigns a **connector** if modeled; records startUtc = now.
3. Operator sees **live session card** with **Start Time**, **Connector #**, and controls: **Pause/Stop**, **Notes**.

No-Show Handling

- If QR never scanned and **T+15m** passes: service auto-marks **NoShow** and **releases capacity**.

5.3 Finalize Session

1. When charging completes, operator taps **Finalize**:
    - Enters **metered kWh** (or duration if fixed-time), optional **photos/notes**, **payment method** (if captured).
    - App calls /api/Sessions/finalize:
        - Service sets endUtc = now, calculates **charges** (kWh * tariff + tax), sets **Status = Completed**.
        - Emits receipt (PDF/HTML) and updates **owner's history**.
2. Owner's app receives a push/in-app entry: **"Charging Completed"** with receipt.

Cancellation/Abort On-Site

- If owner cancels within policy (**≥12h**) → allowed and frees capacity.
- Within 12h: **reject** via policy; operator may mark **CanceledByOperator** with fee if policy allows.

---

6) Modifications & Cancellations (Owner)

6.1 Update a Reservation

- From owner's **Upcoming** tab → **Edit** date/time.
- Service checks: **now ≤ slotStartUtc – 12h**.
    - If OK: re-book into new slot atomically (decrement old, increment new) and retain history.
    - If not OK: respond with rule error ("Changes must be at least 12 hours before start").

6.2 Cancel a Reservation

- From booking details → **Cancel**.
- Service checks: **now ≤ slotStartUtc – 12h**.
    - If OK: set Status = CanceledByOwner, **release capacity**.

- If not OK: rule error; show contact operator/backoffice.

---

7) Deactivations & Safety Rails

7.1 Deactivate EV Owner

- Owner can **Self-Deactivate** in the app → local SQLite status updated; service EvOwner.Status = Deactivated.
- Login remains possible but booking actions disabled.
- **Reactivation**: only a **Backoffice** or **System Admin** can set status back to Active via web.

7.2 Deactivate Station

- Backoffice clicks **Deactivate Station**.
- Service **blocks** if **any future Pending/Approved bookings** exist; shows count and dates.
- If none, station Status = Inactive; scheduler stops generating future buckets.

---

8) Data States & Rules (Authoritative on Service)

8.1 Booking State Machine

Pending → Approved → InProgress → Completed
Pending → Rejected
Approved → NoShow (auto after window)
Any non-completed → CanceledByOwner (≥12h) or CanceledByBackoffice/Operator (policy)

8.2 Station State

Active | Inactive | Maintenance

- Inactive blocks future bookings.
- Maintenance reduces capacity; existing bookings may be flagged as conflicts for manual resolution.

8.3 Owner State

Active | Deactivated

- Deactivated prevents create/modify/cancel and hides QR; history remains visible.

8.4 Scheduling & Capacity

- **Slot granularity** (e.g., 60 min).
- **Capacity per slot** = connectors – holds.
- **Atomic reservation** with **idempotency keys** to prevent double-tap bookings.
- Mongo unique guard: { stationId, slotStartUtc, ownerNIC } (if "one booking per owner per slot"), plus a counter with transaction or server-side "capacity decrement" pattern.

---

9) Mobile App — Local SQLite (User Management Only)

- Tables: OwnerProfile (NIC pk, name, email, phone, status, lastSyncAt), Auth (token, expiry, role, userId).
- **Sync Rules**:
  - **Upsert** to server on profile edits.
  - On login, **server profile overwrites local** to avoid drift.
  - If local shows Deactivated, app disables reservation UI and shows reactivation guidance.
- **Offline**:
  - Owner can **view cached profile** and **cached bookings list** (read-only).
  - Actions that change state **queue** but only **execute when online** (client shows "pending sync" badges).

---

10) Permissions Matrix (Essential)

Action	System Admin	Backoffice	Operator	EV Owner
Request Backoffice Account	-	✓ (via signup)	-	-
Approve/Reject Backoffice	✓	–	–	–

Action	System Admin	Backoffice	Operator	EV Owner
Create/Update Station	✓	✓	(view)	–
Set Schedules	✓	✓	(view)	–
Create Operators	✓	✓	–	–
Owner Reactivation	✓	✓	–	–
Booking Approve/Reject	✓	✓	✓	–
Create Booking	–	–	–	✓
Modify/Cancel Booking	–	–	(cancel per policy)	✓
Check-in (QR)	–	–	✓	–
Finalize Session	–	–	✓	–
Deactivate Station	✓	✓	–	–

---

11) Notifications & Comms (Optional but Polished)

- **Email/SMS/Push** for: booking created, approved, reminder at T-24h/T-2h, QR issued, no-show, session completed (receipt).
- **In-app inbox** mirrored for both Owner and Operator.

---

12) Audits, Logs, and Metrics

- **Audit trail** on every state transition: who, when, from→to, reason.
- **Operator actions** linked to operatorId & stationId.
- **Dashboards**:
    - Backoffice Home: Pending approvals, Today's schedule load, No-show rate, Revenue (if applicable).
    - Owner Dashboard: **Counts** — Pending, Approved (future), Past sessions.
- **Search** by NIC, station, date range.

---

13) Service API (Shape — you can map to your controllers)

- POST /api/Auth/login → JWT with role (EvOwner or Operator or web roles)
- `POST /api/Backoffice/request` → submit registration request (status = PendingApproval).
- `PUT /api/Backoffice/{id}/approve` or `/reject` → System Admin decision.
- GET /api/Station?near=lat,lng&radius=km&type=AC|DC
- GET /api/Station/{id} → details + next 7 days availability
- PUT /api/Station/{id} / PUT /api/Station/{id}/deactivate (guarded)
- PUT /api/Station/{id}/schedule (weekly + exceptions)
- POST /api/Operator (backoffice)
- POST /api/EvOwner / PUT /api/EvOwner/{nic} / PUT /api/EvOwner/{nic}/deactivate / PUT /api/EvOwner/{nic}/reactivate
- GET /api/Booking?ownerNic=… (upcoming/history)
- POST /api/Booking (create)
- PUT /api/Booking/{id} (modify; 12h guard)
- PUT /api/Booking/{id}/cancel (12h guard)
- PUT /api/Booking/{id}/approve / reject
- POST /api/Sessions/checkin (QR token)
- POST /api/Sessions/finalize (metering, totals)
- GET /api/Bookings/daily-capacity?stationId=…&date=… (operator view)

All business rules enforced here (fat service). Clients are “thin UI”.

---

14) Pricing & Finalization (If Needed)

- **Tariff models**:
    - **Time-based** (per 30/60 min),
    - **Energy-based** (per kWh), or

    - **Hybrid** (min fee + per kWh).
- Finalize uses measured kWh if available; else duration * rate.
- Taxes/fees configurable per station/business.

---

15) Edge Cases & Reliability

- **Clock skew**: rely on server time for cutoff windows.
- **Idempotency** on booking create (header: Idempotency-Key).
- **Capacity drift**: if schedule changes, service proactively re-checks future bookings and notifies conflicts.
- **Time zones**: server UTC, client display local; all policy checks done in UTC.
- **Security**: QR contains only minimal identifiers + HMAC; never expose PII in plaintext.

---

16) Happy-Path "Day in the Life" (Narrative)

1. Backoffice submits registration request.
2. System Admin approves → Backoffice receives email, sets password, and logs in.
3. **Backoffice** adds Station A (DC, 4 connectors, 08:00–22:00, 60-min slots), sets weekly schedule.
4. They create **Operator Maya**, assign Station A.
5. **Owner Ravi** registers (NIC 2002…), logs in, sees map, picks Station A at **tomorrow 10:00–11:00**.
6. Booking created **Pending**; Maya approves at 09:00 today; QR issued.
7. **Tomorrow 09:50**, Ravi arrives; Maya scans QR → **InProgress** at 10:02.
8. At 10:55, charge done; Maya **finalizes** with 18.2 kWh; receipt generated; status **Completed**.

9. Later, Backoffice tries to **deactivate Station A** for renovation next week; system blocks due to **3 future bookings** — they reschedule/cancel those first, then deactivate.

---

17) What You'll Build (Concise Checklist)

- **Web** (Backoffice/Admin/Operator):
    - Station CRUD + maps, Schedule editor, Operators management
    - Booking review (approve/reject), dashboards, audits
- **Android** (shared login):
    - **EV Owner**: Register (SQLite + sync), Find stations (map), Create/Modify/Cancel (policy), QR wallet, Dashboard counts, History
    - **Operator**: Inbox (today), QR scan, Check-in, Finalize, Exceptions
- **Service** (.NET on IIS, MongoDB):
    - Auth/JWT, owners/operators/backoffice CRUD
    - Scheduling engine & capacity
    - Booking lifecycle + rules (7-day, 12-hour)
    - QR issuance/validation, sessions finalize, receipts
    - Audits, notifications, metrics
    - Geo search (2dsphere), idempotency, UTC enforcement

Web Service (.NET on IIS, MongoDB) — Step-by-Step Implementation Guide (FAT Service)

Below is a practical build plan for your central **C# .NET Web API** hosted on **IIS** with **MongoDB**. It implements all server-side business logic (fat service) for the finalized workflow you supplied.

---

0) Tech Stack & Prereqs

- **.NET 8 (ASP.NET Core Web API)**
- **MongoDB 6+** (enable replica set if you want multi-document transactions; not required with the inventory pattern below)
- **IIS** + ASP.NET Core Hosting Bundle
- NuGet packages:
    - MongoDB.Driver
    - Microsoft.AspNetCore.Authentication.JwtBearer
    - FluentValidation.AspNetCore
    - BCrypt.Net-Next (passwords)
    - System.IdentityModel.Tokens.Jwt
    - (Optional) Quartz or use BackgroundService for schedulers
- Tools: Postman, OpenAPI/Swagger

---

1) Solution Structure

evcs.sln

├─ src/

```
│   ├─ Evcs.Api/            # ASP.NET Core Web API (controllers, DI, middleware)
│   ├─ Evcs.Core/           # Domain models, DTOs, interfaces, validators
│   └─ Evcs.Infrastructure/ # Mongo repositories, services, auth, schedulers
└─ tests/
    └─ Evcs.Api.Tests/
```

---

2) Configuration & Secrets

appsettings.json

```
{
  "Mongo": {
    "ConnectionString": "mongodb://user:pass@localhost:27017",
    "Database": "evcs"
  },
  "Jwt": {
    "Issuer": "evcs.svc",
    "Audience": "evcs.clients",
    "Key": "CHANGE_ME_LONG_RANDOM_SECRET",
    "AccessTokenMinutes": 120
  },
  "Qr": {
    "HmacKey": "CHANGE_ME_DIFFERENT_LONG_RANDOM_SECRET",
    "CheckInWindowMinutesBefore": 15,
    "CheckInWindowMinutesAfter": 30
  },
  "Booking": {
    "MaxDaysAhead": 7,
```

```
  "MinHoursBeforeModifyOrCancel": 12,
  "DefaultSlotMinutes": 60
 },
 "Cors": { "AllowedOrigins": [ "http://localhost:5173", "http://localhost:8080" ] }
}
```

Program.cs (key points)

```
builder.Services.Configure<MongoOptions>(builder.Configuration.GetSection("Mongo"));
builder.Services.AddSingleton<IMongoClient>(_ => new MongoClient(
  builder.Configuration.GetSection("Mongo")["ConnectionString"]));
builder.Services.AddSingleton<IMongoDatabase>(sp =>
  sp.GetRequiredService<IMongoClient>().GetDatabase(
    builder.Configuration.GetSection("Mongo")["Database"]));

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
 .AddJwtBearer(o => { /* issuer, audience, key, validations */ });

builder.Services.AddAuthorization(o =>
{
  o.AddPolicy("SystemAdmin", p => p.RequireRole("SystemAdmin"));
  o.AddPolicy("Backoffice",  p => p.RequireRole("Backoffice"));
  o.AddPolicy("Operator",    p => p.RequireRole("Operator"));
  o.AddPolicy("EvOwner",     p => p.RequireRole("EvOwner"));
});

builder.Services.AddCors(o => o.AddDefaultPolicy(p =>
  p.WithOrigins(builder.Configuration.GetSection("Cors:AllowedOrigins").Get<string[]>())
```

```
    .AllowAnyHeader().AllowAnyMethod().AllowCredentials()));

builder.Services.AddControllers().AddFluentValidation();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

/* Register repositories, services, schedulers */
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IBackofficeService, BackofficeService>();
builder.Services.AddScoped<IStationService, StationService>();
builder.Services.AddScoped<IScheduleService, ScheduleService>();
builder.Services.AddScoped<IBookingService, BookingService>();
builder.Services.AddScoped<ISessionService, SessionService>();
builder.Services.AddScoped<IQrService, QrService>();
builder.Services.AddScoped<IAuditService, AuditService>();
builder.Services.AddScoped<INotificationService, NotificationService>();
builder.Services.AddHostedService<SlotInventoryRefresher>(); // background job

var app = builder.Build();
app.UseCors();
app.UseSwagger(); app.UseSwaggerUI();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
```

app.Run();

---

3) MongoDB Data Model & Indexes

3.1 Collections (minimal fields shown)

- **users** (for auth of all roles)

```
{
  "_id": "ObjectId",
  "email": "string (unique)",
  "passwordHash": "string",
  "role": "SystemAdmin|Backoffice|Operator|EvOwner",
  "backofficeId": "ObjectId|null",
  "operatorStationIds": ["ObjectId"],
  "ownerNIC": "string|null",
  "status": "Active|Deactivated|PendingApproval|Rejected"  // for backoffice requests too
}
```

- **backoffices**

```
{ "_id": "ObjectId", "businessName": "string", "contact": {...}, "timeZone": "string",
  "status": "PendingApproval|Active|Rejected", "createdAtUtc": "Date" }
```

- **stations**

```
{
  "_id": "ObjectId", "backofficeId": "ObjectId", "name": "string",
  "type": "AC|DC", "connectors": 4, "status": "Active|Inactive|Maintenance",
  "location": { "type": "Point", "coordinates": [lng, lat] }, // GeoJSON
  "defaultSlotMinutes": 60, "pricing": {...}, "hours": {...}, "createdAtUtc": "Date"
}
```

- **schedules** (weekly template + exceptions per station)

```
{ "_id": "ObjectId", "stationId": "ObjectId",
  "weekly": { "mon": [{"start":"06:00","end":"22:00"}], ... },
  "exceptions": [ { "date":"2025-10-10", "closed": true } ],
  "capacityOverrides": [ { "date":"2025-10-12", "connectors": 2 } ],
  "updatedAtUtc": "Date"
}
```

- **station_slot_inventory** (capacity per slot; inventory pattern for atomic booking)

```
{
  "_id": "ObjectId",
  "stationId": "ObjectId",
  "slotStartUtc": "Date",
  "slotEndUtc": "Date",
  "capacity": 4,
  "reserved": 0,
  "holds": 0,                // optional
  "updatedAtUtc": "Date"
}
```

- **ev_owners**

```
{ "_id":"ObjectId", "nic":"string (unique)", "name":"string", "email":"string",
  "phone":"string", "status":"Active|Deactivated", "createdAtUtc":"Date" }
```

- **bookings**

```
{
  "_id":"ObjectId", "ownerNIC":"string", "stationId":"ObjectId",

"slotStartUtc":"Date","slotEndUtc":"Date","status":"Pending|Approved|Rejected|InProgress|
Completed|NoShow|CanceledByOwner|CanceledByBackoffice|CanceledByOperator",
```

"approvedAtUtc":"Date|null", "createdAtUtc":"Date",

"qr": { "token":"string", "issuedAtUtc":"Date", "expiresAtUtc":"Date" },

"idempotencyKey":"string|null", "audit":[…]

}

- **sessions**

{ "_id":"ObjectId","bookingId":"ObjectId","stationId":"ObjectId",

"startUtc":"Date","endUtc":"Date|null","kWh":18.2,"total":"decimal", "notes":"string" }

- **audits**

{ "_id":"ObjectId","entity":"Booking|Station|Backoffice|Owner","entityId":"ObjectId|string",

"action":"Create|Approve|Reject|Update|Cancel|Finalize|Deactivate|Reactivate|Checkin|NoShow",

"byUserId":"ObjectId","atUtc":"Date","from":"json","to":"json","reason":"string" }

- **idempotency**

{ "_id":"string (Idempotency-Key)", "endpoint":"string", "requestHash":"string", "response":"json", "createdAtUtc":"Date", "ttlSec": 604800 }

3.2 Indexes

- users.email unique
- ev_owners.nic unique
- stations.location 2dsphere
- station_slot_inventory: {stationId:1, slotStartUtc:1} unique
- bookings: {ownerNIC:1, slotStartUtc:1, stationId:1} (optional)
- TTL index on idempotency.createdAtUtc with expireAfterSeconds
- Useful compound indexes for queries:
    - bookings: {ownerNIC:1, status:1, slotStartUtc:1}
    - bookings: {stationId:1, status:1, slotStartUtc:1}

Index creation: run once at startup.

---

4) Authentication & Authorization

- **JWT** with claims: sub, role, optional backofficeId, operatorStationIds[], ownerNIC
- **Login** (/api/Auth/login): email+password for SystemAdmin|Backoffice|Operator; NIC+password for EvOwner (or email).
- Passwords hashed with **BCrypt**.
- Controllers guarded with [Authorize(Roles="...")] and fine-grained checks (e.g., operator can only touch their stations).

---

5) Business Services (FAT Service)

- **BackofficeService**
  - Create **registration request** → backoffices (status PendingApproval)
  - Admin **approve/reject** (flip status, create login user for backoffice admin)
- **StationService**
  - CRUD stations, **guard deactivation** when future bookings exist
  - Geo search (nearby)
- **ScheduleService**
  - Manage templates/exceptions
  - Generate **slot inventory** (next 14 days) on changes or nightly
- **BookingService**
  - Enforce **7-day window**, **12-hour modify/cancel**
  - **Atomic capacity** using station_slot_inventory (reserved < capacity filter with $inc)
  - **Idempotency** on create
  - Approve/Reject + **QR issue**
- **SessionService**
  - **Check-in** with QR validation (HMAC + time window), set InProgress

    - **Finalize** (kWh, totals), set Completed
- **QrService**
    - Build token payload booking:<id>;owner:<nic>;ts:<issued>;exp:<iso>
    - HMAC-SHA256 sign, base64url encode
    - Validate and parse
- **AuditService / NotificationService**
    - Append standardized events, optional email/SMS/push hooks

---

6) Background Job: Slot Inventory Refresher

Implement a BackgroundService:

- On start and **every 15 minutes** (and on schedule changes), (re)compute station_slot_inventory for **now → now+14 days**
- Capacity = min(station.connectors, capacityOverride) minus maintenance holds
- Ensure unique {stationId, slotStartUtc} upserts

---

7) Endpoints (Shape, Validation & Rules)

7.1 Auth

POST /api/Auth/login

{ "username": "admin@biz.com", "password": "Secret123" }

→ 200

{ "accessToken":"...", "role":"Backoffice", "backofficeId":"...", "operatorStationIds":[] }

7.2 Backoffice Self-Registration & Approval

POST /api/Backoffice/request (anonymous)

{ "businessName":"VoltCo", "contact":{ "name":"Asha", "email":"ops@volt.co", "phone":"+65..." }, "timeZone":"Asia/Singapore" }

→ 202 { "status":"PendingApproval" }

PUT /api/Backoffice/{id}/approve (SystemAdmin)

{ "adminEmail":"asha@volt.co", "temporaryPassword":"Init#123" }

→ 200 { "status":"Active" }

PUT /api/Backoffice/{id}/reject (SystemAdmin)

{ "reason":"Insufficient documents" }

7.3 Stations

POST /api/Station (Backoffice)

{ "name":"Station A","backofficeId":"...","type":"DC","connectors":4,

"location":{"lng":103.851959,"lat":1.29027},

"defaultSlotMinutes":60,"pricing":{"mode":"time","hourly":6.5} }

→ creates; triggers inventory generation

PUT /api/Station/{id}/deactivate (Backoffice)

- **Reject** if any **future** Pending|Approved bookings exist.

GET /api/Station?near=1.29,103.85&radius=5&type=DC (Owner)

- Returns stations with **7-day availability summary** (per day counts)

PUT /api/Station/{id}/schedule (Backoffice)

{ "weekly":{ "mon":[{"start":"06:00","end":"22:00"}], "sun":[{"start":"08:00","end":"20:00"}] },

"exceptions":[{"date":"2025-10-10","closed":true}],

"capacityOverrides":[{"date":"2025-10-12","connectors":2}] }

7.4 Operators

POST /api/Operator (Backoffice)

{ "email":"maya@volt.co","password":"Init#123","stationIds":["...","..."] }

7.5 EV Owners

POST /api/EvOwner (anonymous)

{ "nic":"200212345679","name":"Ravi","email":"ravi@ex.com","phone":"+94...",
"password":"..." }

PUT /api/EvOwner/{nic} (owner)
PUT /api/EvOwner/{nic}/deactivate (owner)
PUT /api/EvOwner/{nic}/reactivate (Backoffice|SystemAdmin)

7.6 Booking

GET /api/Booking?ownerNic=200212345679 (owner)

- Upcoming & history

POST /api/Booking (owner) **(Idempotency-Key header required)**

{ "ownerNIC":"200212345679","stationId":"...","slotStartUtc":"2025-10-03T02:00:00Z","slotEndUtc":"2025-10-03T03:00:00Z" }

Rules enforced server-side:

- slotStartUtc <= now + 7 days
- idempotency replay returns same result
- atomic inventory: findOneAndUpdate({stationId,slotStartUtc,reserved:{$lt:capacity}}, {$inc:{reserved:1}})
- create booking Pending; if station set to **auto-approve** → Approved and QR issued

PUT /api/Booking/{id} (owner)

- Only if now <= slotStartUtc - 12h
- Rebook: decrement old inventory, increment new (two atomic ops with fallback rollback)

PUT /api/Booking/{id}/cancel (owner)

- Only if now <= slotStartUtc - 12h → set CanceledByOwner + reserved--

PUT /api/Booking/{id}/approve / reject (Backoffice|Operator)

- On approve → create QR:

{ "qr": { "token":"base64url(hdr.payload.sig)", "issuedAtUtc":"...", "expiresAtUtc":"..." } }

7.7 Sessions (Operator)

POST /api/Sessions/checkin

{ "qr":"<token from booking>" }

Server:

- Validate HMAC, **time window** (T-15 to T+30 mins), booking status Approved, station/operator permission
- Set booking InProgress, create session with startUtc

POST /api/Sessions/finalize

{ "bookingId":"...", "kWh":18.2, "notes":"N/A", "payment":{"mode":"cash","amount":12.50} }

- Set endUtc=now, compute totals (tariff), set Completed, emit receipt

7.8 Operator View

GET /api/Bookings/daily-capacity?stationId=...&date=2025-10-03

- Returns per-slot capacity and reserved

---

8) Core Logic Snippets

8.1 Atomic Reserve (Inventory Pattern)

```
var filter = Builders<SlotInv>.Filter.And(
   Builders<SlotInv>.Filter.Eq(x => x.StationId, stationId),
   Builders<SlotInv>.Filter.Eq(x => x.SlotStartUtc, slotStartUtc),
   Builders<SlotInv>.Filter.Expr(x => x.Reserved < x.Capacity)
);
var update = Builders<SlotInv>.Update
   .Inc(x => x.Reserved, 1)
   .Set(x => x.UpdatedAtUtc, DateTime.UtcNow);


var updated = await _invCol.FindOneAndUpdateAsync(filter, update);
if (updated == null) throw new BusinessRuleException("No capacity available");
```

8.2 Idempotency

- Read header Idempotency-Key.

- If exists in idempotency → return stored response.
- Else process, store { key, endpoint, requestHash, response } then return.

8.3 QR Token (HMAC)

```
public string CreateQrToken(string bookingId, string nic, DateTime start, TimeSpan before, TimeSpan after)

{

  var issued = DateTime.UtcNow;

  var exp = start.Add(after);

  var payload = $"booking:{bookingId};owner:{nic};ts:{issued:O};exp:{exp:O}";

  var sig = HmacSha256(payload, _qrOptions.HmacKey);

  return Base64UrlEncode($"{payload}|sig:{sig}");

}
```

Validation:

- Decode → split payload/signature
- Recompute HMAC; must match
- Ensure now >= start - before and now <= start + after

8.4 Deactivate Station Guard

```
var anyFuture = await _bookings.CountDocumentsAsync(b =>

  b.StationId == stationId &&

  (b.Status == "Pending" || b.Status == "Approved") &&

  b.SlotStartUtc > DateTime.UtcNow);


if (anyFuture > 0) throw new BusinessRuleException("Station has future bookings");
```

---

9) Validation & Error Handling

- Use **FluentValidation** per DTO (dates, ranges, NIC format).

- Standard error shape:

{ "error": "BusinessRuleViolation", "message": "Changes must be >= 12 hours before start" }

- Global exception middleware mapping:
    - BusinessRuleException → 400
    - UnauthorizedAccessException → 401
    - ForbiddenException → 403
    - Not found → 404

---

10) Time & Timezones

- **All stored in UTC**.
- Compute windows using DateTime.UtcNow.
- Client displays local time.
- When parsing schedules (HH:mm), convert to UTC using station **timeZone** if needed when generating slots.

---

11) Auditing & Notifications

- After **every** state change, write an **audit** record.
- Hook notification service (email/SMS/push) on:
    - Booking created/approved/rejected
    - Reminder T-24h/T-2h
    - QR issued
    - No-show
    - Session completed (receipt)

---

12) IIS Deployment Steps

1. **Publish**: dotnet publish Evcs.Api -c Release -o .\publish

2. Install **ASP.NET Core Hosting Bundle** on server.
3. In **IIS Manager**:
    - Add Website → physical path to publish
    - Set **App Pool**: No Managed Code, AlwaysRunning
    - Configure **Environment Variables** (optional) for secrets
4. **URL Rewrite** (optional), **HTTPS** binding with cert
5. Ensure MongoDB is reachable from server
6. Set **CORS** origins to web/mobile domains
7. Confirm **/swagger** loads, then smoke-test endpoints

---

13) Build Order (Practical Sprint Plan)

1. **Bootstrap** API + Mongo connection + indexes
2. **Auth** (users, roles, JWT, login) + seed a SystemAdmin
3. **Backoffice**: request → approve/reject → create backoffice admin user
4. **Stations** + **Schedules** + inventory generator (background service)
5. **EV Owner** CRUD (NIC PK) + status
6. **Booking**: create (idempotent + inventory), list, modify, cancel, approve/reject + QR
7. **Sessions**: check-in (QR), finalize (kWh, totals)
8. **Guards**: station deactivate rule, 7-day, 12-hour policies
9. **Audits/Notifications**
10. **Polish**: error shapes, Swagger examples, CORS, rate limiting (optional)

---

14) Postman Test Flow (Essentials)

1. **Auth**: login as SystemAdmin, Backoffice, Operator, EvOwner
2. **Backoffice**: POST /Backoffice/request → approve as admin
3. **Station**: create + schedule → confirm station_slot_inventory populated

4. **Owner**: register → login
5. **Search**: nearby stations → view slots
6. **Booking**: create (Idempotency-Key), list; modify/cancel boundary tests
7. **Approval**: approve → QR issued
8. **Session**: checkin with QR → finalize
9. **Deactivate Station**: verify guard

---

15) Security Notes

- Rotate **JWT** and **QR HMAC** secrets periodically (support dual keys during rotation).
- Do not embed PII in QR beyond NIC if required; prefer ownerId hash.
- Rate-limit sensitive endpoints (/Auth/login, /Booking).
- Always compute rule checks on server time.

---

16) What Clients Need (Connection)

- **Base URL**: https://api.yourdomain.com/
- **Auth**: Authorization: Bearer <JWT>
- **Idempotency** (booking create): Idempotency-Key: <uuid>
- **CORS** allows your web origin(s); Android uses full URL.