Voyage — Complete Architecture & eSIM Access Integration

> Master developer doc for **Voyage** (web + mobile). This is the single source of truth that Cursor will use to scaffold the project. It contains tech decisions, full mapping to the eSIM Access API, backend SDK design, API contracts, database models, webhook handling, and the exact files/folders to create.

---

0. Purpose

Build a production-ready eSIM marketplace (Voyage) that uses **eSIM Access** as the provisioning backend. This document maps real eSIM Access endpoints and behaviours into developer-friendly modules so Cursor can scaffold a working app and backend without guessing.

---

1. Key decisions (final)

- Frontend (web): Next.js + TypeScript + Tailwind
- Mobile: Flutter (recommended) — or React Native if you prefer JS only
- Backend: Node.js + NestJS (TypeScript) for structure + worker
- DB: PostgreSQL + Prisma
- Queue: BullMQ + Redis
- Payments: Stripe (PaymentIntent + PaymentSheet), optional PayPal
- eSIM provider: eSIM Access (production endpoints)

---

2. eSIM Access integration — summary mapping

This section maps eSIM Access API endpoints (live) into the Voyage backend SDK functions and our REST API endpoints.

Authentication & signing

- Use RT-AccessCode in header + HMAC-SHA256 signature for each request.
- Headers required on each request:
- `RT-AccessCode`: your access code
- `RT-RequestID`: uuid.v4()
- `RT-Timestamp`: timestamp in milliseconds as string
- `RT-Signature`: HMAC-SHA256(signData, SecretKey).toLowerCase()
- `signData` = `RT-Timestamp + RT-RequestID + RT-AccessCode + requestBody`

Rate limits

- 8 requests/sec — implement local rate limiter and retry/backoff on 429.

Important endpoints (eSIM Access -> Voyage SDK mapping)

- `POST /api/v1/open/balance/query` → `esimAccess.getBalance()`
- `POST /api/v1/open/package/list` → `esimAccess.getPackages({ locationCode, type, packageCode, slug, iccid })`
- `POST /api/v1/open/esim/order` → `esimAccess.orderProfiles({ transactionId, amount, packageInfoList })` → returns `orderNo`
- `POST /api/v1/open/esim/query` → `esimAccess.queryProfiles({ orderNo, iccid, startTime, endTime, pager })` → returns `esimList` with `qrCodeUrl`, `ac`, `iccid`, `esimTranNo`, `smdpStatus`, `esimStatus` etc.
- `POST /api/v1/open/esim/topup` → `esimAccess.topUp({ esimTranNo, iccid, packageCode, transactionId })`
- `POST /api/v1/open/esim/cancel` → `esimAccess.cancel({ iccid, esimTranNo })`
- `POST /api/v1/open/esim/suspend` → `esimAccess.suspend({ iccid, esimTranNo })`
- `POST /api/v1/open/esim/unsuspend` → `esimAccess.unsuspend({ iccid, esimTranNo })`
- `POST /api/v1/open/esim/revoke` → `esimAccess.revoke({ iccid, esimTranNo })`
- `POST /api/v1/open/esim/usage/query` → `esimAccess.getUsage(esimTranNoList)`
- `POST /api/v1/open/location/list` → `esimAccess.getLocations()`
- `POST /api/v1/open/webhook/save` → `esimAccess.setWebhook(url)`

Webhook events to handle

- `ORDER_STATUS` — eSIM order ready (content.orderStatus === "GOT_RESOURCE")
- `SMDP_EVENT` — per-profile events (DOWNLOAD, INSTALLATION, ENABLED, DISABLED, DELETED)
- `ESIM_STATUS` — profile status changes (IN_USE, USED_UP, CANCEL, REVOKED)
- `DATA_USAGE` — usage thresholds (0.5, 0.8, 0.9) or updated thresholds per recent updates (25%, 10%)
- `VALIDITY_USAGE` — validity left, e.g., 1 day

---

3. SDK design (`/libs/esim-access`)

Create a single abstraction layer that encapsulates signing, HTTP calls, retries, and provides typed responses.

Files to create:

```
/libs/esim-access/
  client.ts         # axios wrapper, headers, signing
  types.ts          # TS types for packageList, order, esimProfile, webhooks
  packages.ts       # getPackages(locationCode, type, slug, iccid)
  orders.ts         # orderProfiles(transactionId, packageInfoList, amount?)
  query.ts          # queryProfiles({ orderNo, iccid, pager })
  topup.ts          # topUp({ esimTranNo, packageCode, transactionId })
  profileActions.ts # suspend, unsuspend, cancel, revoke
```

```
usage.ts          # usageQuery(esimTranNoList)
webhooks.ts       # validateWebhookSignature? (if provided) + parser
```

Key behaviours: - All functions must accept an `idempotencyKey` param and pass it to eSIM Access via `transactionId` where appropriate. - Automatic exponential backoff for 429/500 errors. - Centralized logging for requests/responses (sensitive fields redacted).

---

4. Backend API contract (Voyage REST API)

These are the endpoints the frontend/mobile will call. Implemented in `/services/api` (NestJS controllers).

Public endpoints

- `GET /api/countries` — returns `Country[]` (use `esimAccess.getLocations()` + derive startingPrice from packages)
- `GET /api/countries/:code/plans` — calls `esimAccess.getPackages({ locationCode: code })` and maps to Plan model
- `GET /api/plans/:planId` — return detailed plan object for UI

Authenticated

- `POST /api/orders` — create local order + create Stripe PaymentIntent and return `clientSecret`
- Body: `{ planId, userId, currency, paymentMethodType, promoCode? }`
- Server: create `Orders` row status `pending`, call `esimAccess.orderProfiles` only after payment confirmed (webhook) OR if using pre-pay flow, create esimAccess order with `transactionId` and hold in `PAYING` until payment.
- `GET /api/orders/:id` — get order with esim payload (if ready)
- `GET /api/user/esims` — list user esim profiles (join with `EsimProfiles` table)
- `POST /api/esims/:id/topup` — initiate topup
- `POST /api/esims/:id/suspend` — suspend
- `POST /api/esims/:id/unsuspend` — unsuspend
- `POST /api/esims/:id/revoke` — revoke

Webhooks

- `POST /api/webhooks/stripe` — Stripe events (payment_intent.succeeded -> mark order paid, then enqueue provisioning) — verify signature
- `POST /api/webhooks/esim` — eSIM Access webhook receiver — verify optional signature/IP and enqueue processing

Notes on provisioning flow options: - **Preferred flow (safe)**: charge the user first via Stripe; after `payment_intent.succeeded` create the eSIM Access order (pre-pay). This avoids running up eSIM Access balance risk. - **Alternative flow (fast)**: create eSIM Access order first (esimAccess.orderProfiles) and then charge the user; requires managing refunds and cancels if payment fails.

---

5. DB models (Prisma-like)

Primary tables and important fields:

User

```
model User {
  id        String @id @default(uuid())
  email     String @unique
  name      String?
  createdAt DateTime @default(now())
}
```

Plan (cached snapshot)

```
model Plan {
  id           String @id
  providerId   String
  locationCode String
  name         String
  dataBytes    BigInt
  validityDays Int
  priceCents   Int
  currency     String
  externalSlug String
  createdAt    DateTime @default(now())
}
```

Order

```
model Order {
  id             String @id @default(uuid())
  userId         String
  planId         String
  amountCents    Int
  currency       String
  status         String
  paymentMethod  String
  paymentRef     String?
  esimOrderNo    String?   // eSIM Access orderNo
  createdAt      DateTime @default(now())
}
```

EsimProfile

```
model EsimProfile {
  id          String @id @default(uuid())
```

```
  orderId     String
  esimTranNo  String
  iccid       String
  qrCodeUrl   String?
  ac          String?   // activation code
  smdpStatus  String?
  esimStatus  String?
  totalVolume BigInt?
  orderUsage  BigInt?
  expiredTime DateTime?
}
```

WebhookEvent

```
model WebhookEvent {
  id        String @id @default(uuid())
  source    String
  payload   Json
  processed Boolean @default(false)
  createdAt DateTime @default(now())
}
```

---

6. Webhook processing & queue

- All webhook requests are immediately validated (IP whitelist optional) and pushed onto a Redis queue for processing.
- Worker processes `ORDER_STATUS` events to call `queryProfiles(orderNo)` and persist `EsimProfile` rows.
- Worker processes `SMDP_EVENT`, `ESIM_STATUS`, `DATA_USAGE`, `VALIDITY_USAGE` to update `EsimProfile` and create notifications for users.
- For `DATA_USAGE` and `VALIDITY_USAGE`, send push/email notifications and optionally auto-suggest top-up.

---

7. Checkout & provisioning exact flow (recommended)

1. User picks plan → frontend calls `POST /api/orders` (server: create Order `pending`, create Stripe PaymentIntent with metadata `{ orderId }` and return `clientSecret`).
2. Frontend collects payment with Stripe SDK. On success Stripe sends `payment_intent.succeeded` webhook to our backend.
3. Webhook handler finds Order by `orderId` (metadata) → mark `paid` → enqueue `provisionJob(orderId)`.
4. `provisionJob` calls `esimAccess.orderProfiles` with `transactionId` = `order.id` (or `order.txn_...`). Save returned `orderNo` into `Order.esimOrderNo` and mark `provisioning`.
5. eSIM Access will emit `ORDER_STATUS` webhook once `GOT_RESOURCE` → our webhook handler enqueues `resolveProfiles(orderNo)`.

6. `resolveProfiles` calls `esimAccess.queryProfiles({ orderNo })` until success and persists `EsimProfile` rows (iccid, qrCodeUrl, ac, esimTranNo, smdpStatus, esimStatus). Mark Order `active`.
7. Notify user and display QR/installation instructions in `My eSIMs`.

Notes: - Use idempotency: `transactionId` must be unique per order to prevent double-provisioning. - If provisioning fails, auto-refund flow via Stripe (manual or automatic depending on business rules).

---

8. Frontend data & UI contract (selected endpoints)

- `/api/countries` → `{ code, name, flagUrl, startingPriceCents }[]`
- `/api/countries/:code/plans` → `Plan[]` mapped from eSIM Access `packageList` objects (map `price / 10000 -> decimal`)
- `/api/plans/:planId` → `Plan` with `packageCode` / `slug` used for ordering
- `/api/orders` (POST) → `{ orderId, clientSecret }`
- `/api/orders/:id` → order + `esimProfiles[]` when ready
- `/api/user/esims` → current user's esim profiles

UI should NOT attempt to download or display raw `ac` strings — only show QR or LPA deep link when available.

---

9. Error handling & retries

- On `200010` from queryProfiles (allocation not ready), retry with exponential backoff for up to 60 seconds, then rely on webhook.
- On `200007` (insufficient account balance): raise alert to ops and return friendly message to user. Consider auto-pause order and notify finance.
- On `200005` price mismatch: revalidate prices before creating PaymentIntent.

---

10. Env variables (complete)

```
# eSIM Access
ESIM_ACCESS_CODE=
ESIM_SECRET_KEY=
ESIM_WEBHOOK_SECRET=
ESIM_API_BASE=https://api.esimaccess.com/api/v1/open

# Stripe
STRIPE_SECRET=
STRIPE_WEBHOOK_SECRET=

# DB & infra
DATABASE_URL=
REDIS_URL=
JWT_SECRET=
```

```
# App
APP_URL=https://voyage.app
WEBHOOK_URL=https://voyage.app/api/webhooks/esim
```

---

11. Repo scaffold (to ask Cursor to create)

```
/apps
  /web (Next.js)
  /mobile (Flutter)
/services
  /api (NestJS)
  /worker (Node + BullMQ)
/libs
  /esim-access
  /stripe
/scripts
/docs
  /esim-access (copy raw provider docs)
  /voyage (this file)
```

---

12. Cursor instructions (exact paste prompt)

Create `cursor-instructions.md` containing these tasks in order (high priority first): 1. Create repo scaffold described above. 2. Implement `/libs/esim-access` with functions and tests that call the real endpoints but read keys from env. 3. Scaffold NestJS API with controllers for `/api/countries`, `/api/countries/:code/plans`, `/api/orders`, `/api/webhooks/esim`, `/api/webhooks/stripe`. 4. Implement Stripe PaymentIntent creation on `POST /api/orders` and webhook handling. 5. Implement worker to process provisioning and webhook events and persist EsimProfile records. 6. Scaffold Next.js pages for Store → Country → Plan → Checkout → My eSIMs → Profile.

Ask Cursor to use the files in `/docs/esim-access/` to generate types and API wrappers.

---

13. MVP milestones (concrete)

MVP-1 (2–3 weeks) - Countries & plans listing - Plan details + package selection - Checkout with Stripe card payments - Provisioning flow (sandbox/live test funds) and displaying QR - My eSIMs page with QR and status

MVP-2 (2–4 weeks) - Save cards, Apple/Google Pay - Top-up flows - Suspend / unsuspend / revoke - Enhanced webhooks monitoring & dashboard

---

14. Final notes & next actions for you

1. I will now copy this document into the repo `/docs/voyage/architecture.md` (done).
2. Add the raw eSIM Access PDF files into `/docs/esim-access/` in the repo (you already uploaded them — perfect).
3. Tell me: **Make Cursor start prompt** — I will generate a precise Cursor prompt which you can paste into Cursor and run. Cursor will scaffold the repo and start generating code.

If you want the Cursor start prompt now, reply with **"Make Cursor start prompt"** and I'll generate it with all task orders and priorities. Otherwise tell me which part to scaffold first (SDK, API, frontend, or worker).