JANZEER BLOCKCHAIN

دليل المطوّر

تشغيل عقدة · واجهة REST · توقيع المعاملات

إصدار الوثيقة 1.0 · شبكة جنزير · 2026

دليل مطوّري Janzeer

هذا الدليل موجّه لمطوّري التطبيقات والمتكاملين الذين يبنون على **Janzeer**، وهي طبقة أولى (Layer-1) ذات معروض ثابت ومتحمّلة للأعطال البيزنطية (BFT) وقابلة للانضمام دون إذن ومن **دون حصة/ستيكينغ (stake-free)**، مع مخطّط تشفير مخصّص لمحفظة هرمية حتمية (HD) غير حافظة للمفاتيح. يوضّح لك كيفية تشغيل عقدة، واستهلاك الواجهة البرمجية REST، والأهم — كيفية بناء المعاملات وتوقيعها وإرسالها بالكامل من جهة العميل، لأن العقدة لا تحتفظ أبدًا بمفاتيح المستخدمين. وهو دليل عملي قائم على الأمثلة؛ أما عن التخطيطات البايتية الدقيقة والثوابت فيحيلك إلى الورقة الصفراء بدلًا من تكرارها.

---

1. البدء السريع

1.1 تشغيل عقدة

العقدة **تنزيل عام بصيغة تنفيذية أصلية** (مغلقة المصدر — لا يوجد ملف JAR ولا شيء يُبنى). نزِّل حزمة منصّتك من صفحة **المُصادِقون** في janzeer.org، وتحقَّق من بصمة SHA-256، ثم فكّ الضغط وشغِّلها (`./start.sh`، أو `start.cmd` على Windows) — والتفاصيل في دليل المشغِّل §2. تتابع العقدة حديثة التشغيل السلسلة في وضع **المراقبة فقط**، وهو كل ما يحتاجه التطبيق:

الخدمة	الافتراضي	ملاحظات
واجهة REST	`http://localhost:7019/api/v1`	غلاف الاستجابة `{version, payload, timestamp}`
JSON-RPC 2.0	`POST http://localhost:7019/rpc` · WebSocket `ws://localhost:7019/rpc/ws`	دوالّ `janzeer_*` وتغذيات حيّة للكتل والمعاملات (§2.14)
المستكشف + محفظة الويب	`http://localhost:7019/`	تقدّمهما العقدة نفسها
P2P	`9199` (TCP)	اتصالات صادرة فقط لعقدة المراقبة

يقرأ المُشغِّل المتغيرات `JANZEER_REST_PORT` و`JANZEER_P2P_PORT` و`JANZEER_DATA_DIR` و`JANZEER_ROOT_NODES` من البيئة. للتحقق من العقدة:

```
curl http://localhost:7019/api/v1/info          # networkId, genesisHash, version, syncStatus
curl http://localhost:7019/actuator/health
```

انتظر حتى تصبح `"syncStatus":"SYNCHRONIZED"` قبل الاعتماد على الأرصدة أو إرسال المعاملات. ولا يلزمك تشغيل عقدة لبناء تطبيق: فنقاط النهاية العامة للشبكة (`https://explorer.<domain>/api/v1` و`rpc/`) تتحدث الواجهة نفسها.

1.2 الشبكات

هناك شبكة عامة واحدة هي الشبكة الرئيسية. وتنتمي كل حزمة إلى معرِّف شبكة واحد فقط:

الشبكة	`networkId`	الاستخدام
الشبكة الرئيسية	`janzeer`	السلسلة الحيّة — الشبكة العامة الوحيدة
شبكة اختبار خاصة	`janzeer-testnet`	شبكة تشغّلها بنفسك (شبكة المطوّرين المحلية في عدة التوافق الخاصة بالـ SDK، أو عقدك الخاصة)؛ عملاتها بلا قيمة. لا توجد شبكة اختبار عامة.

يدخل `networkId` في البيانات الموقَّعة لكل معاملة (§3) وفي مصافحة P2P، فالمعاملة الموقَّعة لشبكة غير صالحة في الأخرى. اقرأه من `GET /api/v1/info` بدل تثبيته في الشيفرة، كما تفعل المحافظ الرسمية.

1.3 عملات الاختبار

لا يوجد صنبور عام. للتطوير على سلسلة بعملات مجانية شغّل شبكة خاصة: يشغّل `e2e/node-up.sh` في عدة التوافق الخاصة بالـ SDK شبكة محلية من أربع عقد تشغّل عقدتها الأولى صنبورًا (`POST /api/v1/faucet` مع `{"…address": "0x"}`، بحدّ للمعدّل لكل عنوان ولكل IP) وتموّل محفظة اختبار. ترفض العقدة الصنبور على معرِّف الشبكة الرئيسية. وعلى الشبكة الرئيسية تأتي العملات من تحويل، أو من دفعة لعبة jPong الشهرية، أو من مكافآت الكتل بوصفك مدقِّقًا.

1.4 إعداد التكوين (genesis) وتثبيت مواصفات السلسلة

حالة التكوين (genesis) **مبنية على الإعدادات**. المفاتيح ذات الصلة هي:

المفتاح	الغرض
`janzeer.genesis.anchors`	مُصادِقو المرتكزات (anchor) لبدء التشغيل
`janzeer.genesis.signer-secret`	مفتاح المراسم الذي يوقّع فقط كتلتي بدء التشغيل (وليس مرتكزًا)
`janzeer.genesis.founder`	عنوان عهدة المؤسِّس الاختياري (مُقتطع من الخزينة)
`janzeer.genesis.distribution`	عنوان عهدة مجمّع التوزيع/المنظومة الاختياري
`janzeer.dev-fund`	تمويل تطوير مسبق اختياري لشبكة خاصة بالصيغة `0xaddr:amount`، يُسحب من الخزينة (مرفوض على الشبكة الرئيسية)
`consensus.network-id`	سلسلة معرّف الشبكة (الافتراضي `janzeer`)، المربوطة داخل النص الأصلي الموقّع ومصافحة P2P

مجموعة معاملات التوافق الحرجة (كل ما تحت `*.consensus`) هي مجموعة القواعد المعيارية المثبَّتة عند التكوين (genesis). تحسب كل عقدة **بصمة مواصفات السلسلة** (double-SHA256 مزدوجة فوق تلك المعاملات) وتتبادلها في مصافحة P2P؛ ويُرفض أي نظير تختلف بصمته قبل التحدّي (challenge). والعقدة التي تعدّل أي معامل توافق تفرِّع نفسها عن الشبكة بدلًا من التباعد الصامت — فيمكن للمتكاملين الاعتماد على مجموعة قواعد متّسقة عبر النظراء المُوثَّقين.

---

2. مرجع الواجهة البرمجية REST

2.1 الاصطلاحات

- **المسار الأساسي:** `/api/v1/` على المنفذ `7019`. جميع الأساليب `GET` ما لم يُذكر خلاف ذلك.
- **مغلّف الاستجابة:** كل استجابة تُغلَّف على النحو التالي:

```
{
  "timestamp": 1719700000000,
  "version": "1.1.0",
  "payload": { }
}
```

تكون النتيجة الفعلية دائمًا في `payload`. ويُحذف المغلّف فقط في حالة `/**docs/` و`swagger-resources/` و`/**actuator/`.

- **الأخطاء:** تُعاد على النحو التالي

```
{
  "status": 404,
  "message": "not found",
  "type": "SomeErrorType"
}
```

(الحقل `type` اختياري.) إخفاقات التحقّق ← `400`؛ الموارد المفقودة ← `404`.

- **الترقيم الصفحي:** تقبل نقاط النهاية الخاصة بالقوائم معاملات الاستعلام التالية:

المعامل	المدى / الافتراضي	المعنى
`page`	default `0`, `0 ≤`	فهرس الصفحة المبدوء من الصفر
`size`	default `10`, `1–100`	حجم الصفحة
`sortBy`	field name	حقل الفرز
`sortDirection`	default `DESC`	`ASC` or `DESC`

تبدو الحمولة المرقّمة صفحيًا كالتالي:

```
{
  "total": 42,
  "list": [ ],
  "page": 0,
  "pageSize": 10,
  "totalPages": 5
}
```

تُعاد المبالغ كسلاسل عشرية/`BigDecimal` بمقياس **8 منازل عشرية** (راجع الورقة الصفراء بشأن تحجيم المبالغ).

2.2 المحافظ — `api/v1/wallets/`

الأسلوب	المسار	يُعيد
GET	`wallets/{address}/`	الرصيد المؤكَّد (`BigDecimal`)، المؤكَّد – غير المؤكَّد، بحدٍّ أدنى عند `0`
GET	`wallets/{address}/nonce/`	الرقم التسلسلي (nonce) التالي المتوقَّع للعنوان
POST	`wallets/validateAddress/`	التحقّق من صحّة سلسلة عنوان

```
curl
http://localhost:7019/api/v1/wallets/0x06e1c0fa9955a700876f8cb0acc7f13fba9fb8ba/nonce
```

```
{ "timestamp": 1719700000000, "version": "1.1.0", "payload": 0 }
```

2.3 التحويلات — `api/v1/transactions/transfers/`

الأسلوب	المسار	الاستعلام / المتن	يُعيد
GET	`transactions/transfers/`	`?address=&unconfirmed=` + pagination	تحويلات مرقّمة صفحيًا
GET	`transactions/transfers/{hash}/`	—	تحويل واحد بحسب تجزئة (hash) المعاملة
POST	`transactions/transfers/`	tx request body	يرسل تحويلًا ← `201`

2.4 المُصادِقون (معاملة التسجيل) — `api/v1/transactions/validators/`

الأسلوب	المسار	يُعيد
GET	`transactions/validators/` (+ pagination)	معاملات تسجيل مُصادِقين مرقّمة صفحيًا
GET	`transactions/validators/{hash}/`	معاملة مُصادِق واحدة
POST	`transactions/validators/`	التسجيل كمُصادِق ← `201`

تسجّل العقدة كمُصادِق. الرسوم هي بالضبط **3**؛ وتذهب وديعة التسجيل (غير قابلة للاسترداد) قدرها **2000** بالضبط إلى الخزينة (bank). ويجب أن يكون `validatorKey` فريدًا. ولا يوجد تصويت أو موافقة أو فترة قفل — فالقبول دون إذن، والوديعة إلى جانب مقعد واحد لكل مفتاح هما الحارس ضد هجمات Sybil. ملاحظة: تحتفظ أسماء أنواع (TYPE) المعاملات/الحالة الداخلية، ومفاتيح الإعداد `promoters-count` و`promoter-tx/*`، وجداول قاعدة البيانات بالاسم القديم **promoter** — أما نقاط النهاية وحقول JSON فهي validator (مُصادِق).

2.5 الخروج من المُصادِقين (معاملة التقاعد) — `api/v1/transactions/exit-validators/`

الأسلوب	المسار	يُعيد
GET	`transactions/exit-validators/` (+ pagination)	معاملات خروج مرقّمة صفحيًا
GET	`transactions/exit-validators/{hash}/`	معاملة خروج واحدة
POST	`transactions/exit-validators/`	تقاعد مُصادِق بأناقة ← `201`

تزيل معاملة ExitPromoterTx المُصادِق من المجموعة النشطة ومن جدول الفتحات (slots) عند حدود الحقبة التالية. الرسوم ≤ `0.01`؛ ولا تحمل الحمولة سوى `validatorKey`. ولا تُعاد وديعة التسجيل — فهي تظل غير قابلة للاسترداد بحسب التصميم.

2.6 المكافآت — `api/v1/transactions/rewards/` (للقراءة فقط)

الأسلوب	المسار	يُعيد
GET	`transactions/rewards/` (+ pagination)	معاملات مكافأة نظامية مرقّمة صفحيًا
GET	`transactions/rewards/{hash}/`	معاملة مكافأة واحدة

معاملة RewardTx مولّدة من النظام (الخزينة ← المُصادِق المنتِج، `%100`، الرسوم `0`، الرقم التسلسلي `0`)، **ولا** يمكن للمستخدم إرسالها.

2.7 الكتل — `api/v1/blocks/main/` و `api/v1/blocks/genesis/`

يتشارك الموردان الشكل ذاته:

الأسلوب	المسار	يُعيد
GET	`blocks/main/{hash}/`	كتلة رئيسية بحسب التجزئة (hash)
GET	`blocks/main/{hash}/previous/`	الكتلة السابقة
GET	`blocks/main/{hash}/next/`	الكتلة التالية
GET	`blocks/main/` (+ pagination)	كتل رئيسية مرقّمة صفحيًا

استبدل `main` بـ `genesis` لكتل التكوين (genesis) (كتل حدود الحقبة التي تجمّد مجموعة المُصادِقين النشطة).

2.8 المُصادِقون (الحالة) — `api/v1/validators/`

الأسلوب	المسار	يُعيد
GET	`validators/` (+ pagination, by registration order)	جميع المُصادِقين، مرتّبين بحسب ترتيب التسجيل؛ يحمل كل صف `status` (`ACTIVE` · `PENDING` مسجّل بعد بداية هذه الحقبة ويُعيَّن عند الحدود في مجموعة هذه الحقبة · `STANDBY` مسجّل قبلها وغير معيّن في هذه الحقبة · `BANNED` · `EXITED`)، و`registeredAt`، و`lastBlockTime` / `lastBlockHeight` (أحدث كتلة أنتجها، أو null)، و`joinsAtHeight` (لحالتي PENDING/STANDBY: ارتفاع نشأة الحقبة التالية) — العقدة 0.1.3
GET	`validators/active/` (+ pagination)	المجموعة النشطة الحالية
GET	`validators/view` (+ pagination)	سجلّات عرض المُصادِقين

الانتخاب يجري بحسب **ترتيب التسجيل** (الأقدم تسجيلًا أولًا، بحسب `createDate` ثم المعرّف)، لا بحسب الأصوات أو التقييم — ولا يوجد تفويض.

2.9 المستكشف — `api/v1/explorer/info/`

```
curl http://localhost:7019/api/v1/explorer/info
```

حقول الحمولة:

الحقل	المعنى
`nodesCount`	العقد المعروفة
`blocksCount`	إجمالي الكتل
`transactionsCount`	إجمالي المعاملات
`blockProductionTime`	متوسّط زمن الكتلة
`transactionsPerSecond`	TPS
`epochNumber`	فهرس الحقبة الحالية
`epochDate`	بداية الحقبة الحالية
`validatorsCount`	عدد المُصادِقين
`maxSupply`	`100,000,000`
`circulatingSupply`	`maxSupply` ناقص الخزينة (حتى العقدة 0.1.13)؛ ومن 0.1.14 ناقص كل حساب يحتفظ به البروتوكول أو المشروع — الرقم نفسه في `supply/circulating/`

المعروض — `api/v1/supply/` (من العقدة 0.1.14). `GET /supply` يعيد `{ total, circulating, nonCirculating: [{label, address, balance}] }` داخل الظرف المعتاد؛ و`GET /supply/total` و`GET /supply/circulating` يعيدان **رقمًا واحدًا مجرّدًا بصيغة `text/plain`** دون ظرف (الصيغة التي تستعلم بها مواقع بيانات

السوق). المتداول = الكلي ناقص الأرصدة الحالية للخزينة ولحسابَي المؤسِّس والمنظومة في التكوين ولمحافظ المُصادِقين المؤسِّسين وللمحافظ الأخرى التي يحتفظ بها المشروع والمسجّلة في العقدة لهذه السلسلة (التسميات `treasury` و `founder` و `ecosystem` و `validator` و `project`). انظر دليل الربط القسم 7.

2.10 معلومات العقدة — `api/v1/info/`

الأسلوب	المسار	يُعيد
GET	`info/`	`{ nodeKey, host, port, networkId, genesisHash, chainSpecDigest, version, apiVersion, protocolVersion, syncStatus, faucet }` — اقرأ `networkId` ووقِّع كل معاملة به
GET	`info/version/`	`{ version, apiVersion, protocolVersion }` (البناء، والواجهة البرمجية، وبروتوكول الشبكة — كما في `info/`)
GET	`info/update/`	إصدار العقدة مقابل أحدث إصدار منشور (فحص يومي للبيان، إشعار فقط؛ العقدة 0.1.3)
GET	`info/sync/`	تقدّم التزامن: `syncStatus` و `tipHeight` و `stagedBlocks` (كتل منزّلة ومتحقق منها لم تُثبَّت بعد) و `targetHeight` و `peers` — العقدة الجديدة تجهّز حقبة كاملة قبل أول تثبيت (العقدة 0.1.2)
GET	`info/uptime/`	مدة تشغيل العقدة
GET	`info/genesis/`	ملف `genesis.json` العام للسلسلة (ما تُقلع منه العقدة المُنزَّلة)
GET / POST	`faucet/`	شبكات الاختبار الخاصة فقط (عقدة شُغّلت بمفتاح صنبور على معرِّف شبكة اختبار): الحالة / `{address}` → دفعة 10 JNZ (`Retry-After` + `429` داخل نافذة التهدئة). `404` على الشبكة الرئيسية — لا يوجد صنبور عام.

2.11 المُشغِّل (Actuator) ووثائق الواجهة البرمجية (غير مغلّفة)

المسار	الغرض
`actuator/health/`	الحيوية/الصحّة
`actuator/info/`	معلومات البناء
`actuator/metrics/`	المقاييس
`actuator/prometheus/`	نقطة نهاية كشط Prometheus
`docs/swagger-ui.html/`	واجهة Swagger UI
`docs/api-docs/`	وثيقة OpenAPI

هذه الاستجابات **غير** مغلّفة في المغلّف القياسي.

2.12 معاملات الرموز — `api/v1/transactions/tokens/`

الأسلوب	المسار	الاستعلام / المتن	يُعيد
GET	`transactions/tokens/`	`?+tokenId=&unconfirmed=` pagination	معاملات رموز مرقّمة صفحيًا
GET	`transactions/tokens/{hash}/`	—	معاملة رمز واحدة بحسب التجزئة (hash)
POST	`transactions/tokens/`	tx request body	يرسل عملية رمز ← `201`

تحمل نقطة نهاية واحدة كل عمليات الرموز الأصيلة (إنشاء / سكّ / حرق / تعديل السقف / تحويل) عبر الحقل `op`. راجع §4 (الرموز الأصيلة) للاطّلاع على مجموعة العمليات، والتخطيط البايتي الموقّع، وقواعد الصلاحية.

2.13 الرموز (الحالة) — `api/v1/tokens/`

الأسلوب	المسار	يُعيد
GET	`tokens/` (+ pagination)	جميع تعريفات الرموز
GET	`tokens/{tokenId}/`	تعريف رمز واحد + المعروض الحالي
GET	`tokens/{tokenId}/holders` (+ pagination)/	حائزو رمز ما
GET	`tokens/balances/{address}` (+ pagination)/	أرصدة الرموز لحائزٍ ما

مبالغ الرموز (`cap` و`totalSupply` والأرصدة) هي **وحدات أساسية صحيحة** محمولة كسلاسل عشرية — فلكل رمز `decimals` خاصّة به (للعرض فقط)، لذا فهي **لا** تُحجَّم كتحجيم مبالغ العملة الأصيلة.

2.14 JSON-RPC 2.0 — `POST /rpc` و `ws://…/rpc/ws`

باب ثانٍ إلى **الخدمات والقواعد نفسها** التي تخدمها واجهة REST، بالشكل الذي تتوقعه منصّات التداول وجهات الحفظ. لا يتغيّر شيء في الإجماع ولا في بروتوكول الشبكة بين العقد ولا في صيغة المعاملة الموقَّعة: نداء `janzeer_send*` يأخذ بالضبط حقول جسم `POST` المقابل في REST، والعقدة تتحقّق من التواقيع ولا توقّع أبدًا. يعيد `GET /rpc` قائمة الطرق (ويفعل `janzeer_methods` الشيء نفسه).

- **الغلاف:** JSON-RPC 2.0 القياسي — `{"jsonrpc":"2.0","id":1,"method":"janzeer_getTip","params":[…]}` ← `{"jsonrpc":"2.0","id":1,"result":{…}}` أو `{"jsonrpc":"2.0","id":1,"error":{"code":…,"message":…,"data":…}}`. غلاف REST `{timestamp, version, payload}` **لا** يُطبَّق هنا. رمز HTTP دائمًا `200` (`204` لدفعة مكوّنة من إشعارات فقط).
- **المعاملات (params)** بالاسم (`{"address":"0x"…}`) أو بالموضع (`["0x"…]`) بالترتيب المذكور في الجدول. الدفعات (مصفوفات طلبات) تُعالَج بالترتيب؛ الطلب بلا `id` إشعارٌ لا يُردّ عليه.
- **النهائية فورية:** المعاملة التي تُبلِّغ `FINAL` موجودة في كتلة ملتزَم بها (نصاب COMMIT بمقدار 2f+1). لا يوجد عدد تأكيدات يُنتظر.
- **رموز الأخطاء:** `-32700` خطأ تحليل، `-32600` طلب غير صالح، `-32601` طريقة غير معروفة، `-32602` معاملات غير صالحة (يسرد `data` رسائل الحقول لمعاملة مشوّهة)، `-32603` خطأ داخلي؛ ورموز التطبيق: `-32000` غير موجود، `-32001` رُفضت المعاملة (`data.type` يسمّي القاعدة عند توفّرها)، `-32002` العقدة تتزامن، `-32003` نطاق أو مهلة تتجاوز حدّ الخادم، `-32004` تجاوز حدّ المعدّل.

- **الحدود:** تقبل العقدة لكل عنوان IP للعميل 20 طلبًا/ثانية بدفعة 40 على `rpc/` وعلى إرسال المعاملات عبر REST (HTTP `429` مع `Retry-After: 1` بعد ذلك؛ القراءات الأخرى غير محدودة)؛ حجم جسم `rpc/` حتى 512 كيبيبايت (`413`) والدفعة حتى 50 طلبًا. العقد العامة تضيف حدود nginx أمامها (دليل المشغّل / VPS §9).

الطريقة	المعاملات (بترتيب الموضع)	النتيجة
`janzeer_methods`	—	كل الطرق مع ترتيب معاملاتها
`janzeer_getInfo`	—	`chainId` و`chainSpecDigest` والإصدارات و`nodeKey` و`tip {height, hash, timestamp}` و`epoch` و`syncStatus` و`synchronized` و`peers` و`finality: "instant"` و
`janzeer_getChainSpec`	—	معاملات الإجماع المثبَّتة في التكوين، الرسوم، الوديعة، `maxSupply`، `digest`
`janzeer_getStats`	—	إحصاءات المستكشف (الكتل، المعاملات، المعلّقة، TPS، الحقبة، المعروض)
`janzeer_getAccount`	`address`	`balance` (القابل للإنفاق = الملتزَم − الخصومات المعلّقة)، `committedBalance`، `nextNonce`، `committedNonce`، `pendingCount`، `isValidatorWallet`
`janzeer_getBalance` / `janzeer_getNonce`	`address`	الرصيد القابل للإنفاق / الـ nonce الذي تُوقَّع به المعاملة التالية
`janzeer_getTokenBalances`	`?address, page?, size`	أرصدة JZT-1 مُصفّحة
`janzeer_getTransactionsByAddress`	`?address, page?, size`	التحويلات النهائية المرسَلة أو المستلَمة (مُصفّحة) + `pending` (إدخالات العنوان في تجمّع الانتظار)
`janzeer_sendTransfer` / `janzeer_sendToken` / `janzeer_sendRegisterValidator` / `janzeer_sendExitValidator`	كائن المعاملة الموقَّعة (جسم `POST` في REST)	`hash, status:}` `{""PENDING`؛ إعادة إرسال المعاملة نفسها عملية idempotent
`janzeer_sendRawTransaction`	`transfer` — `type, tx | token | registerValidator | exitValidator`	كما أعلاه

الطريقة	المعاملات (بترتيب الموضع)	النتيجة
`janzeer_getTransactionByHash`	`hash`	`status` `FINAL` (مع `blockHash` و `blockHeight` و `receipt {successful,` `results[]}`) || `PENDING` ؛ شكل واحد لكل `type` `UNKNOWN`
`janzeer_waitForFinality`	`?hash, timeoutMs`	استطلاع طويل (افتراضي 30 ث، حدّ 60 ث): المعاملة `FINAL` ، أو الحالة الراهنة مع `timedOut: true`
`janzeer_estimateFee`	`kind` — `transfer |` `token | tokenCreate |` `registerValidator |` `exitValidator`	`fee, rule: "minimum" |}` `"exact", feeMarket:` `{false`
`janzeer_getTip`	—	رأس كتلة القمّة (`type: "main" |` `""genesis`)
`janzeer_getBlockByNumber` / `janzeer_getBlockByHash`	`hash` | `height` ، `?includeTransactions`	رأس الكتلة (+ `[]transactions` مع الإيصالات)
`janzeer_getBlocks`	`fromHeight, toHeight,` `?includeTransactions`	حتى 100 كتلة تصاعديًا
`janzeer_getToken` / `janzeer_listTokens` / `janzeer_getTokenHolders`	`tokenId` / `page?,` `size?` / `tokenId,` `?page?, size`	تعريف الرمز + المعروض / التعريفات مُصفّحة / الحائزون مُصفّحون
`janzeer_listValidators` / `janzeer_getActiveValidators` / `janzeer_getValidator`	`page?, size?` / — / `key`	`key, walletAddress,}` `registeredAt, active,` `{banned, exited`
`janzeer_getEpoch`	—	`index` ، `genesisHeight/Hash` ، `startTimestamp` ، `nextGenesisHeight` ، `slotPeriodMs` ، `currentSlot` ، `currentSlotOwner` ، `[]validators`
`janzeer_getGenesis`	—	مواصفة الكتلة التكوينية العامة للسلسلة (`networkId` ، `specVersion` ، `treasury` ، `anchors[]` ، `signerPublicKey` ، `mainBlock/genesisBlock` — ملف `{hash, signature}`)

الطريقة	المعاملات (بترتيب الموضع)	النتيجة
		`genesis.json` الذي تُقلِع منه العقدة المُنزَّلة
`janzeer_subscribe` (عبر WebSocket فقط)	`kind, params`	معرّف اشتراك؛ `kind` = `newBlocks` `fromHeight?,}` أو `{?includeTransactions` `addressActivity` `{[...] :addresses}`
`janzeer_unsubscribe` (عبر WebSocket فقط)	`subscription`	`true` / `false`

عبر WebSocket تعمل كل الطرق أعلاه دون تغيير، وتدفع الاشتراكات إشعارات:

```
{"jsonrpc":"2.0","method":"janzeer_subscription",
 "params":{"subscription":"1f0c…","kind":"addressActivity",
           "result":{"blockHeight":128,"blockHash":"…","transaction":
{"hash":"…","type":"transfer","status":"FINAL", "…":"…"}}}}
```

يعيد `newBlocks` مع `fromHeight` بثّ الكتل الرئيسية الملتزَم بها من ذلك الارتفاع قبل متابعة القمّة، بالترتيب، فلا يفوّت المفهرِس الذي يعيد الاتصال أي كتلة. يسلّم `addressActivity` كل معاملة **نهائية** يكون مرسلها أو مستلمها عنوانًا مراقَبًا (تحويلات، عمليات رموز، مكافآت، تسجيلات مدقّقين) — وهو الأساس لاكتشاف الإيداعات. الحدود: 16 اشتراكًا ومخزن إرسال محدود لكل اتصال (العميل الذي لا يواكب يُفصَل)، و1,000 عنوان لكل اشتراك.

مثال عملي — إرسال ثم انتظار النهائية:

```
curl -s localhost:7019/rpc -H 'Content-Type: application/json' -d '[
  {"jsonrpc":"2.0","id":1,"method":"janzeer_getNonce","params":
["0x06e1c0fa9955a700876f8cb0acc7f13fba9fb8ba"]},
  {"jsonrpc":"2.0","id":2,"method":"janzeer_sendTransfer","params":
{"timestamp":1788650000000,"fee":"0.01","nonce":0,

"hash":"…","senderAddress":"0x06e1…","senderSignature":"…","senderPublicKey":"…","amoun
t":"2.5","recipientAddress":"0x598b…"}},
  {"jsonrpc":"2.0","id":3,"method":"janzeer_waitForFinality","params":
{"hash":"…","timeoutMs":30000}}
]'
```

---

3. بناء معاملة وإرسالها (من جهة العميل)

Janzeer **غير حافظ** (لا يحتفظ بالمفاتيح): فلا توجد في العقدة نقاط نهاية لتوليد المفاتيح أو عبارات الاسترداد. فتطبيقك/مجموعة أدواتك (SDK) هي التي تشتقّ المفاتيح، وتبني الصورة الأولية الموقّعة، وتُجزّئ، وتوقّع، وتُرسل المعاملة المكتملة عبر POST. أما العقدة فدورها التحقّق فقط.

3.1 اشتقاق المفاتيح (BIP39 + `m/0/0/0`)

مخطّط الاشتقاق خاص بـ Janzeer — فهو يشبه BIP39/BIP32 لكنه يستخدم سلاسل ملح مخصّصة، لذا **لا** تُعِد استخدام مكتبة جاهزة بصورة عمياء. الملخّص (التفاصيل البايتية الكاملة في الورقة الصفراء):

1. **عبارة الاسترداد ← البذرة:** `PBKDF2-HMAC-SHA512` ، `2048` تكرارًا، خرج `64` بايت. كلمة المرور = `NFKD(mnemonic)` بترميز UTF-8. الملح = السلسلة الحرفية `Janzeer_Blockchain_@` + `NFKD(passphrase)` (تُكمَّل عبارة المرور افتراضيًا بـ `""`). ملاحظة: هذا **ليس** بادئة ملح `"mnemonic"` الخاصة بـ BIP39.
2. **المفتاح الرئيسي الهرمي:** `I = HMAC-SHA512(key = "@_Janzeer_Blockchain_@", msg = seed)` ؛ `IL[0:32]` = المفتاح الخاص الرئيسي، و `IR[32:64]` = شيفرة السلسلة (chain code). ومفتاح HMAC هو سلسلة الملح **نفسها**، لا `"Bitcoin seed"` الخاصة بـ BIP32.
3. **مسار الاشتقاق:** `m/0/0/0` (الحساب / المحفظة / العنوان)، جميعها **غير مصلّدة** (non-hardened)، المنحنى secp256k1، مفاتيح عامة مضغوطة بطول 33 بايت. الاشتقاق CKD غير المصلّد: `data = compressedParentPub(33B) || ser32(i)` ؛ `I = HMAC-SHA512(chainCode, data)` ؛ `childPriv = (IL + parentPriv) mod n` .
4. **العنوان:** `"0x" + hex(keccak256(compressedPubKey)[0:20])` — **Keccak-256** (لا NIST SHA3) فوق المفتاح العام **المضغوط بطول 33 بايت**، مع أخذ أول **20 بايت**. هذا **ليس** مخطّط Ethereum. والصيغة المعيارية المخزّنة/المُجزّأة هي **بأحرف hex صغيرة بالكامل**؛ أما التحقّق الجمعي (checksum) المختلط الحالة وفق EIP-55 الحقيقي فيُطبَّق عند حدود العرض في المحفظة/الواجهة البرمجية فقط (تقديمي، وغير ذي صلة بالتوافق)، ويظل الإدخال بأحرف صغيرة بالكامل مقبولًا.

3.2 خطّ التوقيع

لكل معاملة:

1. **اجلب الرقم التسلسلي (nonce)** لعنوان المُرسِل الخاص بك:

```
curl http://localhost:7019/api/v1/wallets/0x06e1.../nonce
```

2. **ابنِ الصورة الأولية** بربط الحقول الأساسية مع الحمولة الخاصة بالنوع (التخطيطات البايتية في الورقة الصفراء). تبدأ الصورة الأولية الآن **بمعرّف الشبكة** (`consensus.network-id` ، الافتراضي `janzeer`) للحماية من إعادة الإرسال عبر السلاسل — فكل صورة أولية تبدأ بالبايتات ("janzeer") `6a616e7a656572` . جميع الأعداد متعددة البايت بترتيب **big-endian**؛ والسلاسل/العناوين/المفاتيح هي بايتاتها الخام بترميز ASCII-hex / UTF-8؛ وتستخدم المبالغ `toScaledLong(x) = x.setScale(8, HALF_UP) × 10^8` كعدد طويل (long) بطول 8 بايت بترتيب big-endian.

   الأساس (في كل معاملة): `networkId.utf8 || timestamp(8) || toScaledLong(fee)(8) || nonce(8) || senderAddress.utf8() || payload`
   TransferTx: `toScaledLong(amount)(8) ||` حمولة `recipientAddress.utf8 || (data ?? "").utf8`
   PromoterTx: `toScaledLong(amount)(8) ||` حمولة `validatorKey.utf8`
   ExitPromoterTx: `validatorKey.utf8` حمولة TokenTx (لكل العمليات): `op(1) || LP(tokenId) || LP(symbol) || LP(name) || int32(decimals) || LP(cap) || LP(amount) || LP(recipient)` حيث `LP(x) = int32(len) || utf8(x)` — راجع §4

   تُبنى معاملة الرمز الأصيل `TokenTx` بالطريقة نفسها تمامًا: الصورة الأولية الأساسية ذاتها، ثم حمولة الرمز أعلاه. ولا تزال الرسوم الأساسية `fee` مُحجَّمة بـ `toScaledLong` (فهي مدفوعة بالعملة الأصيلة)، لكن حقلَي الرمز `cap` و `amount` هما **وحدات أساسية صحيحة محمولة كسلسلتها العشرية** — فلكل رمز `decimals` خاصّة به — لذا فهما **لا** يمرّان عبر `toScaledLong` .

3. **التجزئة:** `txHash = doubleSHA256(preimage)` ، بأحرف hex صغيرة.

4. **التوقيع:** ECDSA فوق secp256k1، حتمي وفق **RFC-6979** (بـ `k` من HMAC-SHA256)، مع فرض **canonical low-S**، مُرمَّز بـ **DER** ثم **Base64**. والتوقيع يجري فوق **الملخّص الخام بطول 32 بايت** — دون أي تجزئة إضافية — لذا اضبط

`prehash: false` في مكتبة التوقيع لديك.

5. **أرسِل (POST)** المعاملة المكتملة إلى نقطة النهاية المطابقة.

3.3 مثال تحويل مشروح (شبه شيفرة بأسلوب JS)

```
// 0. Derive keys from mnemonic (Janzeer scheme — see §3.1 / Yellow Paper)
const { privKey, pubKeyCompressed, address } = deriveJanzeer(mnemonic); // m/0/0/0

// 1. Fetch the nonce
const nonce = (await getJSON(
  `http://localhost:7019/api/v1/wallets/${address}/nonce`)).payload;

// 2. Transaction fields
const networkId = "janzeer";   // consensus.network-id (per chain)
const tx = {
  timestamp: Date.now(),
  fee: "0.01",                 // ≥ MINIMUM_FEE
  nonce,
  senderAddress: address,
  amount: "1.5",               // ≥ 0.1 unless a memo is attached
  recipientAddress: "0xRECIPIENT...",
  data: ""                     // optional ≤256B memo
};

// 3. Build the preimage (big-endian; amounts scaled ×10^8)
const preimage = concatBytes(
  utf8(networkId),             // network-id prefix (bytes 6a616e7a656572)
  i64be(tx.timestamp),
  i64be(toScaledLong(tx.fee)),
  i64be(tx.nonce),
  utf8(tx.senderAddress),
  // TransferTx payload:
  i64be(toScaledLong(tx.amount)),
  utf8(tx.recipientAddress),
  utf8(tx.data)
);

// 4. Hash + sign
const digest = sha256(sha256(preimage));           // double-SHA256, 32 bytes
const sigDER = secp256k1.sign(digest, privKey, {   // RFC-6979, low-S
  prehash: false, der: true
});

tx.hash            = toHex(digest);                // lowercase hex
tx.senderSignature = base64(sigDER);
tx.senderPublicKey = toHex(pubKeyCompressed);      // 33-byte compressed

// 5. Submit
await postJSON("http://localhost:7019/api/v1/transactions/transfers", tx); // → 201
```

3.4 حقول طلب POST بحسب النوع

كل طلب POST لمعاملة يحمل الحقول المشتركة: `timestamp` و`fee` و`nonce` و`hash` و`senderAddress` و`senderSignature` و`senderPublicKey`. ثم أضِف الحقول الخاصة بالنوع:

نوع المعاملة	نقطة النهاية	الحقول الإضافية
Transfer	`transactions/transfers/`	`data`, `recipientAddress?`, `amount?`
Promoter	`transactions/validators/`	`amount`, `validatorKey`
ExitPromoter	`transactions/exit-validators/`	`validatorKey`
Token	`transactions/tokens/`	`op`, `tokenId?`, `symbol?`, `name?`, `decimals?`, `cap?`, `amount?`, `recipient`

الحدود الدنيا والرسوم (بالضبط): `MAX_DATA_SIZE = 256`، `MINIMUM_AMOUNT = 0.1`، `MINIMUM_FEE = 0.01` بايت؛ رسوم PromoterTx هي بالضبط `3` مع وديعة غير قابلة للاسترداد قدرها `2000`؛ ورسوم ExitPromoterTx ≥ `0.01`؛ ورسوم إنشاء TokenTx (CREATE) هي بالضبط `5` وسائر عمليات الرموز (سكّ / حرق / تعديل السقف / تحويل) ≤ `0.01`. ويُرسَل حقلا الرمز `cap` و`amount` كسلاسل نصية (**strings**) في JSON (عشريات وحدات أساسية صحيحة) تفاديًا لفقدان الدقّة. وتسمح ذاكرة المعاملات المعلّقة (mempool) بأرقام تسلسلية **متتالية** (pipelining): الرقم التسلسلي المتوقَّع = `confirmedNonce + pendingCount`.

استبدال معاملة معلّقة («التسريع»). تشغل المعاملة المعلّقة رقم تسلسلها (nonce)، وتنتظر خلفها أرقام المُرسِل اللاحقة. لاستبدالها أرسل معاملة **بالرقم نفسه وبرسوم أعلى حصرًا** — فتُدخلها العقدة مكان القديمة وتُسقط القديمة (وتُرفض الرسوم المساوية أو الأدنى: `A pending transaction with nonce N exists; a replacement must pay a higher fee than …`). ولا يمكن أن تُخصم إلا رسوم البديلة. يسرد `GET /api/v1/transactions/transfers?address=…&unconfirmed=true` حوالات المحفظة المعلّقة؛ ومنذ العقدة 0.1.0 تحمل كل حوالة في استجابة REST حقل **`nonce`**، وهو ما تحتاجه المحفظة لإعادة بناء الحوالة. وتُطرد المعاملة المعلّقة التي لا تُدرج خلال 6 ساعات. وتقدّم المحافظ الرسمية ذلك باسم *تسريع*: المستلم والمبلغ والملاحظة أنفسهم، والرسوم يختارها المستخدم (المقترح: الضعف).

الطابع الزمني. اضبط `timestamp` على الوقت الحالي بالمللي ثانية **لحظة التوقيع** — فهو جزء من البيانات الموقَّعة. تقبل العقدة المعاملة إذا كان طابعها الزمني متقدّمًا على ساعتها بما لا يزيد على **60 ثانية** وقديمًا بما لا يزيد على **6 ساعات**؛ وداخل الكتلة تكون نافذة الإجماع `block − 6 h … block + 2 min`. الساعات الست ليست تأخيرًا: إنها عمر المعاملة في مجمّع الانتظار، أي المدة التي قد تنتظرها معاملة موقّعة قبل أن تضمّها كتلة (الإدراج المعتاد يستغرق فترة أو فترتين من 15 ثانية). لا توقّع المعاملات مسبقًا لإرسالها لاحقًا ولا تُعِد استخدام حمولة موقّعة قديمة — أعد التوقيع بطابع زمني جديد. الجهاز ذو الساعة الخاطئة يتلقى `Transaction 400 is older than 6 h — … sign it again`، والحمولة القديمة تتلقى `timestamp is in the future (check the device clock)`. ويُحسب `MAX_DATA_SIZE` **ببايتات UTF-8** لا بعدد الأحرف (الحرف العربي بايتان والرمز التعبيري أربعة).

---

4. الرموز الأصيلة (JZT-1)

يدعم Janzeer **رموزًا قابلة للاستبدال (fungible)** اختيارية يصدرها المُصادِقون وتعيش إلى جانب العملة الأصيلة على السلسلة ذاتها. وهي **دفتر حسابات منفصل**: فمعروض الرمز **ليس** جزءًا من معروض العملة الثابت البالغ `100,000,000` ولا يمسّ أبدًا الإصدار أو الخزينة أو مكافآت الكتل. ولا تزال كل عملية رمز تدفع رسومها **بالعملة الأصيلة**. وقد كانت الرموز فعّالة منذ التكوين (genesis) على شبكة جديدة (تبلغ `token-activation-height` افتراضيًا `0`، وهي مثبّتة في بصمة مواصفات السلسلة).

4.1 نوع معاملة واحد، وبايت `op`

لا توجد خمسة أنواع منفصلة لمعاملات الرموز — فكل عملية رمز هي معاملة `TokenTx` واحدة تحمل بايت `op` :

`op`	العملية	من يجوز له الإرسال
`0`	CREATE (إنشاء)	مُصادِق مُسجَّل (عنوان `PromoterState.walletAddress` ، نشط أو احتياطي)
`1`	MINT (سكّ)	مُصدِر الرمز (منشئه) فقط
`2`	BURN (حرق)	مُصدِر الرمز فقط
`3`	SETCAP (تعديل السقف)	مُصدِر الرمز فقط
`4`	TRANSFER (تحويل)	أي حائز لرصيد

4.2 التخطيط البايتي للحمولة الموقّعة

الصورة الأولية الموقّعة هي الأساس المعتاد (`networkId || timestamp || toScaledLong(fee) || nonce || senderAddress`) متبوعًا بحمولة رمز معيارية واحدة **لكل** عملية، مع بادئة طول لكل حقل متغيّر:

```
op(1) || LP(tokenId) || LP(symbol) || LP(name) || int32(decimals) || LP(cap) || LP(amount) || LP(recipient)
    where LP(x) = int32(len) || utf8(x)      // 4-byte big-endian length, then the UTF-8 bytes
```

- `op` ∈ { CREATE `0` ، MINT `1` ، BURN `2` ، SETCAP `3` ، TRANSFER `4` } .
- الحقول الغائبة هي السلسلة الفارغة (طول `0`) — فمثلًا يحمل التحويل `tokenId` و `amount` و `recipient` ويترك `cap` / `name` / `symbol` فارغة.
- **`cap` و `amount` هما وحدات أساسية صحيحة محمولة كسلسلتهما العشرية — لا `toScaledLong` .** فلكل رمز `decimals` خاصّة به (`0`–`18` ، للعرض فقط)، لذا فإن وحداته الأساسية مستقلّة عن تحجيم العملة الأصيلة ذي الثماني منازل.
- تُحسب التجزئة والتوقيع فوق هذه الصورة الأولية تمامًا كأي معاملة أخرى (double-SHA256، وlow-S RFC-6979، وDER → Base64). راجع §3 للخطّ الكامل — فمعاملة TokenTx تُبنى وتُوقَّع بالطريقة ذاتها، فقط بهذه الحمولة.

4.3 الصلاحية والقواعد

- **CREATE (op 0)** — لا يجوز إنشاء رمز إلا لـ**مُصادِق مُسجَّل**. ويضبط `symbol` (1–12 محرفًا)، و `name` (1–40)، و `decimals` (0–18، للعرض فقط)، و `cap` (المعروض الأقصى)، و `initialSupply` اختياري (≤ `cap` ، يُقيَّد للمنشئ). ويكون `tokenId` **الجديد هو تجزئة معاملة CREATE** (فريد عالميًا). ولا يوجد سجلّ للرموز، لذا فإن الرموز (symbols) **ليست** فريدة. ويدفع رسوم CREATE الدقيقة البالغة `5` .
- **MINT / BURN / SETCAP (ops 1–3)** — لا يجوز إلا لـ**مُصدِر** الرمز (منشئه) أن يرفع المعروض أو يخفضه أو يغيّر السقف. ويُفرَض الثابت `totalSupply ≤ cap ≥ 0` على المسارَين الحيّ والمتزامن، ولا يمكن لـ SETCAP إنزال السقف دون المعروض المسكوك أصلًا. الرسوم ≤ `0.01` .
- **TRANSFER (op 4)** — يجوز لـ**أي** حائز لرصيد رمز أن يحوّله إلى أي عنوان. الرسوم ≤ `0.01` .

تُفرَض هذه القواعد الاقتصادية/قواعد الصلاحية في خدمة المعاملات (فتعمل بشكل متطابق على المنتِج وعلى كل نظير يعيد التنفيذ)، **كما** يُعاد فحصها عند إدخال ذاكرة المعاملات المعلّقة (mempool)، فتُرفض معاملة الرمز غير القابلة للتعدين عند الإرسال (HTTP `400`) بدلًا من إهدار فتحة كتلة.

4.4 قراءة حالة الرموز

استخدم نقاط النهاية الواردة في §2.12 (معاملات الرموز) و§2.13 (حالة الرموز): `POST /api/v1/transactions/tokens` لإرسال عملية، وقراءات `*/GET /api/v1/tokens` للتعريفات والمعروض والحائزين والأرصدة لكل عنوان. راجع مرجع REST أعلاه للتفاصيل الكاملة.

---

5. ابنِ على Janzeer — مجموعات الأدوات (SDK) الرسمية والتوافق

لست مضطرًا إلى إعادة تنفيذ مخطّط التشفير في §3: خمس مجموعات أدوات رسمية تقوم بذلك، وهي مُثبتة بايتًا ببايت مقابل العقدة، وتغلّف واجهتَي REST وJSON-RPC في §2 بعملاء مُنمَّطين. جميعها بترخيص Apache-2.0، وغير حافظة للمفاتيح (لا تغادر المفاتيح عمليتك أبدًا)، ومختبَرة مقابل العقدة 0.1.0 / الواجهة 1.1.0.

5.1 مجموعات الأدوات

اللغة	الحزمة	التثبيت	المنصّات
TypeScript / JavaScript	`janzeer-sdk`	`npm install janzeer-sdk`	Node 20+، المتصفّحات (JavaScript خالص، يعمل على صفحات http عادية)، Deno/Bun
Dart / Flutter	`janzeer_sdk`	`dart pub add janzeer_sdk`	Dart VM، وFlutter للجوّال وسطح المكتب والويب
Kotlin / JVM / Android	`org.janzeer:janzeer-sdk-client` (+ `janzeer-sdk-core`)	اعتمادية Gradle/Maven	JVM 11+، Android API 26+، وJava العادية
Python	`janzeer-sdk` (الاستيراد: `janzeer`)	`"pip install "janzeer-sdk[ws]`	Python 3.10–3.13 على لينكس وماك وويندوز؛ عملاء متزامنون وعملاء `asyncio`
Go	`github.com/janzeerorg/janzeer-sdk-go`	`go get github.com/janzeerorg/janzeer-sdk-go`	Go 1.26 فما فوق، بلا cgo (يُبنى لكل المنصّات)

لكل مجموعة أدوات الشكل نفسه: `Account` / `Mnemonic` (المفاتيح والعناوين والتوقيع)، و`TxBuilder` (التحويل، وتسجيل المُصادِق وخروجه، وعمليات الرموز JZT-1 الخمس) ← `SignedTx` ← `UnsignedTx`، وعميل REST مُنمَّط (`JanzeerClient`)، وJSON-RPC عبر (`JanzeerRpc`) HTTP وعبر WebSocket مع اشتراكات / `newBlocks`

(`JanzeerRpcWs`) `addressActivity`، ومساعدَا `sendAndWait` / `waitForFinality`، وتسلسل أخطاء مُنمَّط (`NonceMismatch`، و`TxRejected` مع `type` الصادر عن العقدة، و`RateLimited`، ...)، وخزنة مشفّرة اختيارية لحفظ العبارة. المبالغ سلاسل عشرية، وتُحلَّل الاستجابات دون فقد — لا تمرّ أبدًا عبر عدد عشري عائم.

التكامل كله ستة أسطر في أيٍّ منها — اشتقاق ← nonce ← بناء ← توقيع ← إرسال ← انتظار:

```
// TypeScript
const client = new JanzeerClient('http://localhost:7019'); const rpc = new
JanzeerRpc('http://localhost:7019');
const me = Account.fromMnemonic(phrase);
const tx = me.signTx(TxBuilder.transfer({ from: me.address, to, amount: '1.25', nonce:
await client.nonce(me.address) }));
await client.submit(tx);
const final = await waitForFinality(rpc, tx.hash);   // فتحة أو فتحتان من 15 ثانية؛
الكتلة الملتزَم بها نهائية
```

```
// Dart
final tx = me.signTx(TxBuilder.transfer(from: me.address, to: to, amount: '1.25',
nonce: await client.nonce(me.address)));
await client.submit(tx);
final fin = await waitForFinality(rpc, tx.hash);
```

```
// Kotlin (من Java: TxBuilder.transfer(from, to, "1.25", nonce)،
وHelpers.waitForFinality(rpc, tx.getHash()))
val tx = me.signTx(TxBuilder.transfer(from = me.address, to = to, amount = "1.25",
nonce = client.nonce(me.address)))
client.submit(tx)
val fin = Helpers.waitForFinality(rpc, tx.hash)
```

```
# Python (النُّسخ غير المتزامنة: AsyncJanzeerClient وAsyncJanzeerRpc وJanzeerRpcWs
وawait_finality)
tx = me.sign_tx(TxBuilder.transfer(sender=me.address, to=to, amount="1.25",
nonce=client.nonce(me.address)))
client.submit(tx)
final = wait_for_finality(rpc, tx.hash)
```

```
// Go (كل نداء يأخذ context؛ والأخطاء قيم مُنمّطة تُفحص بـ errors.As)
utx, _ := janzeer.NewTransfer(janzeer.Transfer{Common: janzeer.Common{From:
me.Address(), Nonce: nonce}, To: to, Amount: "1.25"})
tx, _ := me.SignTx(utx)
_, err := c.Submit(ctx, tx)
final, err := rpc.WaitForFinality(ctx, r, tx.Hash)
```

يشحن مستودع كل مجموعة أدوات ملف README بهذا البدء السريع، وأدلّة (المحفظة والمفاتيح، وإرسال كل نوع معاملة، وقراءة بيانات السلسلة، وJSON-RPC والاشتراكات، والأخطاء وnonce، والخزنة، والترحيل)، ومرجعًا مولَّدًا للواجهة (TypeDoc / dartdoc / Dokka)، وأمثلة قابلة للتشغيل. ومحفظة مستكشف Janzeer، ومحفظة تطبيق Telegram المصغّر، ومحفظة Flutter مبنية عليها.

5.2 عدّة التوافق

أيًّا كانت اللغة، يجب أن يعيد العميل إنتاج مخطّط التشفير **بايتًا ببايت** وإلا رفضت العقدة معاملاته. عدّة التوافق العامة (`/sdk_conformance`) هي ما تُختبَر مقابله مجموعات الأدوات — وأي عميل من طرف ثالث:

- **المتجهات** (`vectors/wallet-parity-vectors.json`، الصيغة v2، مولَّدة بواسطة اختبار العقدة `WalletParityVectorsTest`): عبارة الاسترداد ← البذرة ← مفتاح `m/0/0/0` ← العنوان؛ وعنوان من مفتاح خام؛ ولكل نوع معاملة يمكن للعميل إرساله (تحويل مع مذكّرة وبدونها، وتسجيل مُصادِق، وخروج مُصادِق، وإنشاء رمز وتحويله) بايتات الصورة الأولية والتجزئة والتوقيع بالضبط؛ وجدول للمجموع الاختباري EIP-55؛ وجدول `toScaledLong`. وكل صورة أولية تبدأ ببايتات معرّف الشبكة (janzeer" = "`6a616e7a656572`"). المتجه القياسي: العبارة ← `abandon … about` ← `0x06e1c0fa9955a700876f8cb0acc7f13fba9fb8ba`.
- **تدفّق الطرف إلى الطرف** (`e2e/SPEC.md`)، 14 خطوة مقابل شبكة حقيقية: بوّابة الإصدار، والحساب، والاشتراك أولًا، والبناء/التوقيع، والإرسال عبر REST (201)، وإعادة الإرسال المتكرّرة الآمنة عبر RPC، والنهائية على عقدة أخرى، وحساب الرصيد بدقّة، وإشعار WebSocket، وثلاث حالات سلبية (فجوة `INVALID_NONCE` ← nonce، وتوقيع مزوّر ← `INCORRECT_SIGNATURE`، وعنوان خاطئ ← `-32602`).
- نموذج توافق للخزنة (كتلة PBKDF2-SHA256/250k + AES-256-GCM) لتنتقل العبارات المشفّرة بين مجموعات الأدوات.

5.3 كتابة عميلك الخاص

أعد إنتاج المتجهات بهذا الترتيب، ثم شغّل تدفّق SPEC:

- [ ] جدول `scaled`: `toScaledLong(0.01) == 1_000_000`، و`toScaledLong(1.5) == 150_000_000` (8 منازل عشرية، HALF_UP)
- [ ] `abandon … about` → `0x06e1c0fa9955a700876f8cb0acc7f13fba9fb8ba` (ملح PBKDF2 ومفتاح HMAC هو `@_Janzeer_Blockchain_`، والمسار `m/0/0/0`)
- [ ] العنوان بأحرف صغيرة، `0x` + 40 محرف hex، من `keccak256(compressedPub)[0:20]`؛ وحالة الأحرف EIP-55 للعرض فقط
- [ ] كل `preimageHex` مطابق؛ والتجزئة هي `doubleSHA256(preimage)`
- [ ] التوقيع RFC-6979، وlow-S، وDER، وBase64، فوق الملخّص الخام بطول 32 بايت (`prehash: false`)
- [ ] الأخطاء: اقرأ `payload` عندما يحويه الجسم (الغلاف يغلّف الأخطاء أيضًا)، والحمولة المصفوفة خطأ تحقّق، و`type` يسمّي سبب الرفض

مصادر مجموعات الأدوات هي التطبيقات المرجعية (`src/crypto` في `janzeer-sdk`، و`lib/src` في `janzeer_sdk`، و`janzeer-sdk-core`)؛ والتخطيطات البايتية المعتمَدة في الورقة الصفراء.

---

6. ملاحظة بشأن عدم حفظ المفاتيح

عقدة Janzeer **غير حافظة** للمفاتيح بصرامة:

- تُشتقّ المفاتيح وتُحفظ **من جهة العميل فقط**؛ ولا ترى العقدة أبدًا مفتاحًا خاصًّا أو بذرة.
- **لا توجد** نقاط نهاية لتوليد المفاتيح أو عبارات الاسترداد أو التوقيع على العقدة — فبناء المعاملات وتوقيعها مسؤولية تطبيقك بالكامل (راجع §3).
- يتمثّل دور العقدة في **التحقّق** من التواقيع، وفرض الأرقام التسلسلية (nonce) لكل حساب إضافةً إلى معرّف الشبكة في الصورة الأولية الموقّعة (الحماية من إعادة الإرسال، بما في ذلك عبر السلاسل) وحارس الرصيد السالب، وتطبيق انتقالات الحالة.

- خزينة التكوين (genesis) (bank) `0x0000000000000000000000000000000000000000` بلا مفتاح وغير قابلة للاستيلاء؛ ولا يستطيع تحريك عملاتها إلا منطق المكافأة/الرسوم الخاص بالبروتوكول.

ولأن النهائية تتطلّب نِصابًا حيًّا من المجموعة النشطة، فإن العقدة الواحدة لا تستطيع إنهاء الكتل بمفردها — استخدم الشبكة المحلّية متعددة العقد (§1.2)، بـ 4 عقد على الأقل، لإجراء اختبار شامل واقعي من طرف إلى طرف.

حالة الوثيقة: الإصدار 1.0. تعكس إصدار العقدة 0.1.0، بما في ذلك الرموز الأصيلة (JZT-1). الحقائق الدقيقة على مستوى البايت (تخطيطات الصورة الأولية، والتجزئة، وتحجيم المبالغ، وثوابت الاشتقاق) موجودة في الورقة الصفراء.