Подключение к KodaCash API

QR-платежи в ARS · API v1 · 23 ноября 2026

Адрес	Значение
API base URL	https://kodacash.app/integrations
Кабинет	https://kodacash.app/cabinet/
PDF	https://kodacash.app/integrations/docs/partner-api.pdf
TXT	https://kodacash.app/integrations/docs/partner-api.txt
OpenAPI	https://kodacash.app/integrations/openapi.json

Первый вход и ключ API

Войдите в кабинет с логином и паролем от KodaCash. Откройте «Подключение», выберите своего API-мерчанта и скопируйте merchant_id.

Нажмите «Выпустить ключ» и подтвердите выпуск. В открывшемся окне нажмите «Скопировать» и сохраните секрет в менеджере секретов. После закрытия окна доступен только ID ключа. Для запросов используйте секрет в заголовке Authorization: Bearer <API_KEY>.

Если ключ уже активен, используйте сохранённый секрет. Для замены нажмите «Перевыпустить ключ». После подтверждения старый ключ сразу теряет доступ. Если кнопки выпуска нет, запросите у администратора право на выпуск для своего аккаунта.

Права для API

Метод	Право
GET /v1/capabilities	catalog.read
POST /v1/payment-checks	checks.create, checks.read
GET /v1/payment-checks/{check_id}	checks.read
POST /v1/payment-checks/{check_id}/refresh	checks.refresh, checks.read
POST /v1/payment-checks/{check_id}/confirm	checks.confirm, checks.read
GET /v1/orders/{order_id} и /events	orders.read

Права и область доступа назначает KodaCash. Ключ действует для привязанного merchant_id и разрешённых режимов и направлений. Для live KodaCash отдельно включает доступ, платёжный маршрут ARS QR, правило автоматизации мерчанта и лимиты.

Создание проверки в sandbox

Примеры используют Bash, curl и jq 1.7 или новее. Для этой сессии Bash введите секрет в API_KEY командой read -rs API_KEY (ввод скрыт) и нажмите Enter. Подставьте свой merchant_id в check.json.

1. Проверить доступность ARS QR

```
BASE=https://kodacash.app/integrations
curl -sS -i -H "Authorization: Bearer $API_KEY" \
  "$BASE/v1/capabilities?mode=sandbox"
```

mode=sandbox соответствует «Симуляции», mode=live — «Боевому». GET проверяет доступ к направлениям выбранного режима. В ответе HTTP 200 в routes найдите currency="ARS" и payment_method="qr". Читайте amount_scale и qr_formats этого направления. Пустой routes означает, что для ключа в выбранном режиме нет доступных направлений.

2. Подготовить check.json

```
{
  "merchant_id": "your-merchant-id",
  "external_order_id": "ars-order-001",
  "idempotency_key": "ars-check-001",
  "currency": "ARS",
  "amount": "3000.00",
  "payment_method": "qr",
  "payment_data": {
    "qr_payload": "KODA-SANDBOX:ARS:success"
  },
  "execution_mode": "sandbox"
}
```

Используйте новые external_order_id и idempotency_key для каждой операции. Допустимы 1-128 символов по шаблону ^[A-Za-z0-9][A-Za-z0-9._:-]{0,127}$. amount - положительная десятичная строка с точкой, для ARS до двух знаков после точки. Курс и суммы из pricing и settlement храните с полной точностью в decimal.

3. Отправить проверку

```
curl -sS -o check-response.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" -H 'Content-Type: application/json' \
  --data-binary @check.json "$BASE/v1/payment-checks"
```

HTTP 201 означает, что проверка сохранена. Объект check находится в теле ответа, сохранённом в check-response.json. Переходите к проверке условий при check.status="ready".

Проверка условий в sandbox

4. Прочитать условия из check

```
jq '.check | {check_id,status,can_confirm,expires_at,order_id,
  amount,currency,payee,payee_verified,pricing,quote_fingerprint,
  failure_code,limits}' check-response.json
```

Сверьте amount, currency и payee. Подтверждение доступно при status="ready", can_confirm=true и действующем expires_at. Используйте актуальный expires_at (ISO 8601). Для оценки Bitso окно может быть короче общего срока проверки.

При оценке Bitso payee может прийти из QR или быть null. payee_verified=false означает, что провайдер ещё не проверил получателя. В условиях провайдера используйте payee из ответа. Поле payee_verified в этих условиях может отсутствовать.

Блок ниже - фрагмент check из прогона 3000 ARS. Значения check_id и quote_fingerprint относятся к этому прогону. Для своих запросов берите их из собственных ответов.


```
{
  "check_id": "53906c53-65de-4c5d-b5bb-bf7a8d7c2364",
  "amount": "3000.00",
  "currency": "ARS",
  "status": "ready",
  "can_confirm": true,
  "payee_verified": false,
  "rate_observed_at": "2026-11-19T12:28:12Z",
  "quote_fingerprint": "1765222c07869e7e636c915c128620e9ab0ad1c07411a6b7db5c0d4c6343b59e",
  "pricing": {
    "estimated_amount": "1.75",
    "estimated_rate": "0.000583107379223884",
    "koda_fee_amount": "0",
    "koda_fee_currency": "USDT",
    "rate": "0.000583107379223884",
    "rate_base": "ARS",
    "rate_quote": "USDT",
    "rate_status": "estimated",
    "source": "bitso",
    "version": 0
  }
}
```


Чтение и обновление проверки

GET /v1/payment-checks/{check_id} возвращает сохранённый check. POST /v1/payment-checks/{check_id}/refresh с телом {} обновляет условия, пока проверка действует. После refresh заново проверьте условия и quote_fingerprint. При status="failed" смотрите failure_code.

Подтверждение оплаты в sandbox

5. Подтвердить условия текущей проверки

```
CHECK_ID=$(jq -ber '.check.check_id' check-response.json)
FINGERPRINT=$(jq -ber '.check.quote_fingerprint' check-response.json)
jq -n --arg f "$FINGERPRINT" '{confirm:true,quote_fingerprint:$f}' > confirm.json
curl -sS -o confirm-response.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" -H 'Content-Type: application/json' \
  --data-binary @confirm.json "$BASE/v1/payment-checks/$CHECK_ID/confirm"
```

HTTP 202 с объектами order и check означает, что заказ принят. Сохраните order.order_id. Фактическое списание провайдера может превысить подтверждённую оценку в USDT.

HTTP 409 quote_changed

```
{
  "error": {
    "code": "quote_changed"
  }
}
```

При первичной оценке Bitso новый получатель или рост оценки провайдера требуют повторного согласия. При неизменном получателе и оценке провайдера не выше текущей первый confirm может сразу вернуть 202. При 409 прочитайте тот же check_id через GET. Если уже есть order_id, опрашивайте этот заказ.

```
curl -sS -o check-response.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" "$BASE/v1/payment-checks/$CHECK_ID"
```

При HTTP 200 и order_id=null заново проверьте получателя, сумму, pricing, expires_at и can_confirm. После согласия повторите шаг 5 с новым quote_fingerprint из GET. Если отпечаток прежний, проверьте failure_code и can_confirm. При необходимости обновите действующую проверку через refresh и заново проверьте условия.

В прогоне 3000 ARS проверка сохранила исходную оценку Bitso 1.75 USDT в pricing.initial_estimated_amount. Новая оценка провайдера пришла в pricing.estimated_amount, 1.80 USDT.

Результат и списание в sandbox

6. Опросить заказ до получения расчётов

После HTTP 202 возьмите ID из confirm-response.json. Если прочитали созданный заказ из check, используйте check.order_id.

```
ORDER_ID=$(jq -ber '.order.order_id' confirm-response.json)
curl -sS -o order.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" "$BASE/v1/orders/$ORDER_ID"
jq '{order_id,status,status_version,pricing,settlement}' order.json
```

При succeeded продолжайте опрос того же order_id до появления нужных полей. Для live их дополняет фоновая сверка.

В симуляции 3000 ARS итоговый курс 0.00061 USDT за 1 ARS, списание 1.83 USDT. Фрагмент ответа.


```
{
  "order_id": "53906c53-65de-4c5d-b5bb-bf7a8d7c2364",
  "currency": "ARS",
  "amount": "3000.00",
  "status": "succeeded",
  "execution_mode": "simulated",
  "pricing": {
    "estimated_amount": "1.80",
    "estimated_rate": "0.00060",
    "final_rate_source": "provider",
    "initial_estimated_amount": "1.75",
    "initial_rate": "0.000583107379223884",
    "initial_source": "bitso",
    "koda_fee_amount": "0",
    "koda_fee_currency": "USDT",
    "rate": "0.00061",
    "rate_base": "ARS",
    "rate_quote": "USDT",
    "rate_status": "confirmed",
    "source": "provider",
    "version": 0
  },
  "settlement": {
    "actual_debit": "1.83",
    "asset": "USDT",
    "estimated_amount": "1.80",
    "fee_amount": null,
    "fee_currency": null,
    "rate": "0.00061",
    "rate_base": "ARS",
    "rate_quote": "USDT",
    "status": "confirmed"
  }
}
```


Итоговый курс - pricing.rate при pricing.rate_status="confirmed". Списание - settlement.actual_debit в settlement.asset. Поля могут появиться в разных ответах. Даже при settlement.status="confirmed" одно поле может быть null.

Комиссия KodaCash - 0 (pricing.koda_fee_amount). Комиссия провайдера при наличии отражена в settlement.fee_amount и settlement.fee_currency.

Повторы и переход в live

Статусы и события

status	Действие
accepted / processing	Продолжайте опрос исходного заказа.
pending_reconciliation	Исход уточняется. Сохраните исходный order_id и продолжайте опрос.
succeeded	Оплата исполнена. Проверяйте актуальные поля settlement.
failed / expired	Окончательный отказ или срок истёк до отправки. Читайте failure_code.

GET /v1/orders/{order_id}/events возвращает HTTP 200 с массивом events. Снимок order в событии содержит status_version. Применяйте только более новые версии, устраняйте дубли по event_id. Для заказа и событий начните с интервала в 5 секунд, при ожидании увеличьте до 15-30 секунд. Интеграция получает результат опросом API.

Повторы и ошибки

Ситуация	Действие
Повтор POST payment-checks	Сохраните тело и idempotency_key. Повтор вернёт прежний check_id с HTTP 201. Изменённое тело с тем же ключом вернёт 409 idempotency_conflict.
Таймаут / 5xx после confirm	GET исходной проверки. При наличии order_id опрашивайте заказ. Иначе повторите исходный confirm с прежним отпечатком. Точный повтор уже принятого confirm вернёт тот же заказ.
check_expired	Сначала проверьте исходный order_id. При его отсутствии для новой попытки задайте новые external_order_id и idempotency_key.

Для ошибок читайте error.code или check.failure_code. HTTP 401 - проверьте ключ. 403 - согласуйте права, merchant_id и режим. 404 - проверьте ID и область доступа. При 409 проверьте can_confirm, expires_at, failure_code и limits. При rate_policy_changed обновите действующую проверку через refresh.

Запуск live

После отдельного допуска прочитайте GET /v1/capabilities?mode=live. Для проверки задайте новые external_order_id и idempotency_key, execution_mode="live" и полный текст действующего аргентинского QR. При HTTP 202 после confirm заказ принят к реальному исполнению. Прямой POST /v1/orders в live вернёт 409 preparation_required.

API сверяет сумму фиксированного распознанного QR с amount. Если в QR суммы нет, передайте согласованную сумму. При расхождении возможны qr_amount_mismatch и provider_qr_amount_mismatch. Предел QR - 4096 байт UTF-8 в live и 8192 в sandbox. Предел JSON - 16 384 байта. Провайдер может ограничивать QR дополнительно.