Shopify Integration

Создайте проект в Newo

Если проект уже создан в **Newo Builder**, пропустите этот подраздел и переходите к настройкам конкретной интеграции ниже.

1. В **Newo Builder** откройте список проектов и нажмите **Create Project** или **Create New Project** в меню справа сверху.


Меню Create New Project в Newo Builder

2. Заполните **IDN** и **Title**. Имена не важны: используйте любые понятные названия, по которым команда узнает клиента или локацию.
3. В **Registry** выберите канал релиза:
   - **staging** — здесь первыми появляются последние фиксы модуля. Используйте, когда нужно проверить свежий фикс, но учитывайте риск незавершенных изменений.
   - **production** — финальная стабильная версия для рабочих проектов.
4. В **Module** выберите модуль этой интеграции.


Форма Create Project с полями IDN, Title, Registry и Module 5. Оставьте **Module version** = **Latest version**, если поддержка не попросила закрепить конкретную версию, затем нажмите **Create**.

What the AI can do

Summary

Shopify Integration подключает **Shopify Admin GraphQL API** к **Newo Platform** и сейчас сфокусирована на одном основном customer journey:

- периодически выгружать весь каталог товаров в customer attribute
- проверить наличие товара
- создать draft order
- дать клиенту invoice link
- при необходимости обновить или отменить draft order
- после оплаты перевести booking state из `draft` в `paid`
- при запросе клиента проверить order/payment status

Интеграция хранит customer-facing состояние заказа в persona attribute `bookings`.

1.2 Текущий pipeline

Текущая логика построена вокруг оплаты draft order:

1. **CollectProductsFlow** На `session_started` или по ручному trigger интеграция обновляет `shopify_products_catalog` и складывает туда нормализованный каталог всех товаров.
2. **ProductAvailabilityFlow** Агент ищет товар и проверяет остатки.
3. **CreateDraftOrderFlow** Агент создаёт draft order в Shopify, сохраняет запись в `bookings`, помечает её как:

- `status = "draft"`
- `is_draft = "true"`
- `payment_status = "pending"`

4. После успешного создания draft order агент:
   - получает `invoice_url`
   - получает prompt-context через `set_prompt_custom_section`
   - должен сообщить клиенту, что заказ ещё **не оплачен**
5. **UpdateDraftOrderFlow** Пока заказ не оплачен, агент может:
   - менять line items
   - менять note
   - менять shipping address
   - применять скидку
6. **CancelDraftOrderFlow** Пока заказ не оплачен, draft order можно отменить.
7. **CheckOrderFlow** Если клиент сам спрашивает статус или сообщает, что уже оплатил invoice, flow показывает order/payment/fulfillment state и синхронизирует `bookings` в `paid`, если заказ уже найден как оплаченный.

1.3 Что поддерживается

Возможность	Поддержка	Примечание
OAuth setup	✅	Только OAuth, без custom app path
Admin GraphQL requests	✅	Бизнес-логика только через Admin GraphQL API
Выгрузка полного каталога в attribute	✅	Через `CollectProductsFlow` в `shopify_products_catalog`
Проверка заказа	✅	Order status, payment status, fulfillment, tracking
Проверка наличия товара	✅	Product search + inventory levels
Создание draft order	✅	С сохранением в `bookings`
Обновление draft order	✅	Включая discount logic
Отмена draft order	✅	Через `draftOrderDelete`
Sync draft -> paid через ручную проверку заказа	✅	Через `CheckOrderFlow`
Customer sync flow	❌	Удалён как лишний
Store locations flow	❌	Удалён как лишний
Storefront API	❌	Полностью убран

Before You Start

2.1 Основные project attributes

- `shopify_oauth_code`
- `shopify_newo_webhook_api_key` (read-only)
- `shopify_feature_order_status_enabled`
- `shopify_feature_product_availability_enabled`
- `shopify_feature_create_draft_order_enabled`
- `shopify_feature_update_draft_order_enabled`
- `shopify_feature_cancel_draft_order_enabled`
- `shopify_feature_customer_profile_enabled`
- `shopify_feature_return_exchange_enabled`
- `shopify_feature_product_catalog_refresh_enabled`
- `shopify_shop_id`
- `shopify_client_id`
- `shopify_client_secret`
- `shopify_setup_scenarios`
- `shopify_access_token` (hidden)
- `shopify_refresh_token` (hidden)
- `shopify_base_url` (hidden)
- `shopify_api_version` (hidden)
- `shopify_products_catalog` (hidden)
- `shopify_products_catalog_buffer` (hidden)
- `shopify_products_catalog_updated_at` (hidden)
- `shopify_products_catalog_update_interval` (hidden)

OAuth Redirect URL зафиксирован в коде как `https://static.newo.ai/oauth/index.html` и не настраивается через project attribute.

2.2 Рекомендуемые Shopify scopes

- `read_customers`
- `read_orders`
- `read_all_orders`
- `read_products`
- `read_inventory`
- `read_draft_orders`
- `write_draft_orders`
- `read_discounts`

3. Архитектура

4. Обзор flow

4.3 CollectProductsFlow

Что делает

- выгружает весь каталог товаров через Shopify Admin GraphQL `products`
- пагинирует до конца каталога
- нормализует продукты и варианты
- сохраняет результат в `shopify_products_catalog`

- обновляет `shopify_products_catalog_updated_at`

Trigger

- `session_started`
- `shopify_collect_products`
- `shopify_collect_products_webhook`

Ключевые attributes

- `shopify_products_catalog`
- `shopify_products_catalog_updated_at`
- `shopify_products_catalog_update_interval`

4.4 ProductAvailabilityFlow

Что делает

- принимает `product_search_query`
- ищет продукты
- получает inventory levels
- возвращает остатки по локациям

Trigger

- `shopify_product_availability_event`

Webhook для теста

- `shopify_product_availability_test_webhook`

4.5 CreateDraftOrderFlow

Что делает

- принимает customer/product input
- находит customer
- находит product / variant
- создаёт draft order
- сохраняет его в `bookings`
- выставляет `draft/pending` state
- кладёт invoice context в prompt
- просит агента сообщить клиенту invoice link и факт, что оплата ещё pending

Trigger

- `shopify_draft_order_event`

Webhook для теста

- `shopify_create_draft_order_test_webhook`

4.6 UpdateDraftOrderFlow

Что делает

- находит существующий draft order
- подтягивает текущие line items / note / shipping
- обновляет draft order

- при необходимости применяет discount в рамках того же flow
- обновляет запись в `bookings`
- сохраняет invoice link и pending status

Trigger

- `shopify_update_draft_order_event`

Webhook для теста

- `shopify_update_draft_order_test_webhook`

4.7 CancelDraftOrderFlow

Что делает

- находит нужный draft order
- удаляет его через Shopify
- удаляет запись из `bookings`
- обновляет prompt context, чтобы invoice link больше не использовался

Trigger

- `shopify_cancel_draft_order_event`

Webhook для теста

- `shopify_cancel_draft_order_test_webhook`

4.8 CheckOrderFlow

Что делает

- ищет Shopify customer по email/phone
- получает последние заказы
- возвращает:
  - financial status
  - fulfillment status
  - total
  - tracking
- если находит оплаченный заказ с нашим `booking_reference`, обновляет `bookings` в `paid`

Trigger

- `shopify_order_status_event`

Webhook для теста

- `shopify_check_order_test_webhook`

5. Custom tools и Canvas

Сейчас агенту регистрируются только user-facing tools, которые реально нужны текущему pipeline:

- `shopify_product_availability_event`
- `shopify_draft_order_event`
- `shopify_update_draft_order_event`
- `shopify_cancel_draft_order_event`

- `shopify_order_status_event`

Canvas scenarios тоже сокращены под эту же модель:

- product availability
- draft order operations
- order status

How to test that everything works

Минимальный smoke path:

1. `shopify_product_availability_test_webhook`
2. `shopify_create_draft_order_test_webhook`
3. проверить, что в `bookings` появилась запись со статусом `draft`
4. `shopify_update_draft_order_test_webhook`
5. оплатить invoice или использовать уже оплаченный заказ того же клиента
6. `shopify_check_order_test_webhook`
7. проверить, что запись в `bookings` стала `paid`, если Shopify уже видит оплаченный order
8. `shopify_cancel_draft_order_test_webhook` для отдельного черновика

Limitations

- HMAC-валидация Shopify webhook в этой итерации не добавлялась
- автоматическая отправка invoice отдельной Shopify mutation не реализована
- коммуникация с клиентом после создания draft order строится через `urgent_message` + `set_prompt_custom_section`
- автоматического webhook-sync из paid order больше нет; переход `draft -> paid` сейчас происходит через `CheckOrderFlow`
- `CheckOrderFlow` синкает в paid только те заказы, где note содержит наши служебные маркеры

8. Итог

Интеграция больше не является набором разрозненных Shopify CRUD flow. Сейчас это узкий и понятный pipeline:

- найти товар
- создать draft order
- вести клиента к оплате
- при необходимости обновить или отменить draft
- после оплаты перевести booking state в `paid` через `CheckOrderFlow`
- по запросу клиента показать актуальный статус заказа