# Receipts

Namazu Elements Core Features · https://namazustudios.com/docs/namazu-elements-core/receipts/

## Overview

- The Receipt API stores generic receipt objects containing basic metadata and raw receipt payloads. It supports admin/support operations (search, inspect, delete) while concrete provider-specific APIs (Apple, Google Play, Facebook, Oculus) parse and operate on the raw receipt body for user-level flows (validation, redeeming).

## Receipt model

- Description: A generic receipt that stores user purchase information.
- Fields:
  - id (String): The DB id of this receipt.
  - originalTransactionId (String): ID of the original transaction from the payment processor.
  - schema (String): Receipt provider id in reverse-DNS notation (e.g. com.company.platform).
  - user (User): The user associated with this receipt.
  - purchaseTime (long): Purchase time in ms since Unix epoch.
  - body (String): String representation of the raw receipt data (JSON).

## Intended usage

- General Receipt API (admin/support): manage receipts, search by schema, view raw body, delete.
- Concrete provider APIs (user-level): detect provider by receipt.schema, parse JSON in body into provider-specific models (e.g. AppleIapReceipt, GooglePlayIapReceipt, OculusIapReceipt) and perform validation/redeem operations. See Application Configurations for more information on how to map a SKU or product id to an Item in Elements.

## DAO interface (ReceiptDao)

- Key constants:
  - RECEIPT_CREATED = "dev.getelements.elements.sdk.model.dao.receipt.created"
- Important methods:
  - Pagination getReceipts(User user, int offset, int count, String search)
    - Use search to filter by schema when supporting multiple IAP providers.
  - Pagination getReceipts(User user, int offset, int count)
  - Receipt getReceipt(String id)
  - Receipt getReceipt(String schema, String originalTransactionId)
  - Receipt createReceipt(Receipt receipt)
    - Throws InvalidDataException or DuplicateException if invalid/duplicate.
    - Emits RECEIPT_CREATED event (see events below).
  - void deleteReceipt(String receiptId)

### Transaction usage (recommended)

- Always prefer DAO operations inside a transaction to ensure consistency and automatic retry on failure.
- Example:

```
@Inject
private Provider<Transaction> transactionProvider;

final var createdReceipt = transactionProvider.get().performAndClose(tx -> {
    final var receiptDao = tx.getDao(ReceiptDao.class);
    return receiptDao.createReceipt(receipt);
});
```

### Events

- RECEIPT_CREATED fires when any new receipt is created.

- Events for specific receipt types are also fired if you only want to handle a specific type and receive the converted receipt for that type (i.e. the parsed raw receipt JSON). These include:
  - OCULUS_IAP_RECEIPT_CREATED
  - APPLE_IAP_RECEIPT_CREATED
  - GOOGLE_PLAY_IAP_RECEIPT_CREATED
  - FACEBOOK_IAP_RECEIPT_CREATED
- Event parameters variants:
  - (Receipt)
  - (Receipt, Transaction)
- Use this event for downstream processing (e.g., kicking off parsing, validation, fulfillment). Event handlers should inspect receipt.schema to determine provider-specific parsing. See Events for more information on how to register event callbacks and create your own custom events.

### Provider mapping (existing)

- Use schema to detect provider:
  - GOOGLE_IAP_SCHEME -> GooglePlayIapReceipt
  - OCULUS_PLATFORM_IAP_SCHEME -> OculusIapReceipt
  - APPLE_IAP_SCHEME -> AppleIapReceipt
  - FACEBOOK_IAP_SCHEME -> FacebookIapReceipt
- Concrete API implementations parse the JSON in Receipt.body into the corresponding provider model and provide validation/redeem endpoints.

## Integrating a new payment provider

If you want to add support for a payment provider or are creating a custom Element for your own solution, here are some steps to follow:

- Select a schema string
  - Use reverse-DNS notation: e.g. com.example.myprovider
  - Add a constant: String MYPROVIDER_IAP_SCHEME = "com.example.myprovider";

- Define a provider-specific receipt model
  - Create a POJO class that represents the JSON structure of the provider's receipt payload (e.g., MyProviderIapReceipt).
  - Add @Schema annotations for clarity in client side generated code.
- Implement parsing and validation logic
  - Add a parser that takes Receipt.body (string JSON) and maps to MyProviderIapReceipt.
  - Implement validation routines (signature checks, timestamps, server-to-server verification endpoints, etc.) according to provider docs.
- Add concrete API endpoints / service
  - Provide user-level APIs to:
    - Submit a receipt for validation/redeem.
    - Query validated receipts for a user.
  - These APIs should:
    - Locate the generic Receipt (by schema + originalTransactionId or by DB id).
    - Parse Receipt.body to MyProviderIapReceipt.
    - Validate and execute fulfillment (redeem digital goods, record consumptions).

Hint

With dev.getelements.elements.auth.enabled=true set in your Element attributes, you can inject the UserService and then use userService.getCurrentUser() to get the User that made the request. See the example-element project for an example of this.

- Add admin API endpoints/service
  - Provide superuser-level APIs to:
    - Map product ids or SKUs to Items in Elements
    - View receipts for all users
- Persist receipts via DAO

- When creating receipts (e.g., incoming from client or server webhook), use ReceiptDao.createReceipt(...) inside a transaction.
- Example create flow:

Example:

```
Receipt r = new Receipt();
r.setOriginalTransactionId(providerTxId);
r.setSchema(MYPROVIDER_IAP_SCHEME);
r.setUser(user);
r.setPurchaseTime(System.currentTimeMillis());
r.setBody(rawJson);

final var saved = transactionProvider.get().performAndClose(tx -> {
    return tx.getDao(ReceiptDao.class).createReceipt(r);
});
```

- Handle duplicates and errors
  - createReceipt may throw DuplicateException if schema + originalTransactionId already exist.
  - Catch and handle InvalidDataException for malformed receipts.
  - ReceiptDao handles idempotency: on retries, it will detect duplicates and return the existing receipt.
- Subscribe to RECEIPT_CREATED (optional)
  - Use the event to trigger asynchronous parsing/validation or analytics.
  - Event handlers should be resilient: validate schema, parse body, and record results; do not assume body is well-formed.

### Best practices and recommendations

- Always validate schema value before parsing.
- Keep raw body immutable in DB; parse into separate model objects for business logic and caching parsed values if needed.
- Use transactions around DAO writes to ensure consistency.
- Enforce idempotency in validation/redeem flows based on originalTransactionId + schema.

- Log parsing and validation failures, and surface helpful errors to support team via admin tools.
- When adding new providers, document the provider schema constant, payload model, parsing rules, and any external verification endpoints.

Example: reading a receipt and parsing per schema

```
ReceiptDao dao = tx.getDao(ReceiptDao.class);
Receipt receipt = dao.getReceipt("com.example.myprovider", originalTransactionId);

switch (receipt.getSchema()) {
    case MYPROVIDER_IAP_SCHEME:
        MyProviderIapReceipt parsed = myProviderParser.parse(receipt.getBody());
        myProviderValidator.validate(parsed);
        // proceed with redeem/fulfill
        break;
    case APPLE_IAP_SCHEME:
        AppleIapReceipt apple = appleParser.parse(receipt.getBody());
        // ...
        break;
    // other cases
}
```

### Support/admin considerations

- The built in Elements CMS will parse the OpenAPI spec for your custom Element and generate UI for it. This allows you to interact with the endpoints, manage receipts, and create your SKU to Item mappings without writing any frontend code. That said, you may want to:
  - Allow searching/filtering by schema to focus on a specific provider’s receipts.
  - Provide the ability to re-trigger parsing/validation from admin tools when needed (Items can be added to a User's inventory manually through the CMS as well).

## FAQ (short)

- Q: Where to store parsed provider models?
  - A: Store parsed results in separate tables/models; keep Receipt.body as the immutable raw source.

- Q: How to handle provider changes to payload format?
  - A: Version parsed models, add migration/compatibility logic, and keep raw JSON to reparse historical receipts if necessary.
- Q: When should I use the concrete APIs vs the general Receipt API?
  - A: Use concrete APIs for user-level validation/redeem flows. Use the general Receipt API for admin/support and cross-provider management.