VectorShift Integration Backend Documentation

Overview

The VectorShift Integration API is a FastAPI-based backend that supports OAuth flows and data retrieval for multiple third-party integrations: Airtable, Notion, and HubSpot. The API uses Redis as a temporary data store (for CSRF state tokens and credentials) and includes comprehensive logging for both application events and integration-specific activities.

The backend is designed to be modular:

- Each integration has its own module (e.g., `airtable.py`, `notion.py`, and `hubspot.py`) that implements the OAuth flow and data retrieval.
- Common functionality, such as state management and credential storage, is provided via utility functions in `utils.py` and a shared base class in `base_integration.py`.
- A centralized logging system is implemented in `logger.py` to facilitate debugging and event tracking.

Setup and Installation

1. **Install Dependencies**
   Ensure you have Python 3.8+ installed. Install required packages using pip:
   bash

   ```
   pip install fastapi uvicorn httpx redis kombu
   ```

2. **Redis Instance**
   Spin up a Redis server instance (e.g., via `redis-server`).

3. **Run the Application**
   In the backend directory, run:
   `uvicorn main:app --reload`

4. **Client Credentials**
   The Airtable and Notion integrations have redacted client information. For testing, create your own client credentials for each integration (especially for HubSpot, which was part of the assessment).

Project Structure

- **main.py**
  The FastAPI application entry point. It sets up the API routes for each integration, a health-check endpoint, and startup/shutdown events.
- **logger.py**
  Contains logging configuration. Provides helper functions for logging requests, errors, and integration-specific events. Log files are written both to the console and to rotating file handlers.
- **redis_client.py**
  Implements an asynchronous Redis client for storing temporary data (state tokens, OAuth credentials). Includes helper functions to add, retrieve, and delete key–value pairs.
- **utils.py**
  Provides common utility functions such as:
  - Generating and storing a state token for OAuth flows.
  - Validating state tokens.
  - Exchanging authorization codes for access tokens.
  - Building a Basic Authentication header.
  - Recursively searching dictionaries (used in integration item parsing).
  - Storing and retrieving credentials in Redis.
- **airtable.py**
  Implements the Airtable integration:
  - **OAuth Flow**: `authorize_airtable` generates an authorization URL (with state and PKCE values) and stores temporary data in Redis. `oauth2callback_airtable` validates the state, exchanges the code for tokens, and stores credentials.
  - **Data Retrieval**: `get_items_airtable` fetches bases and tables from Airtable using the stored access token, creates IntegrationItem objects, and aggregates them.
- **notion.py**
  Implements the Notion integration:
  - **OAuth Flow**: `authorize_notion` creates a state object, stores it in Redis, and builds an authorization URL. `oauth2callback_notion` validates state, exchanges the code for a token, and stores credentials.
  - **Data Retrieval**: `get_items_notion` performs a search query to retrieve items from Notion and maps them into IntegrationItem objects.

- **hubspot.py**
  Implements the HubSpot integration by extending the common functionality provided in the base class:
  - **OAuth Flow**: Methods `authorize_hubspot`, `oauth2callback_hubspot`, and `get_hubspot_credentials` delegate to the shared OAuth flow implemented in `BaseIntegration` (see below).
  - **Data Retrieval**: `get_items_hubspot` calls HubSpot API endpoints (contacts, companies, deals) concurrently, processes the responses, and creates IntegrationItem objects.
- **base_integration.py**
  Provides an abstract `BaseIntegration` class that contains shared methods for:
  - Initiating the OAuth flow (`authorize`).
  - Handling OAuth callback (`oauth2callback`).
  - Retrieving stored credentials (`get_credentials`).
  - Logging integration events (using helper functions from `logger.py`).
  - A helper method (`create_integration_item`) for consistently formatting integration items.
- **integration_item.py**
  Defines the `IntegrationItem` dataclass which represents a generic integration item. It includes fields such as `id`, `type`, `name`, timestamps, URL, and additional metadata. This class is used to encapsulate data retrieved from the different integrations.

API Endpoints

Health and Basic Endpoints

- **GET /**
  Returns a health check status and the current UTC timestamp.
- **GET /healthcheck**
  A secondary health check endpoint returning a simple status.

Integration Routes

Each integration is exposed via its own router under `/integrations/<integration_name>`. For example:

Airtable Integration (`/integrations/airtable`)

- **POST /authorize**
  Initiates the Airtable OAuth flow. Accepts `user_id` and `org_id` as form data and returns the authorization URL.

- **GET /oauth2callback**
  Handles the Airtable OAuth callback. Validates the state, exchanges the code for tokens, and returns an HTML response that closes the browser window.
- **POST /credentials**
  Retrieves stored Airtable credentials for a given user and organization.
- **POST /load**
  Loads and returns Airtable integration items using the provided credentials.

Notion Integration (`/integrations/notion`)

- **POST /authorize**
  Starts the Notion OAuth flow; similar to Airtable, this endpoint creates and stores the state and returns an authorization URL.
- **GET /oauth2callback**
  Processes the Notion OAuth callback, validating state and exchanging the authorization code.
- **POST /credentials**
  Retrieves stored Notion credentials.
- **POST /load**
  Retrieves integration items from Notion based on the provided credentials.

HubSpot Integration (`/integrations/hubspot`)

- **POST /authorize**
  Initiates the HubSpot OAuth flow using the shared `BaseIntegration` logic.
- **GET /oauth2callback**
  Processes the OAuth callback for HubSpot, including state validation and token exchange.
- **POST /credentials**
  Retrieves HubSpot credentials stored during the OAuth flow.
- **POST /load**
  Loads HubSpot integration items (contacts, companies, deals) by calling HubSpot endpoints concurrently and mapping results to IntegrationItem objects.

OAuth Flow and Credential Management

1. **State Generation and Validation**
   - The functions `generate_and_store_state` and `validate_state` in `utils.py` create a secure, random state token for each OAuth request.
   - The state is stored in Redis with an expiration time and is validated during the callback to protect against CSRF attacks.
2. **Token Exchange**
   - The `exchange_code_for_token` function in `utils.py` handles the exchange of an authorization code for an access token using integration-specific endpoints and client credentials.

    - A Basic Authentication header is generated using `get_basic_auth_header`.
3. **Credential Storage**
    - After token exchange, credentials (access tokens and other details) are stored in Redis with a short expiration period.
    - The retrieval functions (`get_airtable_credentials`, `get_notion_credentials`, and `get_hubspot_credentials`) extract and then delete the stored credentials for one-time use.

Logging and Error Handling

- **Centralized Logging**
    - The `logger.py` module sets up a centralized logging system using Python's `logging` module.
    - Loggers are created for general application events as well as integration-specific events (e.g., HubSpot, Notion, Airtable).
- **Error Logging**
    - Functions such as `log_error` and `log_integration_event` are used throughout the code to capture and log errors and significant events.
    - Detailed logs include context (user IDs, org IDs, integration name, and operation details) to facilitate debugging.
- **Rotating File Handlers**
    - The logging setup includes rotating file handlers to manage log file sizes and backup counts, ensuring that logs remain manageable over time.

Integration Item Data Model

- The `IntegrationItem` class (in `integration_item.py`) is a data class that represents a generic item retrieved from any integration.
- Common attributes include:
    - `id`: Unique identifier.
    - `type`: The category of the item (e.g., contact, table, base).
    - `name`: A display name.
    - `creation_time` and `last_modified_time`: Timestamps.
    - `url`: Link to the item in the third-party system.
    - Additional metadata (e.g., parent item information, MIME type, and visibility).

This standardization enables the frontend to handle items uniformly regardless of the originating integration.

VectorShift Integration Frontend Documentation

Overview

The frontend is a React-based application built with Material UI that allows users to connect to third-party integrations via OAuth. Users enter their user and organization details, select an integration type, and trigger the OAuth flow. Once authenticated, the application fetches and displays integration items from the backend API. This interface is part of the complete assignment alongside the FastAPI backend.

Setup and Installation

Install Dependencies
Navigate to the `/frontend` directory and install dependencies : `npm install`

1. **Run the Application**
   Start the React development server : `npm run start`
2. The app will typically run on http://localhost:3000.
3. **Backend and Redis Requirements**
   Ensure the backend (FastAPI) server is running (e.g., via `uvicorn main:app --reload` in the `/backend` directory) and a Redis instance is active (`redis-server`), as the frontend relies on these services.

Project Structure

- **App.js**
  The root component of the application. It wraps the main `IntegrationForm` component.

- **index.js**
  The entry point that bootstraps the React app by rendering the `App` component into the DOM.

- **index.css**
  Contains global CSS styles and layout definitions to ensure a clean, responsive UI design.

- **integration-form.js**
  The central component that handles user input (user ID and organization), integration

type selection (via an Autocomplete component), and dynamic rendering of integration-specific components (e.g., Airtable, Notion, HubSpot). It also includes the `DataForm` component for loading and displaying integration data.

- **Integration Components**
  These components handle the OAuth process for their respective integrations:
  - **airtable.js**
    Manages the Airtable OAuth flow by requesting an authorization URL from the backend, opening a new window for user authentication, polling for the OAuth window's closure, and fetching OAuth credentials afterward.
  - **notion.js**
    Implements the Notion OAuth flow with similar logic as Airtable—initiating the flow, opening a new window, polling for closure, and then retrieving credentials from the backend.
  - **hubspot.js (Expected)**
    Although not provided in the current file set, the HubSpot integration should follow the same pattern. The integration is referenced in the integration form mapping so that it is accessible in the UI.
- **DataForm Component (data-form.js)**
  Offers a user interface for loading and displaying data from the connected integration. Upon clicking "Load Data," it sends the stored credentials to the backend, retrieves integration items, and displays them. It also provides an option to clear the displayed data.

- **Logger Service**
  Throughout the frontend, a logging service (referenced as `logger`) is used to record user interactions and API calls, providing insight into component behavior and aiding in debugging.

Component Functionality

Integration Form (integration-form.js)

- **User & Organization Inputs**:
  Users provide their unique identifiers and organization name. These inputs are used in the OAuth requests to correctly associate the credentials with the user's account.
- **Integration Type Selection**:
  An Autocomplete component lists available integration types (Airtable, Notion, and HubSpot). When an integration is selected, the corresponding integration component is rendered dynamically.
- **Dynamic Integration Components**:
  Depending on the integration type selected:

    - The **AirtableIntegration** or **NotionIntegration** component is rendered to handle OAuth.
    - Once OAuth is complete and credentials are retrieved, the connection state is updated, and the integration type is stored in the application state.
- **Data Loading**:
  After successful authentication, the `DataForm` component is displayed. This component enables the user to load integration items by sending the stored credentials to the backend API. The loaded data is then displayed within the form.

Integration-Specific Components

- **AirtableIntegration (airtable.js)**:
    - **OAuth Flow**:
      When the "Connect to Airtable" button is clicked, a POST request is sent to the backend endpoint `/integrations/airtable/authorize` with the user and organization details. The received authorization URL is opened in a new window for OAuth processing.
    - **Polling Mechanism**:
      The component polls every 200ms to detect when the OAuth window is closed.
    - **Credential Retrieval**:
      Once the OAuth window closes, the component calls `/integrations/airtable/credentials` to fetch the OAuth credentials and updates the UI state to reflect a successful connection.

- **NotionIntegration (notion.js)**:
    - **OAuth Flow**:
      Similar to the Airtable component, this component sends a POST request to `/integrations/notion/authorize` and opens the provided authorization URL in a new window.
    - **Polling and Credential Fetching**:
      After detecting the window closure, it fetches the Notion credentials by calling `/integrations/notion/credentials` and updates the integration state.

- **HubSpotIntegration**:
  While the frontend file for HubSpot is expected to follow the same approach as Airtable and Notion, it is integrated into the mapping in `integration-form.js` and should be implemented in a similar manner to enable OAuth authentication and data loading for HubSpot.

Data Loading (data-form.js)

- **Load Data Functionality**:
  The `DataForm` component provides a “Load Data” button. On clicking, it sends the OAuth credentials (as a JSON string) to the appropriate backend endpoint (determined by the integration type) to load integration items.

- **Display & Clear Data**:
  Retrieved data is shown in a disabled text field. A “Clear Data” button is available to remove the displayed data.
- **Logging**:
  User interactions and API call durations are logged for monitoring and debugging purposes.

Styling and Layout

- **Global Styles (index.css)**:
  The global CSS file defines styles for:
  - **Layout and Containers**:
    Ensuring a responsive and centered layout with a maximum width.
  - **Background Effects**:
    Circular background elements and shadow effects add visual depth.
  - **Component Specific Styling**:
    Styles for text fields, buttons, and form components are defined to provide a consistent look and feel across the application.

Application Entry Point

- **index.js**:
  Initializes the React application and renders the `App` component into the root DOM element.

- **App.js**:
  A simple component that encapsulates the `IntegrationForm`, serving as the main entry point for the frontend integration functionality.

Workflow Summary

1. **User Input & Integration Selection**:
   - The user enters their user ID and organization name.
   - The user selects an integration (Airtable, Notion, or HubSpot) via the Autocomplete component.
2. **Initiating OAuth Flow**:

    - The selected integration component sends a request to the backend to obtain an authorization URL.
    - A new window opens for the user to complete the OAuth flow.
    - The component polls for the window to close, indicating that the OAuth process is complete.
3. **Credential Retrieval & State Update**:
    - After the OAuth window closes, the component fetches the credentials from the backend.
    - The connection state is updated, and the integration type is stored in the application state.
4. **Data Loading**:
    - With valid credentials, the `DataForm` component becomes active.
    - The user can load integration items by triggering a backend API call, and the retrieved data is then displayed.
5. **Logging & Monitoring**:
    - All significant interactions (e.g., initiating OAuth, loading data, clearing data) are logged for debugging and analytical purposes.