sssync.app: Use Case Driven Component Map (NestJS / TS/JS Version)

This document reframes the component architecture for sssync.app using NestJS conventions, targeting the provided Supabase schema.

Use Case 1: Platform Connection (OAuth) & Verification

- **Goal:** Securely connect and verify user's external platform accounts (Shopify, Clover first) using OAuth 2.0, storing credentials (encrypted), and handling token refresh.
- **Components by Layer:**

Layer	Component Type	Component Name(s)	Purpose in this Flow (NestJS / TS/JS)
API	Controller	PlatformsController	Handles HTTP requests via decorators (@Get, @Post, etc.) for initiating OAuth flow and receiving callbacks. Uses DTOs (Data Transfer Objects) for request/response validation.
	Endpoint	@Get('connect/initiate/:platformType')	Triggers the start of the OAuth flow by calling the PlatformsService.
	Endpoint	@Get('connect/callback/:platformType')	Receives the redirect from the platform after user authorization. Validates state, calls PlatformsService to handle the callback.

	Endpoint	@Get()	(Implied) Lists connections for the authenticated user. Calls PlatformsService.
	Endpoint	@Get(':connectionId/status')	(Implied) Checks connection status. Calls PlatformsService.
	Endpoint	@Delete(':connectionId')	(Implied) Deletes a connection. Calls PlatformsService.
Application	Guard	AuthGuard (e.g., JwtAuthGuard)	Verifies user identity (JWT) via request headers, protecting relevant endpoints.
	Service	PlatformsService	Contains core business logic. Generates platform-specific authorization URLs. Handles OAuth callbacks: verifies state, exchanges code for tokens (using PlatformApiClient), encrypts tokens (using EncryptionService), saves/updates connection (using PlatformConnectionRepository). Manages listing, status checks, and deletion.

	Service Interface	PlatformApiClient (Abstract Class/Interface)	Defines the contract for platform-specific interactions: exchanging auth code for tokens, verifying connection status, refreshing tokens. Implemented by platform-specific services (e.g., ShopifyApiClient, CloverApiClient).
	Service Interface	EncryptionService (Abstract Class/Interface)	Defines the contract for encrypting/decrypting sensitive data like OAuth tokens.
	Repository Interface	PlatformConnectionRepository (Abstract Class/Interface)	Defines the contract for data access operations (CRUD) on the PlatformConnections table.
	DTO	InitiateConnectionDto, CallbackDto, ConnectionStatusDto, PlatformConnectionDto	Defines the shape of data transferred between layers (e.g., request bodies, responses). Used with validation pipes.
Infrastructure	Service Implementation	ShopifyApiClient, CloverApiClient	Implements PlatformApiClient. Handles HTTP calls (axios, Workspace, or NestJS HttpModule) to platform token endpoints. Implements token refresh logic.

	Service Implementation	EncryptionService	Implements EncryptionService, possibly using Node.js crypto module or external libraries.
	Repository Implementation	SupabasePlatformConnectionRepository	Implements PlatformConnectionRepository using the Supabase client library (@supabase/supabase-js) to interact with the PlatformConnections table.
	Module	PlatformsModule	Encapsulates controllers, services, providers related to platform connections. Imports necessary modules (e.g., AuthModule, SupabaseModule).
	Background Task	TokenRefreshService (Optional/Later)	Could use NestJS Schedule (@nestjs/schedule) or a job queue (e.g., Bull) to proactively refresh tokens. For MVP, refresh might be "just-in-time" within PlatformApiClient implementations.
Domain	Entity/Model	PlatformConnection	Represents the PlatformConnections table structure, often defined as a class or interface mirroring the Supabase

			schema. Used by the repository and service.
	Entity/Model	User	Represents the Users table, linking the connection to a specific user.

- **Database Interaction (Supabase):**
  - PlatformConnections: Write (create/update connection, store encrypted credentials, status), Read (list connections, check status, get credentials for refresh/verification). Uses supabase-js.
  - Users: Read (to associate connection). Uses supabase-js.
  - ActivityLogs: Write (log connection success/failure/refresh events). Uses supabase-js.

Use Case 2: Pulling/Pushing Data (Migration & Cross-Posting)

- **Goal:** Import existing product data from connected platforms into sssync (Migration) and push product data from sssync to connected platforms (Cross-Posting Create/Update/Delete).
- **Components by Layer:**

Layer	Component Type	Component Name(s)	Purpose in this Flow (NestJS / TS/JS)
API	Controller	ProductsController, PlatformsController	Handles HTTP requests for listing platform products, importing products, posting sssync products to platforms, and

			deleting platform listings.
	Endpoint	@Get('/platforms/:connectionId/products')	Lists products available for import from a specific platform. Calls PlatformsService.
	Endpoint	@Post('/platforms/:connectionId/products/import/:platformProductId')	Triggers the import of a specific product. Calls PlatformsService or a dedicated MigrationService.
	Endpoint	@Post('/products/:variantId/platforms/:connectionId')	Pushes (creates/updates) an sssync product variant listing onto a specific platform. Calls ProductsService.
	Endpoint	@Delete('/products/:variantId/platforms/:connectionId')	Deletes a product listing from a specific platform. Calls ProductsService.

Application	Service	PlatformsService, ProductsService, (MigrationService?)	Contains business logic. PlatformsService lists platform products (using PlatformApiClient). ProductsService (or MigrationService) handles import (gets details via PlatformApiClient, maps data, saves using Repositories), posting (maps sssync data, pushes via PlatformApiClient, saves mapping), and deletion (deletes via PlatformApiClient, removes mapping).
	Service Interface	PlatformApiClient (Abstract Class/Interface)	Extends contract to include listing products, getting details, creating/updating/deleting products on external platforms.

	Repository Interface	ProductRepository, ProductVariantRepository, ProductImageRepository, PlatformProductMappingRepository, InventoryLevelRepository	Defines contracts for CRUD operations on respective Supabase tables.
	DTO	PlatformProductDto, ImportProductDto, PostProductDto, ProductVariantDto, ProductMappingDto etc.	Defines data shapes for requests, responses, and internal service calls.
Infrastructure	Service Implementation	ShopifyApiClient, CloverApiClient, etc.	Implements PlatformApiClient methods for product operations using platform APIs and authentication.
	Repository Implementation	SupabaseProductRepository, SupabaseProductVariantRepository, etc.	Implements data access interfaces using supabase-js for product, variant, image, mapping, inventory tables.
	Module	ProductsModule, PlatformsModule	Encapsulates related controllers, services, providers.

Domain	Entity/Model	Product, ProductVariant, ProductImage, PlatformConnection, PlatformProductMapping, InventoryLevel	Represent Supabase tables involved in migration, posting, and mapping.
	Service (Optional)	ProductMappingService	Could encapsulate complex logic for mapping data between sssync and various platform formats if needed.

- **Database Interaction (Supabase):**
  - PlatformConnections: Read (get credentials, platform type).
  - Products, ProductVariants, ProductImages: Read (for Cross-Posting), Write (for Migration).
  - PlatformProductMappings: Read (check existence), Write (create/update for Post/Import), Delete (for Delete).
  - InventoryLevels: Write (set initial level during Migration).
  - ActivityLogs: Write (log migration/posting/deletion events).

Use Case 3: Inventory Sync

- **Goal:** Automatically synchronize inventory levels across all connected platforms when a change occurs.
- **Components by Layer:**

Layer	Component Type	Component Name(s)	Purpose in this Flow (NestJS / TS/JS)
API	Controller	WebhooksController	Handles incoming webhook requests from

			platforms (e.g., `orders/created`, `inventory_levels/update`).
	`Endpoint`	`@Post(':platformType/:event')`	Receives webhook payloads. Verifies signatures (via Guard/Pipe or in service). Queues processing (e.g., using BullMQ or calls service directly).
	`Guard/Pipe`	`WebhookSignatureGuard` (or similar)	(Optional but recommended) Verifies the authenticity of incoming webhooks before hitting the controller logic.
Application	`Service`	`WebhookHandlerService`, `InventorySyncService`	`WebhookHandlerService` parses payloads, identifies affected `ProductVariant`(s) via `PlatformProductMappingRepository`, calculates changes. It triggers `InventorySyncService` to update other platforms and the sssync `InventoryLevel`. `InventorySyncService` uses `PlatformApiClient` to push updates and `InventoryLevelRepository` to update sssync records. Logs actions via `ActivityLogRepository`.

	Queue/Events	(Optional: e.g., BullMQ, Kafka, NestJS Events)	Decouples webhook reception from processing. The controller adds a job/event, and a separate processor/listener handles it by calling the appropriate service.
	Service Interface	PlatformApiClient (Abstract Class/Interface)	Extends contract for reading order details (if needed) and updating inventory levels on platforms.
	Repository Interface	PlatformProductMappingRepository, InventoryLevelRepository, ActivityLogRepository	Defines contracts for reading mappings, reading/writing inventory levels, and writing logs.
	DTO	WebhookPayloadDto, InventoryUpdateDto	Defines shapes for webhook payloads and inventory update data.
Infrastructure	Service Implementation	ShopifyApiClient, CloverApiClient, etc.	Implements PlatformApiClient methods for inventory updates and potentially order retrieval. Handles authentication.
	Repository Implementation	SupabasePlatformProductMappingRepository, SupabaseInventoryLevelRepository, SupabaseActivityLogRepository	Implements data access using supabase-js.

	Module	WebhooksModule, InventoryModule	Encapsulates related components.
	Background Service	PollingSyncService (Alternative)	Could use NestJS Schedule (@nestjs/schedule) to periodically poll platform APIs via PlatformApiClient if webhooks are unreliable/insufficient. Triggers sync logic in InventorySyncService.
Domain	Entity/Model	InventoryLevel, PlatformProductMapping, PlatformConnection, ProductVariant, Order, OrderItem, ActivityLog	Represent relevant Supabase tables.
	Service (Optional)	InventorySyncRuleService	Could encapsulate complex rules for conflict resolution.

- **Database Interaction (Supabase):**
  - PlatformConnections: Read (get credentials, identify platforms).
  - PlatformProductMappings: Read (find sssync variant from platform ID/SKU).
  - ProductVariants: Read (identify the master item).
  - InventoryLevels: Read (current levels), Write (update levels after change/sync).
  - Orders, OrderItems: Write (if logging received orders internally) or Read (if polling).
  - ActivityLogs: Write (log every sync action, success, failure, webhook receipt).

Use Case 4: AI Listing Creation

- **Goal:** Create a product in sssync using an image URL, potentially leveraging external APIs like SerpApi/Google Lens and a generative AI like Gemini.
- **Components by Layer:**

Layer	Component Type	Component Name(s)	Purpose in this Flow (NestJS / TS/JS)
API	Controller	ProductsController	Handles HTTP request to trigger AI creation.
	Endpoint	@Post('create-with-ai')	Receives image URL (or image upload) in the request body (using a DTO). Triggers the creation process by calling AiProductCreationService.
Application	Service	AiProductCreationService	Orchestrates the flow: calls external APIs (SerpApiClient, GeminiApiClient), processes results, creates Product, ProductVariant entities, saves AiGeneratedContent, and persists all using respective Repositories.
	Service Interface	SerpApiClient, GeminiApiClient (Abstract Classes/Interfaces)	Defines contracts for interacting with the specific external AI/Search APIs.

	Repository Interface	ProductRepository, ProductVariantRepository, AiGeneratedContentRepository	Defines contracts for saving the newly created product, variant, and AI-generated text data.
	DTO	CreateWithAiDto, AiProductResultDto	Defines shapes for the incoming request and the structured data returned by the AI service.
Infrastructure	Service Implementation	HttpSerpApiClient, HttpGeminiApiClient	Implements the API client interfaces using axios, Workspace, or NestJS HttpModule. Handles authentication (API Keys).
	Repository Implementation	SupabaseProductRepository, SupabaseProductVariantRepository, SupabaseAiGeneratedContentRepository	Implements data access using supabase-js.
	Module	ProductsModule, AiModule	Encapsulates related components.
Domain	Entity/Model	Product, ProductVariant, AiGeneratedContent	Represent the product being created and the associated AI text, mirroring Supabase tables.

- **Database Interaction (Supabase):**
    - Products: Write (create new product record).

- `ProductVariants`: Write (create new variant record with AI-generated title/desc, placeholder SKU/price).
- `AiGeneratedContent`: Write (store generated title, description, keywords, etc.).
- `ActivityLogs`: Write (log AI generation attempt/success/failure).

Use Case 5: Barcode Lookup

- **Goal:** Find an existing sssync product variant using its barcode.
- **Components by Layer:**

Layer	Component Type	Component Name(s)	Purpose in this Flow (NestJS / TS/JS)
API	`Controller`	`ProductsController`	Handles HTTP request for barcode lookup.
	`Endpoint`	`@Get('lookup/barcode/:barcodeValue')`	Receives barcode value from URL parameter. Triggers lookup by calling `ProductsService`.
Application	`Service`	`ProductsService`	Contains logic to lookup the product variant using `ProductVariantRepository`.
	`Repository Interface`	`ProductVariantRepository`	Defines contract including a method like `findByBarcode(userId: string, barcode: string)`.

	DTO	ProductVariantDto	Defines the shape of the returned product variant data.
Infrastructure	Repository Implementation	SupabaseProductVariantRepository	Implements findByBarcode using supabase-js filtering on Barcode and UserId.
	Module	ProductsModule	Encapsulates related components.
Domain	Entity/Model	ProductVariant	Represents the ProductVariants table data being looked up.

- **Database Interaction (Supabase):**
  - ProductVariants: Read (query using index on Barcode and UserId).

NEW: User Settings & Profile Management (Based on Schema v4 Updates)

- **Goal:** Allow users to manage their private settings and public profiles as defined in the Users and UserProfiles tables.
- **Components by Layer:**

Layer	Component Type	Component Name(s)	Purpose in this Flow (NestJS / TS/JS)
API	Controller	UsersController, ProfilesController	Handles HTTP requests for managing user settings and profiles.

	Endpoint	@Get('/users/me/settings')	(UsersController) Fetches private settings for the authenticated user. Calls UsersService. Requires AuthGuard.
	Endpoint	@Put('/users/me/settings')	(UsersController) Updates private settings for the authenticated user. Uses UserSettingsDto. Calls UsersService. Requires AuthGuard.
	Endpoint	@Get('/profiles/me')	(ProfilesController) Fetches the public profile for the authenticated user. Calls ProfilesService. Requires AuthGuard.
	Endpoint	@Put('/profiles/me')	(ProfilesController) Updates the public profile for the authenticated user. Uses UserProfileDto. Calls ProfilesService. Requires AuthGuard.
	Endpoint	@Get('/profiles/:userId')	(ProfilesController) Fetches the public profile for a specific userId. Calls ProfilesService. AuthGuard optional

			depending on visibility rules.
Application	Service	UsersService, ProfilesService	UsersService handles getting/updating user settings using UserRepository. ProfilesService handles getting/updating user profiles using UserProfileRepository.
	Repository Interface	UserRepository, UserProfileRepository	Defines contracts for CRUD operations on Users and UserProfiles tables.
	Guard	AuthGuard (e.g., JwtAuthGuard)	Protects endpoints requiring authentication.
	DTO	UserSettingsDto, UserProfileDto	Defines data shapes for settings and profile updates/responses. Used with validation pipes.
Infrastructure	Repository Implementation	SupabaseUserRepository, SupabaseUserProfileRepository	Implements data access interfaces using supabase-js for Users and UserProfiles tables, handling the new fields.

	Module	UsersModule, ProfilesModule	Encapsulates related controllers, services, providers. Imports AuthModule, SupabaseModule.
Domain	Entity/Model	User, UserProfile	Represent the Users and UserProfiles table structures.

- **Database Interaction (Supabase):**
  - Users: Read/Write (for settings, using authenticated user ID).
  - UserProfiles: Read/Write (for profiles, using authenticated user ID or specified userId).

This revised structure aligns the component roadmap with common NestJS patterns and terminology, ready for implementation with your Supabase backend. Remember to install necessary NestJS modules (@nestjs/config, @nestjs/jwt, @nestjs/passport, @nestjs/schedule, potentially @nestjs/bullmq, etc.) and the Supabase JS client (@supabase/supabase-js).