TypeScript SDK

This topic explains how to develop Kibo eCommerce applications using the TypeScript SDK.

Before You Begin

Install the following software on your local machine:

- Node.js—Provides a platform for creating scalable network applications. Includes the npm package manager, which you use in this tutorial to install the NodeJS SDK.
- You need access to a Dev Center Account that contains a provisioned sandbox.
- KiboCommerce Application ClientID / Secret.
- KiboCommerce Tenant.

Get Started

The following steps guide you through installing the TypeScript SDK, authenticating your project with your Kibo credentials, and making a call to the API. Specifically, the tutorial demonstrates how to create a console application that retrieves the number of customer accounts for a Kibo eCommerce site, teaching you the necessary concepts for then building a fully-fledged application of your own.

Create an application in Dev Center with the appropriate behaviors:

1. Log in to Dev Center.
2. Create a new application.
3. Add the `Customer Read` behavior to the application. This step is necessary to give your application the necessary permissions to read customer accounts. If you design additional functionality for your application, such as updating an order, add the appropriate behaviors to avoid a permissions error.
4. Install the application to the sandbox of your choice.
5. Enable the application in Admin. If you decide to add additional behaviors to your application after this step, you must reinstall the application to your sandbox and re-enable the application in Admin to apply the new behaviors.
6. Note the application key, shared secret, tenant ID, and site ID. You can obtain the application key and shared secret from the application details page. You can obtain the tenant ID and site ID by viewing your live site and looking at the URL, which has the pattern `t`*TenantID*`-s`*SiteID*`.sandbox.mozu.com`. You can obtain the master catalog ID through a GetTenant API call, which also returns the tenant ID and site ID, but the master catalog ID is not required for the API call used in this tutorial.

Create a TypeScript SDK Application

Follow the below steps to create a TypeScript SDK application that uses the TypeScript:

1. Create a new directory on your local machine.
2. Open a command prompt in the new directory.
3. Run `npm init` to create a `package.json` file in your directory. When prompted, provide a name for your npm package and accept the default values for the remaining prompts, making sure that the entry point for your application is `index.js`. When you build a fully-fledged application, you can customize these responses, instead of accepting the default values like you do for this tutorial.
4. Run `npm install @kibocommerce/rest-sdk` to install the TypeScript SDK.
5. Run `npm install typescript ts-node @kibocommerce/rest-sdk`.

Create an API Client

This section guides you through the process of creating an API client and setting up environment variables.

Follow the steps below to Configure Environment Variables (Optional):

1. Create a `.env` file at the root of your project and configure the required settings:

```
KIBO_LOCALE=
KIBO_TENANT=
KIBO_SITE=
KIBO_MASTER_CATALOG=
KIBO_CATALOG=
KIBO_CURRENCY=
KIBO_AUTH_HOST=
KIBO_CLIENT_ID=
KIBO_SHARED_SECRET=
KIBO_API_ENV=
```

2. Import the Kibo Configuration, CustomerAccountAPI, and `.env` (if using the `.env` file optionally).

```
// Import necessary modules
import { Configuration } from '@kibocommerce/rest-sdk';
import { CustomerAccountApi } from '@kibocommerce/rest-sdk/clients/CustomerAccount';

// Load configuration from environment variables using dotenv
import 'dotenv/config';
```

3. Create a Configuration Object.
    1. Using hardcoded values:

```
// Import necessary modules
import { Configuration } from '@kibocommerce/rest-sdk';
import { CustomerAccountApi } from '@kibocommerce/rest-sdk/clients/CustomerAccoun
t';

// Create a Configuration Object with hardcoded values
const configuration = new Configuration({
  tenantId: 26507,
  siteId: 41315,
  catalog: 1,
  masterCatalog: 1,
  sharedSecret: '12345_Secret',
  clientId: 'KIBO_APP.1.0.0.Release',
  pciHost: 'pmts.mozu.com',
  authHost: 't00000.sandbox.mozu.com',
  apiEnv: 'sandbox',
});
```

2. Optionally, Using environment variables:

```
// Import necessary modules
import { Configuration } from '@kibocommerce/rest-sdk';
import { CustomerAccountApi } from '@kibocommerce/rest-sdk/clients/CustomerAccoun
t';
import 'dotenv/config';


// Load configuration from environment variables
const configuration = Configuration.fromEnv();
```

4. Create a CustomerAccount API using your configuration and retrieve customer accounts.

```
// Import necessary modules
import { Configuration } from '@kibocommerce/rest-sdk';
import { CustomerAccountApi } from '@kibocommerce/rest-sdk/clients/CustomerAccount';
import 'dotenv/config';

// Load configuration from environment variables
const configuration = Configuration.fromEnv();

// Create an instance of the CustomerAccount API client
const client = new CustomerAccountApi(configuration);

try {
    // Attempt to retrieve customer accounts
    const response = await client.getAccounts();

    // Process the response as needed
} catch (error) {
    // Handle any errors that occur during the API request
    console.error(error);
}
```

Locating the API Clients

1. Clients are separated by different domains and should align with the Kibo API Documentation, you can find and import these under the `clients` folder.

```
import { SomeApi } from '@kibocommerce/rest-sdk/clients/*'
```

2. For example, If you are looking for APIs under the "Inventory" section of the documentation you can find the related API clients under `'@kibocommerce/rest-sdk/clients/Inventory'`

```
// Import the InventoryApi module from the SDK
import { InventoryApi } from '@kibocommerce/rest-sdk/clients/Inventory'

// Create an instance of the InventoryApi using the provided configuration (assuming 'config' i
s defined elsewhere)
const client = new InventoryApi(config)

// Attempt to retrieve inventory information for items with UPC '1234'
const resp = await client.getInventory({ items: [{ upc: '1234' }] })
```

Customizing with Middleware

Every API client supports custom middleware that can be executed before and after request execution, as well as in the event of any errors during the request.

Define a class that implements the Middleware interface and offers the configuration object.

An example use case would be request logging

```
// Import necessary modules and types from the SDK
import {
  Middleware,
  RequestContext,
  ResponseContext,
  FetchParams,
  ErrorContext,
} from '@kibocommerce/rest-sdk/types'

// Define a LoggerMiddleware class that implements the Middleware interface
export class LoggerMiddleware implements Middleware {
  // Middleware pre() method: executed before making the API request
  public async pre(context: RequestContext): Promise<FetchParams | void> {
    console.log(`sending METHOD: ${context.init.method} URL: ${context.url}`)
  }

  // Middleware post() method: executed after receiving the API response
  public async post(context: ResponseContext): Promise<Response | void> {
    console.log(`response STATUS: ${context.response.status}`)
  }

  // Middleware onError() method: executed if an error occurs during the API request
  public async onError(context: ErrorContext): Promise<Response | void> {
    console.error('logging error', context.error)
  }
}
```

```
// Import necessary modules from the SDK
import { Configuration } from '@kibocommerce/rest-sdk'
import { ProductSearchApi } from '@kibocommerce/rest-sdk/clients/CatalogStorefront'

// Create an instance of the LoggerMiddleware
const loggerMiddleware = new LoggerMiddleware()

// Create a Configuration object with specified settings and middleware
const configuration = new Configuration({
  tenantId: 26507,
  siteId: 41315,
  catalog: 1,
  masterCatalog: 1,
  sharedSecret: '12345_Secret',
  clientId: 'KIBO_APP.1.0.0.Release',
  pciHost: 'pmts.mozu.com',
  authHost: 't00000.sandbox.mozu.com',
  apiEnv: 'sandbox',
  middleware: [loggerMiddleware], // Include the logger middleware
})

// Create an instance of the ProductSearchApi using the configuration
const client = new ProductSearchApi(configuration)
```

Middleware can also be added after creating an API client.

```
// Import necessary modules from the SDK
import { Configuration } from '@kibocommerce/rest-sdk';
import { ProductSearchApi } from '@kibocommerce/rest-sdk/clients/CatalogStorefront';

// Create an instance of the LoggerMiddleware
const loggerMiddleware = new LoggerMiddleware();
// Create an instance of the ProductSearchApi using the 'configuration'
const client = new ProductSearchApi(configuration);

// Attach the logger middleware to the ProductSearchApi client
client.withMiddleware([loggerMiddleware]);
```

About the Toolkits

NPM Package Name	Description
typescript-rest-sdk	The TypeScript SDK package. View the source on GitHub

SDK Function Reference

The TypeScript SDK provides dozens of built-in functions that help you interact with the API quickly and easily. To familiarize yourself with these functions:

1. View the SDK source on GitHub, and either dig through the source files or use the repository search box to find the

function you are looking for.

2. Dig through the source files or use the repository search box (on GitHub) to find the function you are looking for. The files mirror the structure of the API.