TypeScript SDK

When developing applications, you can install the TypeScript SDK to authenticate with Kibo and send API requests using the TypeScript extension of JavaScript. This SDK has access to the full functionality of Kibo's REST APIs.

For more detailed information about application development, refer to the Application guides. This guide describes how to expand upon an application by adding an SDK.

Before You Begin

You must have the following software installed on your local machine as well as Kibo access details.

- Node.js: a platform for creating scalable network applications that includes the npm package manager.
- A Dev Center Account with a provisioned sandbox.
- An Application Key (also called the Client ID) and Secret.
- A Kibo tenant.

Get Started

This tutorial demonstrates installing the TypeScript SDK, authenticating with Kibo credentials, and making an API call. The example is of a console application that retrieves the number of customer accounts for a Kibo eCommerce site. Use these concepts to build a fully-fledged application of your own.

Create an Application

First, create an application in the 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, then add the appropriate behaviors to avoid a permissions error.
4. Install the application to a sandbox.
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. Locate your Application Key, Shared Secret, Tenant ID, and Site ID. Obtain the Application Key and Shared Secret from the application details page in the Dev Center. Obtain the Tenant ID and Site ID by viewing your site and looking at the URL, which has the format `t{TenantID}-s{SiteID}.sandbox.mozu.com`.
    - You can also use the Get Tenant API call to return the Site ID and a Master Catalog ID if needed. The Master Catalog ID is not required for the particular API call used in this tutorial.

Install the TypeScript SDK

Follow the below steps to install the TypeScript SDK:

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

You should now create an API client and decide whether to use hardcoded configuration values or environment variables.

1. If you want to use environment variables, create a `.env` file at the root of your project and enter your credentials and any other required values such as Catalog IDs.

    ```
    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 and Customer Account API packages, as well as `.env` configurations if you are using environment variables.

```
// 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 using either hardcoded values or environment variables.
    - 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',
      });
      ```

    - 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 Customer Account API client using your configurations to 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);
}
```

Other API Clients

API clients are separated by domain, similar to the organization in the Kibo API documentation. Find and import these from the `clients` folder with the following command:

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

For example, if you are looking for Inventory APIs then you can find the related API clients under `'@kibocommerce/rest-sdk/clients/Inventory'` and import with the following code:

```
// 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' is def
ined elsewhere)
const client = new InventoryApi(config)

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

Customize with Middleware

Every API client supports custom middleware that can be executed before and after the API request, as well as in the event of any errors during the request. Do this by defining a class that implements the Middleware interface and refers to the Configuration object, such as in this example for 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 an API client has been created:

```
// 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]);
```

SDK Functions

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, 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.