Kibo Commerce Java SDK

This topic explains how to develop Kibo eCommerce Java SDK modules generated from Kibo Commerce API documentation.

Get Started

The following steps guide you through installing the Kibo eCommerce Java SDK, authenticating your project with your Kibo eCommerce credentials, and making a call to the Kibo eCommerce 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 a Kibo eCommerce application in Dev Center with the appropriate behaviors:

1. Log in to Dev Center.
2. Create a new Kibo eCommerce 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.
7. Using the terminal, generate an example project using Maven:

```
mvn archetype:generate -DarchetypeArtifactId="maven-archetype-quickstart" -DarchetypeGr
oupId="org.apache.maven.archetypes" -DarchetypeVersion="1.4" -DgroupId="com.example
" -DartifactId="kibodemo"
```

8. Open the new pom.xml file and add the Kibo Commerce “customer” module as a dependency.

```
<dependencies>
  <dependency>
    <groupId>com.kibocommerce</groupId>
    <artifactId>customer</artifactId>
    <version>2.0.0</version>
  </dependency>
  <!-- ...other Kibo Service Modules -->
<dependencies>
```

9. Open the App.java and import the Kibo dependencies.

```
// App.java     import com.kibocommerce.sdk.common.KiboConfiguration;
import com.kibocommerce.sdk.common.ApiException;
import com.kibocommerce.sdk.customer.api.CustomerAccountApi;
import com.kibocommerce.sdk.customer.models.CustomerAccountCollection;
```

10. Inside the main method, configure the API client and get customer accounts.

```
try {
  // Configure the Kibo SDK with your Kibo Commerce tenant credentials
  KiboConfiguration configuration = KiboConfiguration.builder().build();

  // Initialize the CustomerAccountApi with your Configuration.
  CustomerAccountApi customerAccountApi = new CustomerAccountApi(configuration);

  // Fetch 10 customer accounts to retrieve a list of customers
  CustomerAccountCollection customers = customerAccountApi.getAccounts(0, 10, null, null,
null, null, null, null, null);

  // Display the total number of customers
  System.out.println("Total Customers: " + customers.getTotalCount());
} catch (ApiException e) {
  // Handle the exception
  System.out.println("Error in getting customers: " + e.getMessage());
}
```

11. Run the application.

12. All together, the App.java file should look like this.

```
import com.kibocommerce.sdk.common.ApiCredentials;
import com.kibocommerce.sdk.common.ApiException;
import com.kibocommerce.sdk.common.KiboConfiguration;
import com.kibocommerce.sdk.customer.api.CustomerAccountApi;
import com.kibocommerce.sdk.customer.models.CustomerAccountCollection;


/**
 * Hello world!
 */
public class App {
    public static void main( String[] args ) {
        try {
            // Use your Application Key and Application Secret
            ApiCredentials credentials = ApiCredentials.builder()
                            .setClientId("client_id")
                            .setClientSecret("client_secret")
                            .build();


            // Configure your Tenant ID, Site ID, and Hostname
            KiboConfiguration.builder()
                .withTenantId(12345)
                .withSiteId(12345)
                .withCredentials(credentials)
                .withTenantHost("t39368.sandbox.mozu.com")
                .withHomeHost("t39368.sandbox.mozu.com")
                .build();


            // Configure the Kibo SDK with your Kibo Commerce tenant credentials
            KiboConfiguration configuration = KiboConfiguration.builder().build();


            // Initialize the CustomerAccountApi with your Configuration.
            CustomerAccountApi customerAccountApi = new CustomerAccountApi(configuration);


            // Fetch 10 customer accounts to retrieve a list of customers
            CustomerAccountCollection customers = customerAccountApi.getAccounts(0, 10, null
, null, null, null, null, null, null);


            // Display the total number of customers
            System.out.println("Total Customers: " + customers.getTotalCount());
          } catch (ApiException e) {
            // Handle the exception
            System.out.println("Error in getting customers: " + e.getMessage());
          }
    }
}
```

Modules

There are modules for each micro-service.

- `com.kibocommerce.adminuser`

- `com.kibocommerce.appdevelopment`
- `com.kibocommerce.catalogadministration`
- `com.kibocommerce.catalogstorefront`
- `com.kibocommerce.commerce`
- `com.kibocommerce.common`
- `com.kibocommerce.content`
- `com.kibocommerce.customer`
- `com.kibocommerce.entities`
- `com.kibocommerce.event`
- `com.kibocommerce.fulfillment`
- `com.kibocommerce.importexport`
- `com.kibocommerce.inventory`
- `com.kibocommerce.locationadmin`
- `com.kibocommerce.locationstorefront`
- `com.kibocommerce.orderrouting`
- `com.kibocommerce.pricingstorefront`
- `com.kibocommerce.reference`
- `com.kibocommerce.reservation`
- `com.kibocommerce.settings`
- `com.kibocommerce.shippingadmin`
- `com.kibocommerce.shippingstorefront`
- `com.kibocommerce.solrschemamanager`
- `com.kibocommerce.subscription`

Usage

Create a KiboConfiguration instance and configure it with your credentials and tenant details:

- TenantID
- SiteID (optional)
- ClientID
- ClientSecret
- TenantHost
- HomeHost
- Create an API Client instance using the KiboConfiguration instance
- Make API calls!

Maven

This Maven configuration sets the version for the Kibo Commerce library using a properties block.

```
<properties>
   <kibocommerce.version>2.0.0-SNAPSHOT</kibocommerce.version>
</properties>

<dependencies>
   <dependency>
      <groupId>com.kibocommerce</groupId>
      <artifactId>catalogadministration</artifactId>
      <version>${kibocommerce.version}</version>
   </dependency>
   <!-- ...other Kibo Service Modules -->
</dependencies>
```

Configuration

Create a KiboConfiguration instance. This object can be re-used for the creation of all micro-service API Clients.

Building Configuration

```
import com.kibocommerce.sdk.common.ApiCredentials;
import com.kibocommerce.sdk.common.KiboConfiguration;


//...
public KiboConfiguration getConfiguration() {
   return KiboConfiguration.builder()
            .withTenantId(12345)
            .withSiteId(12345)
            .withCredentials(
                  ApiCredentials.builder().setClientId("client_id")
                        .setClientSecret("client_secret").build())
            .withTenantHost("t39368.sandbox.mozu.com")
            .withHomeHost("t39368.sandbox.mozu.com")
            .build();
}
```

Configuration From System Properties

```
import io.github.cdimascio.dotenv.Dotenv;
import com.kibocommerce.sdk.common.KiboConfiguration;


public KiboConfiguration getConfiguration() {
    // using dotenv to load .env file into system properties
    Dotenv dotenv = Dotenv
        .configure()
        .systemProperties()
        .load();
    // initialize configuration object from system properties
    KiboConfiguration configuration = KiboConfiguration.builder()
                .fromSystemProperties()
                .build();
    return configuration
}
```

Environment Template

```
KIBO_TENANT=
KIBO_SITE=
KIBO_CLIENT_ID=
KIBO_CLIENT_SECRET=
KIBO_CATALOG=
KIBO_MASTER_CATALOG=
KIBO_LOCALE=
KIBO_CURRENCY=
KIBO_HOME_HOST=
KIBO_TENANT_HOST=
KIBO_PCI_HOST=
KIBO_DEBUG_CLIENT=
```

Creating an API Client

Following is the process of creating an API client in Java with Kibo Commerce Integration

```
// Import Kibo Configuration
import com.kibocommerce.sdk.common.ApiCredentials;
import com.kibocommerce.sdk.common.KiboConfiguration;
// Import Kibo Commerce Catalog Administration API
import com.kibocommerce.sdk.catalogadministration.api.ProductsApi;
// Import the Kibo Commerce Catalog Administration Models
import com.kibocommerce.sdk.catalogadministration.models.CatalogAdminsProduct;


public final class App {
    private App() {
    }


    public static void main(String[] args) {
        // Get Kibo Configuration Reference
        KiboConfiguration configuration = getConfiguration();
        // Build API Instance
        ProductsApi api = ProductsApi.builder().withConfig(configuration).build();
        // Make API call to Catalog Administration service to get product
        CatalogAdminsProduct product = api.getProduct(productCode);


    }
}
```