Custom SDKs

If you want an SDK for a language that Kibo does not provide, you can use open source software to develop your own for any language. Kibo recommends using the OpenAPI Generator tool and then improving its API authentication process as shown in our provided packages.

While you can view API endpoints and request/response models in our API documentation, you can also access their OpenAPI specifications through this repository.

OpenAPI Generator

You can access the OpenAPI Generator in two ways:

- Directly from the open source repository
- Using a cli wrapper around the above repository

Refer to the ReadMe files of those repositories for the most up-to-date usage instructions. As an example, the high-level steps for creating a Kibo SDK in Java with the cli wrapper would be:

1. Install the cli tool with `npm install -g @openapitools/openapi-generator-cli`
2. Clone Kibo's OpenAPI specifications.
3. Run cli pointing to the file in the OpenAPI specs folder with `openapi-generator-cli generate -g java -i kibo-open-api-specs/main/docs/openapi_commerce.json -o /var/tmp/java-commerce-client`

SDK Authentication

The OpenAPI Generator doesn't automatically generate the best API authentication. To improve this process, Kibo provides the following public packages for you to reference while implementing similar authentication in your desired language.

- TypeScript/JavaScript authentication package
- Java authentication package

Example

Kibo recommends using the TypeScript/JavaScript package if possible, as it includes cleaner code. The below steps describe how to use that package.

1. Install with `npm install @kibocommerce/sdk-authentication`
2. Ensure you have the following information for configuration.
    - `authHost` : Kibo Commerce Authentication Host Server. It is used to request an access token from the Kibo Commerce OAuth 2.0 service.

- `clientId`: Unique Application (Client) ID of your application. This is viewable from your Dev Center.
- `sharedSecret`: Secret API key used to authenticate your application/client ID.

3. Run the following code for authorization:

```
// import API Auth Client
import { APIAuthClient } from '@kibocommerce/sdk-authentication'

// configuration parameters
const config = {
  clientId: 'client_id'
  sharedSecret: 'secret',
  authHost: 'home.mozu.com'
}

const apiAuthClient = new APIAuthClient(config, fetch)
const kiboAccessToken = await apiAuthClient.getAccessToken()

const response = await fetch('https://some-kibo-api', { headers: { 'Authorization': `Bearer ${k
iboAccessToken}` }})
```

Alternatively, if you are using an version of Node prior to Version 18 or need to use a custom Fetch client:

```
// import API Auth Client
import { APIAuthClient } from '@kibocommerce/sdk-authentication'

// import Fetch API compatible client
import fetch from 'node-fetch';

// configuration parameters
const config = {
  clientId: 'client_id'
  sharedSecret: 'secret',
  authHost: 'home.mozu.com'
}

const apiAuthClient = new APIAuthClient(config, fetch)
const kiboAccessToken = await apiAuthClient.getAccessToken()

const response = await fetch('https://some-kibo-api', { headers: { 'Authorization': `Bearer ${k
iboAccessToken}` }})
```

Token Refresh

Kibo access tokens are valid for one hour and Kibo recommends reusing them as much as possible. This package will automatically handle the refresh of tokens. The client constructor accepts an optional `authTicketCache`.

A simple example of an in-memory cache:

```
interface AuthTicketCache {
    getAuthTicket: (clientId:string) => Promise
    setAuthTicket: (clientId: string, kiboAuthTicket: AppAuthTicket) => void
}
const memo = {}
const memCache: AuthTicketCache = {
  getAuthTicket: async (clientId:string) => {
    return memo[clientId]
   },
  setAuthTicket: (clientId: string, kiboAuthTicket: AppAuthTicket) => {
    memo[clientId] = kiboAuthTicket
  }
}
// import API Auth Client
import { APIAuthClient } from '@kibocommerce/sdk-authentication'

// import Fetch API compatible client
import fetch from 'node-fetch';

// configuration parameters
const config = {
  clientId: 'client_id'
  sharedSecret: 'secret',
  authHost: 'home.mozu.com'
}

const apiAuthClient = new APIAuthClient(config, fetch, memCache)
const kiboAccessToken = await apiAuthClient.getAccessToken()
```