import ApiClient from "../ApiClient";

import Batch from '../model/Batch';

import QueryTransaction from '../model/QueryTransaction';

import TerminalSettings from '../model/TerminalSettings';

export default class ReportingApi {

constructor(apiClient) {

this.apiClient = apiClient || ApiClient.instance;

}

getBatchesByDate(batchDate, account, password, callback) {

const postBody = null;

const pathParams = {

'batchDate': batchDate

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

const accepts = ['text/plain', 'application/json', 'text/json'];

const returnType = [Batch];

return this.apiClient.callApi(

`/api/v1/Reporting/GetBatchesByDate/${batchDate}`,

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

getCheckTerminalSettingsBySecCode(secCode, account, password, callback) {

const postBody = null;

const pathParams = {

'secCode': secCode

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

const accepts = ['text/plain', 'application/json', 'text/json'];

const returnType = [TerminalSettings];

return this.apiClient.callApi(

`/api/v1/Reporting/GetCheckTerminalSettingsBySEC/${secCode}`,

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

getCheckTerminalSettingsById(terminalID, account, password, callback) {

const postBody = null;

const pathParams = {

'terminalID': terminalID

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

const accepts = ['text/plain', 'application/json', 'text/json'];

const returnType = [TerminalSettings];

return this.apiClient.callApi(

`/api/v1/Reporting/GetCheckTerminalSettings/${terminalID}`,

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

getTerminalsSettings(account, password, callback) {

const postBody = null;

const pathParams = {};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

const accepts = ['text/plain', 'application/json', 'text/json'];

const returnType = [TerminalSettings];

return this.apiClient.callApi(

'/api/v1/Reporting/GetCheckTerminals',

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

getTransactionsByBatch(batch, account, password, callback) {

const postBody = null;

const pathParams = {

'batch': batch

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

const accepts = ['text/plain', 'application/json', 'text/json'];

const returnType = [QueryTransaction];

return this.apiClient.callApi(

`/api/v1/Reporting/GetTransactionsByBatch/${batch}`,

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

getTransactionsByDate(transactionDate, account, password, callback) {

const postBody = null;

const pathParams = {

'transactionDate': transactionDate

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

const accepts = ['text/plain', 'application/json', 'text/json'];

const returnType = [QueryTransaction];

return this.apiClient.callApi(

`/api/v1/Reporting/GetTransactionsByDate/${transactionDate}`,

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

}

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using RestSharp;

using IO.Swagger.Client;

using IO.Swagger.Model;

namespace IO.Swagger.Api

{

public interface IReportingApi : IApiAccessor

{

List<Batch> ApiV1ReportingGetBatchesByDateBatchDateGet(DateTime? batchDate, string account, string password);

ApiResponse<List<Batch>> ApiV1ReportingGetBatchesByDateBatchDateGetWithHttpInfo(DateTime? batchDate, string account, string password);

List<TerminalSettings> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet(string secCode, string account, string password);

ApiResponse<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetWithHttpInfo(string secCode, string account, string password);

List<TerminalSettings> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet(int? terminalID, string account, string password);

ApiResponse<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetWithHttpInfo(int? terminalID, string account, string password);

List<TerminalSettings> ApiV1ReportingGetCheckTerminalsGet(string account, string password);

ApiResponse<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalsGetWithHttpInfo(string account, string password);

List<QueryTransaction> ApiV1ReportingGetTransactionsByBatchBatchGet(string batch, string account, string password);

ApiResponse<List<QueryTransaction>> ApiV1ReportingGetTransactionsByBatchBatchGetWithHttpInfo(string batch, string account, string password);

List<QueryTransaction> ApiV1ReportingGetTransactionsByDateTransactionDateGet(DateTime? transactionDate, string account, string password);

ApiResponse<List<QueryTransaction>> ApiV1ReportingGetTransactionsByDateTransactionDateGetWithHttpInfo(DateTime? transactionDate, string account, string password);

System.Threading.Tasks.Task<List<Batch>> ApiV1ReportingGetBatchesByDateBatchDateGetAsync(DateTime? batchDate, string account, string password);

System.Threading.Tasks.Task<ApiResponse<List<Batch>>> ApiV1ReportingGetBatchesByDateBatchDateGetAsyncWithHttpInfo(DateTime? batchDate, string account, string password);

System.Threading.Tasks.Task<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsync(string secCode, string account, string password);

System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsyncWithHttpInfo(string secCode, string account, string password);

System.Threading.Tasks.Task<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsync(int? terminalID, string account, string password);

System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsyncWithHttpInfo(int? terminalID, string account, string password);

System.Threading.Tasks.Task<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalsGetAsync(string account, string password);

System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalsGetAsyncWithHttpInfo(string account, string password);

System.Threading.Tasks.Task<List<QueryTransaction>> ApiV1ReportingGetTransactionsByBatchBatchGetAsync(string batch, string account, string password);

System.Threading.Tasks.Task<ApiResponse<List<QueryTransaction>>> ApiV1ReportingGetTransactionsByBatchBatchGetAsyncWithHttpInfo(string batch, string account, string password);

System.Threading.Tasks.Task<List<QueryTransaction>> ApiV1ReportingGetTransactionsByDateTransactionDateGetAsync(DateTime? transactionDate, string account, string password);

System.Threading.Tasks.Task<ApiResponse<List<QueryTransaction>>> ApiV1ReportingGetTransactionsByDateTransactionDateGetAsyncWithHttpInfo(DateTime? transactionDate, string account, string password);

}

public partial class ReportingApi : IReportingApi

{

private IO.Swagger.Client.ExceptionFactory \_exceptionFactory = (name, response) => null;

public ReportingApi(String basePath)

{

this.Configuration = new IO.Swagger.Client.Configuration { BasePath = basePath };

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public ReportingApi()

{

this.Configuration = IO.Swagger.Client.Configuration.Default;

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public ReportingApi(IO.Swagger.Client.Configuration configuration = null)

{

if (configuration == null) // use the default one in Configuration

this.Configuration = IO.Swagger.Client.Configuration.Default;

else

this.Configuration = configuration;

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public String GetBasePath()

{

return this.Configuration.ApiClient.RestClient.BaseUrl.ToString();

}

[Obsolete("SetBasePath is deprecated, please do 'Configuration.ApiClient = new ApiClient(\"http://new-path\")' instead.")]

public void SetBasePath(String basePath)

{

// do nothing

}

public IO.Swagger.Client.Configuration Configuration { get; set; }

public IO.Swagger.Client.ExceptionFactory ExceptionFactory

{

get

{

if (\_exceptionFactory != null && \_exceptionFactory.GetInvocationList().Length > 1)

{

throw new InvalidOperationException("Multicast delegate for ExceptionFactory is unsupported.");

}

return \_exceptionFactory;

}

set { \_exceptionFactory = value; }

}

[Obsolete("DefaultHeader is deprecated, please use Configuration.DefaultHeader instead.")]

public IDictionary<String, String> DefaultHeader()

{

return new ReadOnlyDictionary<string, string>(this.Configuration.DefaultHeader);

}

[Obsolete("AddDefaultHeader is deprecated, please use Configuration.AddDefaultHeader instead.")]

public void AddDefaultHeader(string key, string value)

{

this.Configuration.AddDefaultHeader(key, value);

}

public List<Batch> ApiV1ReportingGetBatchesByDateBatchDateGet(DateTime? batchDate, string account, string password)

{

ApiResponse<List<Batch>> localVarResponse = ApiV1ReportingGetBatchesByDateBatchDateGetWithHttpInfo(batchDate, account, password);

return localVarResponse.Data;

}

public ApiResponse<List<Batch>> ApiV1ReportingGetBatchesByDateBatchDateGetWithHttpInfo(DateTime? batchDate, string account, string password)

{

// verify the required parameter 'batchDate' is set

if (batchDate == null)

throw new ApiException(400, "Missing required parameter 'batchDate' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

var localVarPath = "/api/v1/Reporting/GetBatchesByDate/{batchDate}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (batchDate != null) localVarPathParams.Add("batchDate", this.Configuration.ApiClient.ParameterToString(batchDate)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetBatchesByDateBatchDateGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<Batch>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<Batch>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<Batch>)));

}

public async System.Threading.Tasks.Task<List<Batch>> ApiV1ReportingGetBatchesByDateBatchDateGetAsync(DateTime? batchDate, string account, string password)

{

ApiResponse<List<Batch>> localVarResponse = await ApiV1ReportingGetBatchesByDateBatchDateGetAsyncWithHttpInfo(batchDate, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<Batch>>> ApiV1ReportingGetBatchesByDateBatchDateGetAsyncWithHttpInfo(DateTime? batchDate, string account, string password)

{

// verify the required parameter 'batchDate' is set

if (batchDate == null)

throw new ApiException(400, "Missing required parameter 'batchDate' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

var localVarPath = "/api/v1/Reporting/GetBatchesByDate/{batchDate}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (batchDate != null) localVarPathParams.Add("batchDate", this.Configuration.ApiClient.ParameterToString(batchDate)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetBatchesByDateBatchDateGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<Batch>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<Batch>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<Batch>)));

}

public List<TerminalSettings> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet(string secCode, string account, string password)

{

ApiResponse<List<TerminalSettings>> localVarResponse = ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetWithHttpInfo(secCode, account, password);

return localVarResponse.Data;

}

public ApiResponse<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetWithHttpInfo(string secCode, string account, string password)

{

// verify the required parameter 'secCode' is set

if (secCode == null)

throw new ApiException(400, "Missing required parameter 'secCode' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (secCode != null) localVarPathParams.Add("secCode", this.Configuration.ApiClient.ParameterToString(secCode)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

public async System.Threading.Tasks.Task<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsync(string secCode, string account, string password)

{

ApiResponse<List<TerminalSettings>> localVarResponse = await ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsyncWithHttpInfo(secCode, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsyncWithHttpInfo(string secCode, string account, string password)

{

// verify the required parameter 'secCode' is set

if (secCode == null)

throw new ApiException(400, "Missing required parameter 'secCode' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (secCode != null) localVarPathParams.Add("secCode", this.Configuration.ApiClient.ParameterToString(secCode)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

public List<TerminalSettings> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet(int? terminalID, string account, string password)

{

ApiResponse<List<TerminalSettings>> localVarResponse = ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetWithHttpInfo(terminalID, account, password);

return localVarResponse.Data;

}

public ApiResponse<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetWithHttpInfo(int? terminalID, string account, string password)

{

// verify the required parameter 'terminalID' is set

if (terminalID == null)

throw new ApiException(400, "Missing required parameter 'terminalID' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminalSettings/{terminalID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (terminalID != null) localVarPathParams.Add("terminalID", this.Configuration.ApiClient.ParameterToString(terminalID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

public async System.Threading.Tasks.Task<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsync(int? terminalID, string account, string password)

{

ApiResponse<List<TerminalSettings>> localVarResponse = await ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsyncWithHttpInfo(terminalID, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsyncWithHttpInfo(int? terminalID, string account, string password)

{

// verify the required parameter 'terminalID' is set

if (terminalID == null)

throw new ApiException(400, "Missing required parameter 'terminalID' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminalSettings/{terminalID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (terminalID != null) localVarPathParams.Add("terminalID", this.Configuration.ApiClient.ParameterToString(terminalID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

public List<TerminalSettings> ApiV1ReportingGetCheckTerminalsGet(string account, string password)

{

ApiResponse<List<TerminalSettings>> localVarResponse = ApiV1ReportingGetCheckTerminalsGetWithHttpInfo(account, password);

return localVarResponse.Data;

}

public ApiResponse<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalsGetWithHttpInfo(string account, string password)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalsGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalsGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminals";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalsGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

public async System.Threading.Tasks.Task<List<TerminalSettings>> ApiV1ReportingGetCheckTerminalsGetAsync(string account, string password)

{

ApiResponse<List<TerminalSettings>> localVarResponse = await ApiV1ReportingGetCheckTerminalsGetAsyncWithHttpInfo(account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalsGetAsyncWithHttpInfo(string account, string password)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalsGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalsGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminals";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalsGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

public List<QueryTransaction> ApiV1ReportingGetTransactionsByBatchBatchGet(string batch, string account, string password)

{

ApiResponse<List<QueryTransaction>> localVarResponse = ApiV1ReportingGetTransactionsByBatchBatchGetWithHttpInfo(batch, account, password);

return localVarResponse.Data;

}

public ApiResponse<List<QueryTransaction>> ApiV1ReportingGetTransactionsByBatchBatchGetWithHttpInfo(string batch, string account, string password)

{

// verify the required parameter 'batch' is set

if (batch == null)

throw new ApiException(400, "Missing required parameter 'batch' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

var localVarPath = "/api/v1/Reporting/GetTransactionsByBatch/{batch}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (batch != null) localVarPathParams.Add("batch", this.Configuration.ApiClient.ParameterToString(batch)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetTransactionsByBatchBatchGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<QueryTransaction>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<QueryTransaction>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<QueryTransaction>)));

}

public async System.Threading.Tasks.Task<List<QueryTransaction>> ApiV1ReportingGetTransactionsByBatchBatchGetAsync(string batch, string account, string password)

{

ApiResponse<List<QueryTransaction>> localVarResponse = await ApiV1ReportingGetTransactionsByBatchBatchGetAsyncWithHttpInfo(batch, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<QueryTransaction>>> ApiV1ReportingGetTransactionsByBatchBatchGetAsyncWithHttpInfo(string batch, string account, string password)

{

// verify the required parameter 'batch' is set

if (batch == null)

throw new ApiException(400, "Missing required parameter 'batch' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

var localVarPath = "/api/v1/Reporting/GetTransactionsByBatch/{batch}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (batch != null) localVarPathParams.Add("batch", this.Configuration.ApiClient.ParameterToString(batch)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetTransactionsByBatchBatchGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<QueryTransaction>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<QueryTransaction>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<QueryTransaction>)));

}

public List<QueryTransaction> ApiV1ReportingGetTransactionsByDateTransactionDateGet(DateTime? transactionDate, string account, string password)

{

ApiResponse<List<QueryTransaction>> localVarResponse = ApiV1ReportingGetTransactionsByDateTransactionDateGetWithHttpInfo(transactionDate, account, password);

return localVarResponse.Data;

}

public ApiResponse<List<QueryTransaction>> ApiV1ReportingGetTransactionsByDateTransactionDateGetWithHttpInfo(DateTime? transactionDate, string account, string password)

{

// verify the required parameter 'transactionDate' is set

if (transactionDate == null)

throw new ApiException(400, "Missing required parameter 'transactionDate' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

var localVarPath = "/api/v1/Reporting/GetTransactionsByDate/{transactionDate}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (transactionDate != null) localVarPathParams.Add("transactionDate", this.Configuration.ApiClient.ParameterToString(transactionDate)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetTransactionsByDateTransactionDateGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<QueryTransaction>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<QueryTransaction>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<QueryTransaction>)));

}

public async System.Threading.Tasks.Task<List<QueryTransaction>> ApiV1ReportingGetTransactionsByDateTransactionDateGetAsync(DateTime? transactionDate, string account, string password)

{

ApiResponse<List<QueryTransaction>> localVarResponse = await ApiV1ReportingGetTransactionsByDateTransactionDateGetAsyncWithHttpInfo(transactionDate, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<QueryTransaction>>> ApiV1ReportingGetTransactionsByDateTransactionDateGetAsyncWithHttpInfo(DateTime? transactionDate, string account, string password)

{

// verify the required parameter 'transactionDate' is set

if (transactionDate == null)

throw new ApiException(400, "Missing required parameter 'transactionDate' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

var localVarPath = "/api/v1/Reporting/GetTransactionsByDate/{transactionDate}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (transactionDate != null) localVarPathParams.Add("transactionDate", this.Configuration.ApiClient.ParameterToString(transactionDate)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetTransactionsByDateTransactionDateGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<QueryTransaction>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<QueryTransaction>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<QueryTransaction>)));

}

}

}

<?php

namespace Swagger\Client\Api;

use GuzzleHttp\Client;

use GuzzleHttp\ClientInterface;

use GuzzleHttp\Exception\RequestException;

use GuzzleHttp\Psr7\MultipartStream;

use GuzzleHttp\Psr7\Request;

use GuzzleHttp\RequestOptions;

use Swagger\Client\ApiException;

use Swagger\Client\Configuration;

use Swagger\Client\HeaderSelector;

use Swagger\Client\ObjectSerializer;

class ReportingApi

{

protected $client;

protected $config;

protected $headerSelector;

public function \_\_construct(

ClientInterface $client = null,

Configuration $config = null,

HeaderSelector $selector = null

) {

$this->client = $client ?: new Client();

$this->config = $config ?: new Configuration();

$this->headerSelector = $selector ?: new HeaderSelector();

}

public function getConfig()

{

return $this->config;

}

public function apiV1ReportingGetBatchesByDateBatchDateGet($batch\_date, $account, $password)

{

list($response) = $this->apiV1ReportingGetBatchesByDateBatchDateGetWithHttpInfo($batch\_date, $account, $password);

return $response;

}

public function apiV1ReportingGetBatchesByDateBatchDateGetWithHttpInfo($batch\_date, $account, $password)

{

$returnType = '\Swagger\Client\Model\Batch[]';

$request = $this->apiV1ReportingGetBatchesByDateBatchDateGetRequest($batch\_date, $account, $password);

try {

$options = $this->createHttpClientOption();

try {

$response = $this->client->send($request, $options);

} catch (RequestException $e) {

throw new ApiException(

"[{$e->getCode()}] {$e->getMessage()}",

$e->getCode(),

$e->getResponse() ? $e->getResponse()->getHeaders() : null,

$e->getResponse() ? $e->getResponse()->getBody()->getContents() : null

);

}

$statusCode = $response->getStatusCode();

if ($statusCode < 200 || $statusCode > 299) {

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$request->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if (!in\_array($returnType, ['string','integer','bool'])) {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

} catch (ApiException $e) {

switch ($e->getCode()) {

case 200:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Batch[]',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

case 400:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Response',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

}

throw $e;

}

}

public function apiV1ReportingGetBatchesByDateBatchDateGetAsync($batch\_date, $account, $password)

{

return $this->apiV1ReportingGetBatchesByDateBatchDateGetAsyncWithHttpInfo($batch\_date, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1ReportingGetBatchesByDateBatchDateGetAsyncWithHttpInfo($batch\_date, $account, $password)

{

$returnType = '\Swagger\Client\Model\Batch[]';

$request = $this->apiV1ReportingGetBatchesByDateBatchDateGetRequest($batch\_date, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

protected function apiV1ReportingGetBatchesByDateBatchDateGetRequest($batch\_date, $account, $password)

{

// verify the required parameter 'batch\_date' is set

if ($batch\_date === null || (is\_array($batch\_date) && count($batch\_date) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $batch\_date when calling apiV1ReportingGetBatchesByDateBatchDateGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetBatchesByDateBatchDateGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetBatchesByDateBatchDateGet'

);

}

$resourcePath = '/api/v1/Reporting/GetBatchesByDate/{batchDate}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($batch\_date !== null) {

$resourcePath = str\_replace(

'{' . 'batchDate' . '}',

ObjectSerializer::toPathValue($batch\_date),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

public function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet($sec\_code, $account, $password)

{

list($response) = $this->apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetWithHttpInfo($sec\_code, $account, $password);

return $response;

}

public function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetWithHttpInfo($sec\_code, $account, $password)

{

$returnType = '\Swagger\Client\Model\TerminalSettings[]';

$request = $this->apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetRequest($sec\_code, $account, $password);

try {

$options = $this->createHttpClientOption();

try {

$response = $this->client->send($request, $options);

} catch (RequestException $e) {

throw new ApiException(

"[{$e->getCode()}] {$e->getMessage()}",

$e->getCode(),

$e->getResponse() ? $e->getResponse()->getHeaders() : null,

$e->getResponse() ? $e->getResponse()->getBody()->getContents() : null

);

}

$statusCode = $response->getStatusCode();

if ($statusCode < 200 || $statusCode > 299) {

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$request->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if (!in\_array($returnType, ['string','integer','bool'])) {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

} catch (ApiException $e) {

switch ($e->getCode()) {

case 200:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\TerminalSettings[]',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

case 400:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Response',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

}

throw $e;

}

}

public function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsync($sec\_code, $account, $password)

{

return $this->apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsyncWithHttpInfo($sec\_code, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsyncWithHttpInfo($sec\_code, $account, $password)

{

$returnType = '\Swagger\Client\Model\TerminalSettings[]';

$request = $this->apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetRequest($sec\_code, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

protected function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetRequest($sec\_code, $account, $password)

{

// verify the required parameter 'sec\_code' is set

if ($sec\_code === null || (is\_array($sec\_code) && count($sec\_code) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $sec\_code when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

$resourcePath = '/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($sec\_code !== null) {

$resourcePath = str\_replace(

'{' . 'secCode' . '}',

ObjectSerializer::toPathValue($sec\_code),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

public function apiV1ReportingGetCheckTerminalSettingsTerminalIDGet($terminal\_id, $account, $password)

{

list($response) = $this->apiV1ReportingGetCheckTerminalSettingsTerminalIDGetWithHttpInfo($terminal\_id, $account, $password);

return $response;

}

public function apiV1ReportingGetCheckTerminalSettingsTerminalIDGetWithHttpInfo($terminal\_id, $account, $password)

{

$returnType = '\Swagger\Client\Model\TerminalSettings[]';

$request = $this->apiV1ReportingGetCheckTerminalSettingsTerminalIDGetRequest($terminal\_id, $account, $password);

try {

$options = $this->createHttpClientOption();

try {

$response = $this->client->send($request, $options);

} catch (RequestException $e) {

throw new ApiException(

"[{$e->getCode()}] {$e->getMessage()}",

$e->getCode(),

$e->getResponse() ? $e->getResponse()->getHeaders() : null,

$e->getResponse() ? $e->getResponse()->getBody()->getContents() : null

);

}

$statusCode = $response->getStatusCode();

if ($statusCode < 200 || $statusCode > 299) {

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$request->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if (!in\_array($returnType, ['string','integer','bool'])) {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

} catch (ApiException $e) {

switch ($e->getCode()) {

case 200:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\TerminalSettings[]',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

case 400:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Response',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

}

throw $e;

}

}

public function apiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsync($terminal\_id, $account, $password)

{

return $this->apiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsyncWithHttpInfo($terminal\_id, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsyncWithHttpInfo($terminal\_id, $account, $password)

{

$returnType = '\Swagger\Client\Model\TerminalSettings[]';

$request = $this->apiV1ReportingGetCheckTerminalSettingsTerminalIDGetRequest($terminal\_id, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

protected function apiV1ReportingGetCheckTerminalSettingsTerminalIDGetRequest($terminal\_id, $account, $password)

{

// verify the required parameter 'terminal\_id' is set

if ($terminal\_id === null || (is\_array($terminal\_id) && count($terminal\_id) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $terminal\_id when calling apiV1ReportingGetCheckTerminalSettingsTerminalIDGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetCheckTerminalSettingsTerminalIDGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetCheckTerminalSettingsTerminalIDGet'

);

}

$resourcePath = '/api/v1/Reporting/GetCheckTerminalSettings/{terminalID}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($terminal\_id !== null) {

$resourcePath = str\_replace(

'{' . 'terminalID' . '}',

ObjectSerializer::toPathValue($terminal\_id),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

public function apiV1ReportingGetCheckTerminalsGet($account, $password)

{

list($response) = $this->apiV1ReportingGetCheckTerminalsGetWithHttpInfo($account, $password);

return $response;

}

public function apiV1ReportingGetCheckTerminalsGetWithHttpInfo($account, $password)

{

$returnType = '\Swagger\Client\Model\TerminalSettings[]';

$request = $this->apiV1ReportingGetCheckTerminalsGetRequest($account, $password);

try {

$options = $this->createHttpClientOption();

try {

$response = $this->client->send($request, $options);

} catch (RequestException $e) {

throw new ApiException(

"[{$e->getCode()}] {$e->getMessage()}",

$e->getCode(),

$e->getResponse() ? $e->getResponse()->getHeaders() : null,

$e->getResponse() ? $e->getResponse()->getBody()->getContents() : null

);

}

$statusCode = $response->getStatusCode();

if ($statusCode < 200 || $statusCode > 299) {

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$request->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if (!in\_array($returnType, ['string','integer','bool'])) {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

} catch (ApiException $e) {

switch ($e->getCode()) {

case 200:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\TerminalSettings[]',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

case 400:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Response',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

}

throw $e;

}

}

public function apiV1ReportingGetCheckTerminalsGetAsync($account, $password)

{

return $this->apiV1ReportingGetCheckTerminalsGetAsyncWithHttpInfo($account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1ReportingGetCheckTerminalsGetAsyncWithHttpInfo($account, $password)

{

$returnType = '\Swagger\Client\Model\TerminalSettings[]';

$request = $this->apiV1ReportingGetCheckTerminalsGetRequest($account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

protected function apiV1ReportingGetCheckTerminalsGetRequest($account, $password)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetCheckTerminalsGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetCheckTerminalsGet'

);

}

$resourcePath = '/api/v1/Reporting/GetCheckTerminals';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

public function apiV1ReportingGetTransactionsByBatchBatchGet($batch, $account, $password)

{

list($response) = $this->apiV1ReportingGetTransactionsByBatchBatchGetWithHttpInfo($batch, $account, $password);

return $response;

}

public function apiV1ReportingGetTransactionsByBatchBatchGetWithHttpInfo($batch, $account, $password)

{

$returnType = '\Swagger\Client\Model\QueryTransaction[]';

$request = $this->apiV1ReportingGetTransactionsByBatchBatchGetRequest($batch, $account, $password);

try {

$options = $this->createHttpClientOption();

try {

$response = $this->client->send($request, $options);

} catch (RequestException $e) {

throw new ApiException(

"[{$e->getCode()}] {$e->getMessage()}",

$e->getCode(),

$e->getResponse() ? $e->getResponse()->getHeaders() : null,

$e->getResponse() ? $e->getResponse()->getBody()->getContents() : null

);

}

$statusCode = $response->getStatusCode();

if ($statusCode < 200 || $statusCode > 299) {

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$request->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if (!in\_array($returnType, ['string','integer','bool'])) {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

} catch (ApiException $e) {

switch ($e->getCode()) {

case 200:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\QueryTransaction[]',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

case 400:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Response',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

}

throw $e;

}

}

public function apiV1ReportingGetTransactionsByBatchBatchGetAsync($batch, $account, $password)

{

return $this->apiV1ReportingGetTransactionsByBatchBatchGetAsyncWithHttpInfo($batch, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1ReportingGetTransactionsByBatchBatchGetAsyncWithHttpInfo($batch, $account, $password)

{

$returnType = '\Swagger\Client\Model\QueryTransaction[]';

$request = $this->apiV1ReportingGetTransactionsByBatchBatchGetRequest($batch, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

protected function apiV1ReportingGetTransactionsByBatchBatchGetRequest($batch, $account, $password)

{

// verify the required parameter 'batch' is set

if ($batch === null || (is\_array($batch) && count($batch) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $batch when calling apiV1ReportingGetTransactionsByBatchBatchGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetTransactionsByBatchBatchGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetTransactionsByBatchBatchGet'

);

}

$resourcePath = '/api/v1/Reporting/GetTransactionsByBatch/{batch}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($batch !== null) {

$resourcePath = str\_replace(

'{' . 'batch' . '}',

ObjectSerializer::toPathValue($batch),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

public function apiV1ReportingGetTransactionsByDateTransactionDateGet($transaction\_date, $account, $password)

{

list($response) = $this->apiV1ReportingGetTransactionsByDateTransactionDateGetWithHttpInfo($transaction\_date, $account, $password);

return $response;

}

public function apiV1ReportingGetTransactionsByDateTransactionDateGetWithHttpInfo($transaction\_date, $account, $password)

{

$returnType = '\Swagger\Client\Model\QueryTransaction[]';

$request = $this->apiV1ReportingGetTransactionsByDateTransactionDateGetRequest($transaction\_date, $account, $password);

try {

$options = $this->createHttpClientOption();

try {

$response = $this->client->send($request, $options);

} catch (RequestException $e) {

throw new ApiException(

"[{$e->getCode()}] {$e->getMessage()}",

$e->getCode(),

$e->getResponse() ? $e->getResponse()->getHeaders() : null,

$e->getResponse() ? $e->getResponse()->getBody()->getContents() : null

);

}

$statusCode = $response->getStatusCode();

if ($statusCode < 200 || $statusCode > 299) {

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$request->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if (!in\_array($returnType, ['string','integer','bool'])) {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

} catch (ApiException $e) {

switch ($e->getCode()) {

case 200:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\QueryTransaction[]',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

case 400:

$data = ObjectSerializer::deserialize(

$e->getResponseBody(),

'\Swagger\Client\Model\Response',

$e->getResponseHeaders()

);

$e->setResponseObject($data);

break;

}

throw $e;

}

}

public function apiV1ReportingGetTransactionsByDateTransactionDateGetAsync($transaction\_date, $account, $password)

{

return $this->apiV1ReportingGetTransactionsByDateTransactionDateGetAsyncWithHttpInfo($transaction\_date, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1ReportingGetTransactionsByDateTransactionDateGetAsyncWithHttpInfo($transaction\_date, $account, $password)

{

$returnType = '\Swagger\Client\Model\QueryTransaction[]';

$request = $this->apiV1ReportingGetTransactionsByDateTransactionDateGetRequest($transaction\_date, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

protected function apiV1ReportingGetTransactionsByDateTransactionDateGetRequest($transaction\_date, $account, $password)

{

// verify the required parameter 'transaction\_date' is set

if ($transaction\_date === null || (is\_array($transaction\_date) && count($transaction\_date) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $transaction\_date when calling apiV1ReportingGetTransactionsByDateTransactionDateGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetTransactionsByDateTransactionDateGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetTransactionsByDateTransactionDateGet'

);

}

$resourcePath = '/api/v1/Reporting/GetTransactionsByDate/{transactionDate}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($transaction\_date !== null) {

$resourcePath = str\_replace(

'{' . 'transactionDate' . '}',

ObjectSerializer::toPathValue($transaction\_date),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

protected function createHttpClientOption()

{

$options = [];

if ($this->config->getDebug()) {

$options[RequestOptions::DEBUG] = fopen($this->config->getDebugFile(), 'a');

if (!$options[RequestOptions::DEBUG]) {

throw new \RuntimeException('Failed to open the debug file: ' . $this->config->getDebugFile());

}

}

return $options;

}

}

# coding: utf-8

from \_\_future\_\_ import absolute\_import

import re # noqa: F401

# python 2 and python 3 compatibility library

import six

from swagger\_client.api\_client import ApiClient

class ReportingApi(object):

def \_\_init\_\_(self, api\_client=None):

if api\_client is None:

api\_client = ApiClient()

self.api\_client = api\_client

def api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get(self, batch\_date, account, password, \*\*kwargs): # noqa: E501

kwargs['\_return\_http\_data\_only'] = True

if kwargs.get('async\_req'):

return self.api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get\_with\_http\_info(batch\_date, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get\_with\_http\_info(batch\_date, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get\_with\_http\_info(self, batch\_date, account, password, \*\*kwargs): # noqa: E501

all\_params = ['batch\_date', 'account', 'password'] # noqa: E501

all\_params.append('async\_req')

all\_params.append('\_return\_http\_data\_only')

all\_params.append('\_preload\_content')

all\_params.append('\_request\_timeout')

params = locals()

for key, val in six.iteritems(params['kwargs']):

if key not in all\_params:

raise TypeError(

"Got an unexpected keyword argument '%s'"

" to method api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'batch\_date' is set

if ('batch\_date' not in params or

params['batch\_date'] is None):

raise ValueError("Missing the required parameter `batch\_date` when calling `api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get`") # noqa: E501

# verify the required parameter 'account' is set

if ('account' not in params or

params['account'] is None):

raise ValueError("Missing the required parameter `account` when calling `api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get`") # noqa: E501

# verify the required parameter 'password' is set

if ('password' not in params or

params['password'] is None):

raise ValueError("Missing the required parameter `password` when calling `api\_v1\_reporting\_get\_batches\_by\_date\_batch\_date\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'batch\_date' in params:

path\_params['batchDate'] = params['batch\_date'] # noqa: E501

query\_params = []

header\_params = {}

if 'account' in params:

header\_params['Account'] = params['account'] # noqa: E501

if 'password' in params:

header\_params['Password'] = params['password'] # noqa: E501

form\_params = []

local\_var\_files = {}

body\_params = None

# HTTP header `Accept`

header\_params['Accept'] = self.api\_client.select\_header\_accept(

['text/plain', 'application/json', 'text/json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Reporting/GetBatchesByDate/{batchDate}', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[Batch]', # noqa: E501

auth\_settings=auth\_settings,

async\_req=params.get('async\_req'),

\_return\_http\_data\_only=params.get('\_return\_http\_data\_only'),

\_preload\_content=params.get('\_preload\_content', True),

\_request\_timeout=params.get('\_request\_timeout'),

collection\_formats=collection\_formats)

def api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get(self, sec\_code, account, password, \*\*kwargs): # noqa: E501

kwargs['\_return\_http\_data\_only'] = True

if kwargs.get('async\_req'):

return self.api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get\_with\_http\_info(sec\_code, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get\_with\_http\_info(sec\_code, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get\_with\_http\_info(self, sec\_code, account, password, \*\*kwargs): # noqa: E501

all\_params = ['sec\_code', 'account', 'password'] # noqa: E501

all\_params.append('async\_req')

all\_params.append('\_return\_http\_data\_only')

all\_params.append('\_preload\_content')

all\_params.append('\_request\_timeout')

params = locals()

for key, val in six.iteritems(params['kwargs']):

if key not in all\_params:

raise TypeError(

"Got an unexpected keyword argument '%s'"

" to method api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'sec\_code' is set

if ('sec\_code' not in params or

params['sec\_code'] is None):

raise ValueError("Missing the required parameter `sec\_code` when calling `api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get`") # noqa: E501

# verify the required parameter 'account' is set

if ('account' not in params or

params['account'] is None):

raise ValueError("Missing the required parameter `account` when calling `api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get`") # noqa: E501

# verify the required parameter 'password' is set

if ('password' not in params or

params['password'] is None):

raise ValueError("Missing the required parameter `password` when calling `api\_v1\_reporting\_get\_check\_terminal\_settings\_by\_sec\_sec\_code\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'sec\_code' in params:

path\_params['secCode'] = params['sec\_code'] # noqa: E501

query\_params = []

header\_params = {}

if 'account' in params:

header\_params['Account'] = params['account'] # noqa: E501

if 'password' in params:

header\_params['Password'] = params['password'] # noqa: E501

form\_params = []

local\_var\_files = {}

body\_params = None

# HTTP header `Accept`

header\_params['Accept'] = self.api\_client.select\_header\_accept(

['text/plain', 'application/json', 'text/json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[TerminalSettings]', # noqa: E501

auth\_settings=auth\_settings,

async\_req=params.get('async\_req'),

\_return\_http\_data\_only=params.get('\_return\_http\_data\_only'),

\_preload\_content=params.get('\_preload\_content', True),

\_request\_timeout=params.get('\_request\_timeout'),

collection\_formats=collection\_formats)

def api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get(self, terminal\_id, account, password, \*\*kwargs): # noqa: E501

kwargs['\_return\_http\_data\_only'] = True

if kwargs.get('async\_req'):

return self.api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get\_with\_http\_info(terminal\_id, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get\_with\_http\_info(terminal\_id, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get\_with\_http\_info(self, terminal\_id, account, password, \*\*kwargs): # noqa: E501

all\_params = ['terminal\_id', 'account', 'password'] # noqa: E501

all\_params.append('async\_req')

all\_params.append('\_return\_http\_data\_only')

all\_params.append('\_preload\_content')

all\_params.append('\_request\_timeout')

params = locals()

for key, val in six.iteritems(params['kwargs']):

if key not in all\_params:

raise TypeError(

"Got an unexpected keyword argument '%s'"

" to method api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'terminal\_id' is set

if ('terminal\_id' not in params or

params['terminal\_id'] is None):

raise ValueError("Missing the required parameter `terminal\_id` when calling `api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get`") # noqa: E501

# verify the required parameter 'account' is set

if ('account' not in params or

params['account'] is None):

raise ValueError("Missing the required parameter `account` when calling `api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get`") # noqa: E501

# verify the required parameter 'password' is set

if ('password' not in params or

params['password'] is None):

raise ValueError("Missing the required parameter `password` when calling `api\_v1\_reporting\_get\_check\_terminal\_settings\_terminal\_id\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'terminal\_id' in params:

path\_params['terminalID'] = params['terminal\_id'] # noqa: E501

query\_params = []

header\_params = {}

if 'account' in params:

header\_params['Account'] = params['account'] # noqa: E501

if 'password' in params:

header\_params['Password'] = params['password'] # noqa: E501

form\_params = []

local\_var\_files = {}

body\_params = None

# HTTP header `Accept`

header\_params['Accept'] = self.api\_client.select\_header\_accept(

['text/plain', 'application/json', 'text/json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Reporting/GetCheckTerminalSettings/{terminalID}', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[TerminalSettings]', # noqa: E501

auth\_settings=auth\_settings,

async\_req=params.get('async\_req'),

\_return\_http\_data\_only=params.get('\_return\_http\_data\_only'),

\_preload\_content=params.get('\_preload\_content', True),

\_request\_timeout=params.get('\_request\_timeout'),

collection\_formats=collection\_formats)

def api\_v1\_reporting\_get\_check\_terminals\_get(self, account, password, \*\*kwargs): # noqa: E501

kwargs['\_return\_http\_data\_only'] = True

if kwargs.get('async\_req'):

return self.api\_v1\_reporting\_get\_check\_terminals\_get\_with\_http\_info(account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_reporting\_get\_check\_terminals\_get\_with\_http\_info(account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_reporting\_get\_check\_terminals\_get\_with\_http\_info(self, account, password, \*\*kwargs): # noqa: E501

all\_params = ['account', 'password'] # noqa: E501

all\_params.append('async\_req')

all\_params.append('\_return\_http\_data\_only')

all\_params.append('\_preload\_content')

all\_params.append('\_request\_timeout')

params = locals()

for key, val in six.iteritems(params['kwargs']):

if key not in all\_params:

raise TypeError(

"Got an unexpected keyword argument '%s'"

" to method api\_v1\_reporting\_get\_check\_terminals\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'account' is set

if ('account' not in params or

params['account'] is None):

raise ValueError("Missing the required parameter `account` when calling `api\_v1\_reporting\_get\_check\_terminals\_get`") # noqa: E501

# verify the required parameter 'password' is set

if ('password' not in params or

params['password'] is None):

raise ValueError("Missing the required parameter `password` when calling `api\_v1\_reporting\_get\_check\_terminals\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

query\_params = []

header\_params = {}

if 'account' in params:

header\_params['Account'] = params['account'] # noqa: E501

if 'password' in params:

header\_params['Password'] = params['password'] # noqa: E501

form\_params = []

local\_var\_files = {}

body\_params = None

# HTTP header `Accept`

header\_params['Accept'] = self.api\_client.select\_header\_accept(

['text/plain', 'application/json', 'text/json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Reporting/GetCheckTerminals', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[TerminalSettings]', # noqa: E501

auth\_settings=auth\_settings,

async\_req=params.get('async\_req'),

\_return\_http\_data\_only=params.get('\_return\_http\_data\_only'),

\_preload\_content=params.get('\_preload\_content', True),

\_request\_timeout=params.get('\_request\_timeout'),

collection\_formats=collection\_formats)

def api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get(self, batch, account, password, \*\*kwargs): # noqa: E501

kwargs['\_return\_http\_data\_only'] = True

if kwargs.get('async\_req'):

return self.api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get\_with\_http\_info(batch, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get\_with\_http\_info(batch, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get\_with\_http\_info(self, batch, account, password, \*\*kwargs): # noqa: E501

all\_params = ['batch', 'account', 'password'] # noqa: E501

all\_params.append('async\_req')

all\_params.append('\_return\_http\_data\_only')

all\_params.append('\_preload\_content')

all\_params.append('\_request\_timeout')

params = locals()

for key, val in six.iteritems(params['kwargs']):

if key not in all\_params:

raise TypeError(

"Got an unexpected keyword argument '%s'"

" to method api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'batch' is set

if ('batch' not in params or

params['batch'] is None):

raise ValueError("Missing the required parameter `batch` when calling `api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get`") # noqa: E501

# verify the required parameter 'account' is set

if ('account' not in params or

params['account'] is None):

raise ValueError("Missing the required parameter `account` when calling `api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get`") # noqa: E501

# verify the required parameter 'password' is set

if ('password' not in params or

params['password'] is None):

raise ValueError("Missing the required parameter `password` when calling `api\_v1\_reporting\_get\_transactions\_by\_batch\_batch\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'batch' in params:

path\_params['batch'] = params['batch'] # noqa: E501

query\_params = []

header\_params = {}

if 'account' in params:

header\_params['Account'] = params['account'] # noqa: E501

if 'password' in params:

header\_params['Password'] = params['password'] # noqa: E501

form\_params = []

local\_var\_files = {}

body\_params = None

# HTTP header `Accept`

header\_params['Accept'] = self.api\_client.select\_header\_accept(

['text/plain', 'application/json', 'text/json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Reporting/GetTransactionsByBatch/{batch}', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[QueryTransaction]', # noqa: E501

auth\_settings=auth\_settings,

async\_req=params.get('async\_req'),

\_return\_http\_data\_only=params.get('\_return\_http\_data\_only'),

\_preload\_content=params.get('\_preload\_content', True),

\_request\_timeout=params.get('\_request\_timeout'),

collection\_formats=collection\_formats)

def api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get(self, transaction\_date, account, password, \*\*kwargs): # noqa: E501

kwargs['\_return\_http\_data\_only'] = True

if kwargs.get('async\_req'):

return self.api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get\_with\_http\_info(transaction\_date, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get\_with\_http\_info(transaction\_date, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get\_with\_http\_info(self, transaction\_date, account, password, \*\*kwargs): # noqa: E501

all\_params = ['transaction\_date', 'account', 'password'] # noqa: E501

all\_params.append('async\_req')

all\_params.append('\_return\_http\_data\_only')

all\_params.append('\_preload\_content')

all\_params.append('\_request\_timeout')

params = locals()

for key, val in six.iteritems(params['kwargs']):

if key not in all\_params:

raise TypeError(

"Got an unexpected keyword argument '%s'"

" to method api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'transaction\_date' is set

if ('transaction\_date' not in params or

params['transaction\_date'] is None):

raise ValueError("Missing the required parameter `transaction\_date` when calling `api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get`") # noqa: E501

# verify the required parameter 'account' is set

if ('account' not in params or

params['account'] is None):

raise ValueError("Missing the required parameter `account` when calling `api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get`") # noqa: E501

# verify the required parameter 'password' is set

if ('password' not in params or

params['password'] is None):

raise ValueError("Missing the required parameter `password` when calling `api\_v1\_reporting\_get\_transactions\_by\_date\_transaction\_date\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'transaction\_date' in params:

path\_params['transactionDate'] = params['transaction\_date'] # noqa: E501

query\_params = []

header\_params = {}

if 'account' in params:

header\_params['Account'] = params['account'] # noqa: E501

if 'password' in params:

header\_params['Password'] = params['password'] # noqa: E501

form\_params = []

local\_var\_files = {}

body\_params = None

# HTTP header `Accept`

header\_params['Accept'] = self.api\_client.select\_header\_accept(

['text/plain', 'application/json', 'text/json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Reporting/GetTransactionsByDate/{transactionDate}', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[QueryTransaction]', # noqa: E501

auth\_settings=auth\_settings,

async\_req=params.get('async\_req'),

\_return\_http\_data\_only=params.get('\_return\_http\_data\_only'),

\_preload\_content=params.get('\_preload\_content', True),

\_request\_timeout=params.get('\_request\_timeout'),

collection\_formats=collection\_formats)