import ApiClient from "../ApiClient";

import CheckTransaction from '../model/CheckTransaction';

import Record from '../model/Record';

import Response from '../model/Response';

import Transaction from '../model/Transaction';

export default class VaultApi {

constructor(apiClient) {

this.apiClient = apiClient || ApiClient.instance;

}

deleteVaultCardByID(vaultCardID, account, password, callback) {

const postBody = null;

const pathParams = {

'vaultCardID': vaultCardID

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

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

const returnType = Response;

return this.apiClient.callApi(

`/api/v1/Vault/DeleteVaultCardByID/${vaultCardID}`,

'POST',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

deleteVaultCheckByID(vaultCheckID, account, password, callback) {

const postBody = null;

const pathParams = {

'vaultCheckID': vaultCheckID

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

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

const returnType = Response;

return this.apiClient.callApi(

'/api/v1/Vault/DeleteVaultCheckByID/{vaultCheckID}',

'POST',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

getVaultRecordReference(reference, account, password, callback) {

const postBody = null;

const pathParams = {

'reference': reference

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = [];

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

const returnType = Record;

return this.apiClient.callApi(

'/api/v1/Vault/GetVaultRecord/{reference}',

'GET',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

saveVaultCardVaultId(vaultID, account, password, opts, callback) {

opts = opts || {};

const postBody = opts['body'];

const pathParams = {

'vaultID': vaultID

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = ['application/json', 'text/json', 'application/\_\*+json'];

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

const returnType = Response;

return this.apiClient.callApi(

'/api/v1/Vault/SaveVaultCard/{vaultID}',

'POST',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

saveVault(account, password, opts, callback) {

opts = opts || {};

const postBody = opts['body'];

const pathParams = {

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = ['application/json', 'text/json', 'application/\_\*+json'];

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

const returnType = [Response];

return this.apiClient.callApi(

'/api/v1/Vault/SaveVault',

'POST',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

searchVault(account, password, opts, callback) {

opts = opts || {};

const postBody = opts['body'];

const pathParams = {

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = ['application/json', 'text/json', 'application/\_\*+json'];

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

const returnType = [Record];

return this.apiClient.callApi(

'/api/v1/Vault/SearchVault',

'POST',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

submitCheckWithVaultCheckId(vaultCheckID, account, password, opts, callback) {

opts = opts || {};

const postBody = opts['body'];

const pathParams = {

'vaultCheckID': vaultCheckID

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = ['application/json', 'text/json', 'application/\_\*+json'];

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

const returnType = CheckTransaction;

return this.apiClient.callApi(

'/api/v1/Vault/SubmitCheckWithVaultCheckID/{vaultCheckID}',

'POST',

pathParams,

headerParams,

postBody,

contentTypes,

accepts,

returnType,

callback

);

}

submitWithVaultCardId(vaultCardID, account, password, opts, callback) {

opts = opts || {};

const postBody = opts['body'];

const pathParams = {

'vaultCardID': vaultCardID

};

const headerParams = {

'Account': account,

'Password': password

};

const contentTypes = ['application/json', 'text/json', 'application/\_\*+json'];

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

const returnType = Transaction;

return this.apiClient.callApi(

'/api/v1/Vault/SubmitWithVaultCardID/{vaultCardID}',

'POST',

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 IVaultApi : IApiAccessor

{

System.Threading.Tasks.Task<Response> ApiV1VaultDeleteVaultCardByIDVaultCardIDPostAsync(int? vaultCardID, string account, string password);

System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo(int? vaultCardID, string account, string password);

System.Threading.Tasks.Task<Response> ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsync(int? vaultCheckID, string account, string password);

System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo(int? vaultCheckID, string account, string password);

System.Threading.Tasks.Task<Record> ApiV1VaultGetVaultRecordReferenceGetAsync(string reference, string account, string password);

System.Threading.Tasks.Task<ApiResponse<Record>> ApiV1VaultGetVaultRecordReferenceGetAsyncWithHttpInfo(string reference, string account, string password);

System.Threading.Tasks.Task<Response> ApiV1VaultSaveVaultCardVaultIDPostAsync(string account, string password, int? vaultID, VaultCreditCard body = null);

System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo(string account, string password, int? vaultID, VaultCreditCard body = null);

System.Threading.Tasks.Task<List<Response>> ApiV1VaultSaveVaultPostAsync(string account, string password, Record body = null);

System.Threading.Tasks.Task<ApiResponse<List<Response>>> ApiV1VaultSaveVaultPostAsyncWithHttpInfo(string account, string password, Record body = null);

System.Threading.Tasks.Task<List<Record>> ApiV1VaultSearchVaultPostAsync(string account, string password, SearchVault body = null);

System.Threading.Tasks.Task<ApiResponse<List<Record>>> ApiV1VaultSearchVaultPostAsyncWithHttpInfo(string account, string password, SearchVault body = null);

System.Threading.Tasks.Task<CheckTransaction> ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsync(string account, string password, int? vaultCheckID, CheckTransaction body = null);

System.Threading.Tasks.Task<ApiResponse<CheckTransaction>> ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo(string account, string password, int? vaultCheckID, CheckTransaction body = null);

System.Threading.Tasks.Task<Transaction> ApiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsync(string account, string password, int? vaultCardID, Transaction body = null);

System.Threading.Tasks.Task<ApiResponse<Transaction>> ApiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo(string account, string password, int? vaultCardID, Transaction body = null);

}

public partial class VaultApi : IVaultApi

{

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

public VaultApi(String basePath)

{

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

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

}

public VaultApi()

{

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

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

}

public VaultApi(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 async System.Threading.Tasks.Task<Response> ApiV1VaultDeleteVaultCardByIDVaultCardIDPostAsync(int? vaultCardID, string account, string password)

{

ApiResponse<Response> localVarResponse = await ApiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo(vaultCardID, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo(int? vaultCardID, string account, string password)

{

// verify the required parameter 'vaultCardID' is set

if (vaultCardID == null)

throw new ApiException(400, "Missing required parameter 'vaultCardID' when calling VaultApi->ApiV1VaultDeleteVaultCardByIDVaultCardIDPost");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultDeleteVaultCardByIDVaultCardIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultDeleteVaultCardByIDVaultCardIDPost");

var localVarPath = "/api/v1/Vault/DeleteVaultCardByID/{vaultCardID}";

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 (vaultCardID != null) localVarPathParams.Add("vaultCardID", this.Configuration.ApiClient.ParameterToString(vaultCardID)); // 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.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultDeleteVaultCardByIDVaultCardIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

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

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

public async System.Threading.Tasks.Task<Response> ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsync(int? vaultCheckID, string account, string password)

{

ApiResponse<Response> localVarResponse = await ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo(vaultCheckID, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo(int? vaultCheckID, string account, string password)

{

// verify the required parameter 'vaultCheckID' is set

if (vaultCheckID == null)

throw new ApiException(400, "Missing required parameter 'vaultCheckID' when calling VaultApi->ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost");

var localVarPath = "/api/v1/Vault/DeleteVaultCheckByID/{vaultCheckID}";

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 (vaultCheckID != null) localVarPathParams.Add("vaultCheckID", this.Configuration.ApiClient.ParameterToString(vaultCheckID)); // 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.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

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

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

public async System.Threading.Tasks.Task<Record> ApiV1VaultGetVaultRecordReferenceGetAsync(string reference, string account, string password)

{

ApiResponse<Record> localVarResponse = await ApiV1VaultGetVaultRecordReferenceGetAsyncWithHttpInfo(reference, account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Record>> ApiV1VaultGetVaultRecordReferenceGetAsyncWithHttpInfo(string reference, string account, string password)

{

// verify the required parameter 'reference' is set

if (reference == null)

throw new ApiException(400, "Missing required parameter 'reference' when calling VaultApi->ApiV1VaultGetVaultRecordReferenceGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultGetVaultRecordReferenceGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultGetVaultRecordReferenceGet");

var localVarPath = "/api/v1/Vault/GetVaultRecord/{reference}";

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 (reference != null) localVarPathParams.Add("reference", this.Configuration.ApiClient.ParameterToString(reference)); // 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("ApiV1VaultGetVaultRecordReferenceGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Record>(localVarStatusCode,

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

(Record)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Record)));

}

public async System.Threading.Tasks.Task<Response> ApiV1VaultSaveVaultCardVaultIDPostAsync(string account, string password, int? vaultID, VaultCreditCard body = null)

{

ApiResponse<Response> localVarResponse = await ApiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo(account, password, vaultID, body);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo(string account, string password, int? vaultID, VaultCreditCard body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

// verify the required parameter 'vaultID' is set

if (vaultID == null)

throw new ApiException(400, "Missing required parameter 'vaultID' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

var localVarPath = "/api/v1/Vault/SaveVaultCard/{vaultID}";

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[] {

"application/json",

"text/json",

"application/\_\*+json"

};

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 (vaultID != null) localVarPathParams.Add("vaultID", this.Configuration.ApiClient.ParameterToString(vaultID)); // 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

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

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

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

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSaveVaultCardVaultIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

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

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

public async System.Threading.Tasks.Task<List<Response>> ApiV1VaultSaveVaultPostAsync(string account, string password, Record body = null)

{

ApiResponse<List<Response>> localVarResponse = await ApiV1VaultSaveVaultPostAsyncWithHttpInfo(account, password, body);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<Response>>> ApiV1VaultSaveVaultPostAsyncWithHttpInfo(string account, string password, Record body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSaveVaultPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSaveVaultPost");

var localVarPath = "/api/v1/Vault/SaveVault";

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[] {

"application/json",

"text/json",

"application/\_\*+json"

};

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

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

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

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

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSaveVaultPost", localVarResponse);

if (exception != null) throw exception;

}

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

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

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

}

public async System.Threading.Tasks.Task<List<Record>> ApiV1VaultSearchVaultPostAsync(string account, string password, SearchVault body = null)

{

ApiResponse<List<Record>> localVarResponse = await ApiV1VaultSearchVaultPostAsyncWithHttpInfo(account, password, body);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<List<Record>>> ApiV1VaultSearchVaultPostAsyncWithHttpInfo(string account, string password, SearchVault body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSearchVaultPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSearchVaultPost");

var localVarPath = "/api/v1/Vault/SearchVault";

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[] {

"application/json",

"text/json",

"application/\_\*+json"

};

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

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

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

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

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSearchVaultPost", localVarResponse);

if (exception != null) throw exception;

}

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

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

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

}

public async System.Threading.Tasks.Task<CheckTransaction> ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsync(string account, string password, int? vaultCheckID, CheckTransaction body = null)

{

ApiResponse<CheckTransaction> localVarResponse = await ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo(account, password, vaultCheckID, body);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<CheckTransaction>> ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo(string account, string password, int? vaultCheckID, CheckTransaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost");

// verify the required parameter 'vaultCheckID' is set

if (vaultCheckID == null)

throw new ApiException(400, "Missing required parameter 'vaultCheckID' when calling VaultApi->ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost");

var localVarPath = "/api/v1/Vault/SubmitCheckWithVaultCheckID/{vaultCheckID}";

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[] {

"application/json",

"text/json",

"application/\_\*+json"

};

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 (vaultCheckID != null) localVarPathParams.Add("vaultCheckID", this.Configuration.ApiClient.ParameterToString(vaultCheckID)); // 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

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

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

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

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<CheckTransaction>(localVarStatusCode,

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

(CheckTransaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(CheckTransaction)));

}

public async System.Threading.Tasks.Task<Transaction> ApiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsync(string account, string password, int? vaultCardID, Transaction body = null)

{

ApiResponse<Transaction> localVarResponse = await ApiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo(account, password, vaultCardID, body);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Transaction>> ApiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo(string account, string password, int? vaultCardID, Transaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost");

// verify the required parameter 'vaultCardID' is set

if (vaultCardID == null)

throw new ApiException(400, "Missing required parameter 'vaultCardID' when calling VaultApi->ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost");

var localVarPath = "/api/v1/Vault/SubmitWithVaultCardID/{vaultCardID}";

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[] {

"application/json",

"text/json",

"application/\_\*+json"

};

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 (vaultCardID != null) localVarPathParams.Add("vaultCardID", this.Configuration.ApiClient.ParameterToString(vaultCardID)); // 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

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

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

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

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Transaction>(localVarStatusCode,

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

(Transaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Transaction)));

}

}

}

<?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 VaultApi

{

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 apiV1VaultDeleteVaultCardByIDVaultCardIDPostAsync($vault\_card\_id, $account, $password)

{

return $this->apiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo($vault\_card\_id, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo($vault\_card\_id, $account, $password)

{

$returnType = '\Swagger\Client\Model\Response';

$request = $this->apiV1VaultDeleteVaultCardByIDVaultCardIDPostRequest($vault\_card\_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()

);

}

);

}

public function apiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsync($vault\_check\_id, $account, $password)

{

return $this->apiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo($vault\_check\_id, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo($vault\_check\_id, $account, $password)

{

$returnType = '\Swagger\Client\Model\Response';

$request = $this->apiV1VaultDeleteVaultCheckByIDVaultCheckIDPostRequest($vault\_check\_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()

);

}

);

}

public function apiV1VaultGetVaultRecordReferenceGetAsync($reference, $account, $password)

{

return $this->apiV1VaultGetVaultRecordReferenceGetAsyncWithHttpInfo($reference, $account, $password)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultGetVaultRecordReferenceGetAsyncWithHttpInfo($reference, $account, $password)

{

$returnType = '\Swagger\Client\Model\Record';

$request = $this->apiV1VaultGetVaultRecordReferenceGetRequest($reference, $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()

);

}

);

}

public function apiV1VaultSaveVaultCardVaultIDPostAsync($account, $password, $vault\_id, $body = null)

{

return $this->apiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo($account, $password, $vault\_id, $body)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo($account, $password, $vault\_id, $body = null)

{

$returnType = '\Swagger\Client\Model\Response';

$request = $this->apiV1VaultSaveVaultCardVaultIDPostRequest($account, $password, $vault\_id, $body);

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()

);

}

);

}

public function apiV1VaultSaveVaultPostAsync($account, $password, $body = null)

{

return $this->apiV1VaultSaveVaultPostAsyncWithHttpInfo($account, $password, $body)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultSaveVaultPostAsyncWithHttpInfo($account, $password, $body = null)

{

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

$request = $this->apiV1VaultSaveVaultPostRequest($account, $password, $body);

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()

);

}

);

}

public function apiV1VaultSearchVaultPostAsync($account, $password, $body = null)

{

return $this->apiV1VaultSearchVaultPostAsyncWithHttpInfo($account, $password, $body)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultSearchVaultPostAsyncWithHttpInfo($account, $password, $body = null)

{

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

$request = $this->apiV1VaultSearchVaultPostRequest($account, $password, $body);

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()

);

}

);

}

public function apiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsync($account, $password, $vault\_check\_id, $body = null)

{

return $this->apiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo($account, $password, $vault\_check\_id, $body)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo($account, $password, $vault\_check\_id, $body = null)

{

$returnType = '\Swagger\Client\Model\CheckTransaction';

$request = $this->apiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostRequest($account, $password, $vault\_check\_id, $body);

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()

);

}

);

}

public function apiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsync($account, $password, $vault\_card\_id, $body = null)

{

return $this->apiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo($account, $password, $vault\_card\_id, $body)

->then(

function ($response) {

return $response[0];

}

);

}

public function apiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo($account, $password, $vault\_card\_id, $body = null)

{

$returnType = '\Swagger\Client\Model\Transaction';

$request = $this->apiV1VaultSubmitWithVaultCardIDVaultCardIDPostRequest($account, $password, $vault\_card\_id, $body);

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 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 VaultApi(object):

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

if api\_client is None:

api\_client = ApiClient()

self.api\_client = api\_client

def api\_v1\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post(self, vault\_card\_id, account, password, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post\_with\_http\_info(vault\_card\_id, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post\_with\_http\_info(vault\_card\_id, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post\_with\_http\_info(self, vault\_card\_id, account, password, \*\*kwargs): # noqa: E501

all\_params = ['vault\_card\_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\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'vault\_card\_id' is set

if ('vault\_card\_id' not in params or

params['vault\_card\_id'] is None):

raise ValueError("Missing the required parameter `vault\_card\_id` when calling `api\_v1\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post`") # 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\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post`") # 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\_vault\_delete\_vault\_card\_by\_id\_vault\_card\_id\_post`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'vault\_card\_id' in params:

path\_params['vaultCardID'] = params['vault\_card\_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/Vault/DeleteVaultCardByID/{vaultCardID}', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='Response', # 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\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post(self, vault\_check\_id, account, password, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post\_with\_http\_info(vault\_check\_id, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post\_with\_http\_info(vault\_check\_id, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post\_with\_http\_info(self, vault\_check\_id, account, password, \*\*kwargs): # noqa: E501

all\_params = ['vault\_check\_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\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'vault\_check\_id' is set

if ('vault\_check\_id' not in params or

params['vault\_check\_id'] is None):

raise ValueError("Missing the required parameter `vault\_check\_id` when calling `api\_v1\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post`") # 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\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post`") # 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\_vault\_delete\_vault\_check\_by\_id\_vault\_check\_id\_post`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'vault\_check\_id' in params:

path\_params['vaultCheckID'] = params['vault\_check\_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/Vault/DeleteVaultCheckByID/{vaultCheckID}', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='Response', # 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\_vault\_get\_vault\_record\_reference\_get(self, reference, account, password, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_get\_vault\_record\_reference\_get\_with\_http\_info(reference, account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_get\_vault\_record\_reference\_get\_with\_http\_info(reference, account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_get\_vault\_record\_reference\_get\_with\_http\_info(self, reference, account, password, \*\*kwargs): # noqa: E501

all\_params = ['reference', '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\_vault\_get\_vault\_record\_reference\_get" % key

)

params[key] = val

del params['kwargs']

# verify the required parameter 'reference' is set

if ('reference' not in params or

params['reference'] is None):

raise ValueError("Missing the required parameter `reference` when calling `api\_v1\_vault\_get\_vault\_record\_reference\_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\_vault\_get\_vault\_record\_reference\_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\_vault\_get\_vault\_record\_reference\_get`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'reference' in params:

path\_params['reference'] = params['reference'] # 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/Vault/GetVaultRecord/{reference}', 'GET',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='Record', # 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\_vault\_save\_vault\_card\_vault\_id\_post(self, account, password, vault\_id, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_save\_vault\_card\_vault\_id\_post\_with\_http\_info(account, password, vault\_id, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_save\_vault\_card\_vault\_id\_post\_with\_http\_info(account, password, vault\_id, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_save\_vault\_card\_vault\_id\_post\_with\_http\_info(self, account, password, vault\_id, \*\*kwargs): # noqa: E501

all\_params = ['account', 'password', 'vault\_id', 'body'] # 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\_vault\_save\_vault\_card\_vault\_id\_post" % 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\_vault\_save\_vault\_card\_vault\_id\_post`") # 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\_vault\_save\_vault\_card\_vault\_id\_post`") # noqa: E501

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

if ('vault\_id' not in params or

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

raise ValueError("Missing the required parameter `vault\_id` when calling `api\_v1\_vault\_save\_vault\_card\_vault\_id\_post`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'vault\_id' in params:

path\_params['vaultID'] = params['vault\_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

if 'body' in params:

body\_params = params['body']

# HTTP header `Accept`

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

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

# HTTP header `Content-Type`

header\_params['Content-Type'] = self.api\_client.select\_header\_content\_type( # noqa: E501

['application/json', 'text/json', 'application/\*+json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Vault/SaveVaultCard/{vaultID}', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='Response', # 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\_vault\_save\_vault\_post(self, account, password, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_save\_vault\_post\_with\_http\_info(account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_save\_vault\_post\_with\_http\_info(account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_save\_vault\_post\_with\_http\_info(self, account, password, \*\*kwargs): # noqa: E501

all\_params = ['account', 'password', 'body'] # 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\_vault\_save\_vault\_post" % 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\_vault\_save\_vault\_post`") # 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\_vault\_save\_vault\_post`") # 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

if 'body' in params:

body\_params = params['body']

# HTTP header `Accept`

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

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

# HTTP header `Content-Type`

header\_params['Content-Type'] = self.api\_client.select\_header\_content\_type( # noqa: E501

['application/json', 'text/json', 'application/\*+json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Vault/SaveVault', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[Response]', # 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\_vault\_search\_vault\_post(self, account, password, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_search\_vault\_post\_with\_http\_info(account, password, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_search\_vault\_post\_with\_http\_info(account, password, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_search\_vault\_post\_with\_http\_info(self, account, password, \*\*kwargs): # noqa: E501

all\_params = ['account', 'password', 'body'] # 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\_vault\_search\_vault\_post" % 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\_vault\_search\_vault\_post`") # 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\_vault\_search\_vault\_post`") # 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

if 'body' in params:

body\_params = params['body']

# HTTP header `Accept`

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

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

# HTTP header `Content-Type`

header\_params['Content-Type'] = self.api\_client.select\_header\_content\_type( # noqa: E501

['application/json', 'text/json', 'application/\*+json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Vault/SearchVault', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='list[Record]', # 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\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post(self, account, password, vault\_check\_id, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post\_with\_http\_info(account, password, vault\_check\_id, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post\_with\_http\_info(account, password, vault\_check\_id, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post\_with\_http\_info(self, account, password, vault\_check\_id, \*\*kwargs): # noqa: E501

all\_params = ['account', 'password', 'vault\_check\_id', 'body'] # 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\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post" % 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\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post`") # 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\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post`") # noqa: E501

# verify the required parameter 'vault\_check\_id' is set

if ('vault\_check\_id' not in params or

params['vault\_check\_id'] is None):

raise ValueError("Missing the required parameter `vault\_check\_id` when calling `api\_v1\_vault\_submit\_check\_with\_vault\_check\_id\_vault\_check\_id\_post`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'vault\_check\_id' in params:

path\_params['vaultCheckID'] = params['vault\_check\_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

if 'body' in params:

body\_params = params['body']

# HTTP header `Accept`

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

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

# HTTP header `Content-Type`

header\_params['Content-Type'] = self.api\_client.select\_header\_content\_type( # noqa: E501

['application/json', 'text/json', 'application/\*+json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Vault/SubmitCheckWithVaultCheckID/{vaultCheckID}', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='CheckTransaction', # 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\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post(self, account, password, vault\_card\_id, \*\*kwargs): # noqa: E501

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

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

return self.api\_v1\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post\_with\_http\_info(account, password, vault\_card\_id, \*\*kwargs) # noqa: E501

else:

(data) = self.api\_v1\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post\_with\_http\_info(account, password, vault\_card\_id, \*\*kwargs) # noqa: E501

return data

def api\_v1\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post\_with\_http\_info(self, account, password, vault\_card\_id, \*\*kwargs): # noqa: E501

all\_params = ['account', 'password', 'vault\_card\_id', 'body'] # 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\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post" % 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\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post`") # 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\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post`") # noqa: E501

# verify the required parameter 'vault\_card\_id' is set

if ('vault\_card\_id' not in params or

params['vault\_card\_id'] is None):

raise ValueError("Missing the required parameter `vault\_card\_id` when calling `api\_v1\_vault\_submit\_with\_vault\_card\_id\_vault\_card\_id\_post`") # noqa: E501

collection\_formats = {}

path\_params = {}

if 'vault\_card\_id' in params:

path\_params['vaultCardID'] = params['vault\_card\_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

if 'body' in params:

body\_params = params['body']

# HTTP header `Accept`

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

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

# HTTP header `Content-Type`

header\_params['Content-Type'] = self.api\_client.select\_header\_content\_type( # noqa: E501

['application/json', 'text/json', 'application/\*+json']) # noqa: E501

# Authentication setting

auth\_settings = [] # noqa: E501

return self.api\_client.call\_api(

'/api/v1/Vault/SubmitWithVaultCardID/{vaultCardID}', 'POST',

path\_params,

query\_params,

header\_params,

body=body\_params,

post\_params=form\_params,

files=local\_var\_files,

response\_type='Transaction', # 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)