```
use std::collections::HashMap;

use log::{debug, error, info};
use reqwest::blocking::Client;
use serde::de::DeserializeOwned;
use serde::Serialize;

use crate::client::jsonrpc::{ZabbixApiRequest, ZabbixApiResponse};
use crate::client::post::send_post_request;
use crate::client::ZabbixApiClient;
use crate::error::ZabbixApiError;
use crate::host::{ZabbixHost, ZabbixHostGroup};
use crate::host::create::{CreateHostGroupRequest, CreateHostGroupResponse, CreateHostRequest, CreateHostRespo
nse};
use crate::item::create::{CreateItemRequest, CreateItemResponse};
use crate::item::ZabbixItem;
use crate::trigger::create::{CreateTriggerRequest, CreateTriggerResponse};
use crate::trigger::ZabbixTrigger;
use crate::webscenario::create::{CreateWebScenarioRequest, CreateWebScenarioResponse};
use crate::webscenario::ZabbixWebScenario;

const JSON_RPC_VERSION: &str = "2.0";

/// Zabbix API Client implementation for [Zabbix API v6](https://www.zabbix.com/documentation/6.0/en/manual/a
pi)
#[derive(Debug,Clone)]
pub struct ZabbixApiV6Client {
    client: Client,
    api_endpoint_url: String
}

impl ZabbixApiV6Client {
    pub fn new(client: Client, api_endpoint_url: &str) -> ZabbixApiV6Client {
        ZabbixApiV6Client {
            client,
            api_endpoint_url: api_endpoint_url.to_string()
        }
    }
}

impl ZabbixApiClient for ZabbixApiV6Client {

    /// # get_api_info
    ///
    /// Implements `ZabbixApiClient::get_api_info`.
    ///
    /// See the trait documentation for more details.
    fn get_api_info(&self) -> Result<String, ZabbixApiError> {
        let request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "apiinfo.version".to_string(),
            params: HashMap::<String,String>::new(),
            id: 1,
            auth: None,
        };

        match send_post_request(&self.client, &self.api_endpoint_url, request) {
            Ok(response_body) => {
                let response = serde_json::from_str::<ZabbixApiResponse<String>>(&response_body)?;

                match response.result {
                    Some(api_version) => {
                        info!("zabbix api version: '{api_version}'");
                        Ok(api_version)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
```

```
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # get_auth_session
    ///
    /// Implements `ZabbixApiClient::get_auth_session`.
    ///
    /// See the trait documentation for more details.
    fn get_auth_session(&self,  login: &str, token: &str) -> Result<String, ZabbixApiError> {
        info!("getting auth session for user '{login}'..");

        let params = HashMap::from([
            ("username".to_string(), login.to_string()),
            ("password".to_string(), token.to_string()),
        ]);

        let request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "user.login".to_string(),
            params,
            id: 1,
            auth: None,
        };

        match send_post_request(&self.client, &self.api_endpoint_url, request) {
            Ok(response_body) => {
                let response = serde_json::from_str::<ZabbixApiResponse<String>>(&response_body)?;

                match response.result {
                    Some(session) => {
                        info!("auth ok");
                        Ok(session)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # raw_api_call
    ///
    /// Implements `ZabbixApiClient::raw_api_call`.
    ///
    /// See the trait documentation for more details.
    fn raw_api_call<P: Serialize, R: DeserializeOwned>(&self, session: &str,
                                        method: &str, params: &P) -> Result<ZabbixApiResponse<R>, ZabbixAp
iError> {
        info!("call api method '{method}'..");

        let request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: method.to_string(),
            params,
            id: 1,
            auth: Some(session.to_string()),
        };
```

```
        match send_post_request(&self.client, &self.api_endpoint_url, request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<R>>(&response_body)?;

                match response.result {
                    Some(_) => {
                        info!("api method '{method}' has been successfully called");
                        Ok(response)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # get_host_groups
    ///
    /// Implements `ZabbixApiClient::get_host_groups`.
    ///
    /// See the trait documentation for more details.
    fn get_host_groups<P: Serialize>(&self, session: &str, params: &P) -> Result<Vec<ZabbixHostGroup>, Zabbix
ApiError> {
        info!("getting host groups with params");

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "hostgroup.get".to_string(),
            params,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<Vec<ZabbixHostGroup>>>(&response_body
)?;

                match response.result {
                    Some(results) => {
                        info!("host groups found: {:?}", results);
                        Ok(results)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
```

```
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # get_hosts
    ///
    /// Implements `ZabbixApiClient::get_hosts`.
    ///
    /// See the trait documentation for more details.
    fn get_hosts<P: Serialize>(&self, session: &str, params: &P) -> Result<Vec<ZabbixHost>, ZabbixApiError> {
        info!("getting hosts with params");

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "host.get".to_string(),
            params,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<Vec<ZabbixHost>>>(&response_body)?;

                match response.result {
                    Some(results) => {
                        info!("hosts found: {:?}", results);
                        Ok(results)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # get_items
    ///
    /// Implements `ZabbixApiClient::get_items`.
    ///
    /// See the trait documentation for more details.
    fn get_items<P: Serialize>(&self, session: &str, params: &P) -> Result<Vec<ZabbixItem>, ZabbixApiError> {
        info!("getting items with params");

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "item.get".to_string(),
            params,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
```

```
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<Vec<ZabbixItem>>>(&response_body)?;

                match response.result {
                    Some(results) => {
                        info!("hosts found: {:?}", results);
                        Ok(results)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # get_triggers
    ///
    /// Implements `ZabbixApiClient::get_triggers`.
    ///
    /// See the trait documentation for more details.
    fn get_triggers<P: Serialize>(&self, session: &str, params: &P) -> Result<Vec<ZabbixTrigger>, ZabbixApiEr
ror> {
        info!("getting triggers..");

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "trigger.get".to_string(),
            params,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<Vec<ZabbixTrigger>>>(&response_body)?
;

                match response.result {
                    Some(results) => {
                        info!("hosts found: {:?}", results);
                        Ok(results)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
```

```
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # get_webscenarios
    ///
    /// Implements `ZabbixApiClient::get_webscenarios`.
    ///
    /// See the trait documentation for more details.
    fn get_webscenarios<P: Serialize>(&self, session: &str, params: &P) -> Result<Vec<ZabbixWebScenario>, Zab
bixApiError> {
        info!("getting web-scenarios..");

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "httptest.get".to_string(),
            params,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<Vec<ZabbixWebScenario>>>(&response_bo
dy)?;

                match response.result {
                    Some(results) => {
                        info!("hosts found: {:?}", results);
                        Ok(results)
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # create_host_group
    ///
    /// Implements `ZabbixApiClient::create_host_group`.
    ///
    /// See the trait documentation for more details.
    fn create_host_group(&self, session: &str, request: &CreateHostGroupRequest) -> Result<u32, ZabbixApiErro
r> {
        info!("creating host group '{}'..", request.name);

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "hostgroup.create".to_string(),
            params: request,
            id: 1,
            auth: Some(session.to_string()),
        };
```

```
        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<CreateHostGroupResponse>>(&response_b
ody)?;

                match response.result {
                    Some(result) => {
                        info!("host group '{}' has been created", request.name);

                        match result.group_ids.first() {
                            Some(id) => {
                                id.parse::<u32>().map_err(|_| ZabbixApiError::Error)
                            }
                            None => {
                                error!("unexpected error, server returned empty id list");
                                Err(ZabbixApiError::Error)
                            }
                        }
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # create_host
    ///
    /// Implements `ZabbixApiClient::create_host`.
    ///
    /// See the trait documentation for more details.
    fn create_host(&self, session: &str, request: &CreateHostRequest) -> Result<u32, ZabbixApiError> {
        info!("creating host '{}'..", request.host);

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "host.create".to_string(),
            params: request,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<CreateHostResponse>>(&response_body)?
;

                match response.result {
                    Some(result) => {

                        info!("host '{}' has been created", request.host);

                        match result.host_ids.first() {
                            Some(host_id) => {
                                host_id.parse::<u32>().map_err(|_| ZabbixApiError::Error)
```

```
                            }
                            None => {
                                error!("unexpected error, server returned empty id list");
                                Err(ZabbixApiError::Error)
                            }
                        }
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # create_item
    ///
    /// Implements `ZabbixApiClient::create_item`.
    ///
    /// See the trait documentation for more details.
    fn create_item(&self, session: &str, request: &CreateItemRequest) -> Result<u32, ZabbixApiError> {
        info!("creating item with key '{}' for host id {}..", request.key_, request.host_id);

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "item.create".to_string(),
            params: request,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<CreateItemResponse>>(&response_body)?
;

                match response.result {
                    Some(result) => {

                        info!("item '{}' has been created", request.key_);

                        match result.item_ids.first() {
                            Some(host_id) => {
                                host_id.parse::<u32>().map_err(|_| ZabbixApiError::Error)
                            }
                            None => {
                                error!("unexpected error, server returned empty id list");
                                Err(ZabbixApiError::Error)
                            }
                        }
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
```

```
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # create_trigger
    ///
    /// Implements `ZabbixApiClient::create_trigger`.
    ///
    /// See the trait documentation for more details.
    fn create_trigger(&self, session: &str, request: &CreateTriggerRequest) -> Result<u32, ZabbixApiError> {
        info!("creating trigger '{}' with expression '{}'..", request.description, request.expression);

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "trigger.create".to_string(),
            params: request,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<CreateTriggerResponse>>(&response_bod
y)?;

                match response.result {
                    Some(result) => {

                        info!("trigger '{}' has been created", request.description);

                        match result.trigger_ids.first() {
                            Some(host_id) => {
                                host_id.parse::<u32>().map_err(|_| ZabbixApiError::Error)
                            }
                            None => {
                                error!("unexpected error, server returned empty id list");
                                Err(ZabbixApiError::Error)
                            }
                        }
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }

    /// # create_webscenario
    ///
    /// Implements `ZabbixApiClient::create_webscenario`.
    ///
```

```
    /// See the trait documentation for more details.
    fn create_webscenario(&self, session: &str, request: &CreateWebScenarioRequest) -> Result<u32, ZabbixApiE
rror> {
        info!("creating web-scenario '{}' for host id '{}'..", request.name, request.host_id);

        let api_request = ZabbixApiRequest {
            jsonrpc: JSON_RPC_VERSION.to_string(),
            method: "httptest.create".to_string(),
            params: request,
            id: 1,
            auth: Some(session.to_string()),
        };

        match send_post_request(&self.client, &self.api_endpoint_url, api_request) {
            Ok(response_body) => {
                debug!("[response body]");
                debug!("{response_body}");
                debug!("[/response body]");

                let response = serde_json::from_str::<ZabbixApiResponse<CreateWebScenarioResponse>>(&response
_body)?;

                match response.result {
                    Some(result) => {

                        info!("web-scenario '{}' has been created", request.name);

                        match result.http_test_ids.first() {
                            Some(host_id) => {
                                host_id.parse::<u32>().map_err(|_| ZabbixApiError::Error)
                            }
                            None => {
                                error!("unexpected error, server returned empty id list");
                                Err(ZabbixApiError::Error)
                            }
                        }
                    }
                    None => {
                        match response.error {
                            Some(error) => {
                                error!("{:?}", error);

                                Err(ZabbixApiError::ApiCallError {
                                    zabbix: error,
                                })
                            }
                            None => Err(ZabbixApiError::BadRequestError)
                        }
                    }
                }
            }
            Err(e) => {
                error!("{}", e);
                Err(e)
            }
        }
    }
}

#[cfg(test)]
mod tests {
    use std::error::Error;

    use log::{error, info};
    use reqwest::blocking::Client;
    use serde::Serialize;

    use crate::client::v6::ZabbixApiV6Client;
    use crate::client::ZabbixApiClient;
    use crate::host::get::{GetHostGroupsRequest, GetHostsRequest};
    use crate::host::ZabbixHost;
    use crate::item::create::CreateItemRequest;
    use crate::item::get::GetItemsRequestById;
    use crate::tests::{get_random_string, init_logging};
    use crate::tests::builder::TestEnvBuilder;
    use crate::tests::integration::{are_integration_tests_enabled, get_integration_tests_config};
    use crate::trigger::create::CreateTriggerRequest;
```

```
    use crate::trigger::get::GetTriggerByIdRequest;
    use crate::webscenario::create::CreateWebScenarioRequest;
    use crate::webscenario::get::GetWebScenarioByIdRequest;
    use crate::webscenario::ZabbixWebScenarioStep;
    use crate::ZABBIX_EXTEND_PROPERTY_VALUE;

    #[test]
    fn get_api_info() {
        if are_integration_tests_enabled() {
            let test_env = TestEnvBuilder::build();

            match test_env.client.get_api_info() {
                Ok(result) => {
                    assert!(!result.is_empty())
                }
                Err(e) => {
                    error!("error: {}", e);
                    panic!("unexpected error")
                }
            }
        }
    }

    #[test]
    fn session_should_be_returned() {
        init_logging();

        if are_integration_tests_enabled() {
            let http_client = Client::new();

            let tests_config = get_integration_tests_config();

            let client = ZabbixApiV6Client::new(http_client, &tests_config.zabbix_api_url);

            match client.get_auth_session(&tests_config.zabbix_api_user, &tests_config.zabbix_api_password) {
                Ok(session) => assert!(session.len() > 0),
                Err(e) => {
                    error!("error: {}", e);
                    panic!("unexpected error")
                }
            }
        }
    }

    #[test]
    fn raw_api_call_test() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();
            test_env.get_session();

            #[derive(Serialize)]
            struct Params {
                pub filter: Filter
            }

            #[derive(Serialize)]
            struct Filter {
                pub host: Vec<String>
            }

            let params = Params {
                filter: Filter {
                    host: vec!["Zabbix server".to_string()],
                },
            };

            match test_env.client.raw_api_call::<Params, Vec<ZabbixHost>>(
                &test_env.session, "host.get", &params) {

                Ok(response) => {
                    let results = response.result.unwrap();
                    info!("{:?}", results.first().unwrap());
                    assert_eq!(1, results.len())
                }
                Err(e) => {
```

```
                    error!("api call error: {}", e);
                    panic!("unexpected api call error")
                }
            }
        }
    }

    #[test]
    fn get_host_groups_test() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let group_name2 = get_random_string();
            let group_name3 = get_random_string();

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host_group(&group_name2)
                .create_host_group(&group_name3);

            #[derive(Serialize)]
            struct Filter {
                pub name: Vec<String>
            }

            let request = GetHostGroupsRequest {
                output: ZABBIX_EXTEND_PROPERTY_VALUE.to_string(),
                filter: Filter {
                    name: vec![group_name2.to_string()],
                },
            };

            match test_env.client.get_host_groups(&test_env.session, &request) {
                Ok(host_groups) => {
                    assert_eq!(host_groups.len(), 1);

                    let host_group = host_groups.first().unwrap();

                    assert_eq!(&host_group.name, &group_name2)
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("host group get error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn get_hosts_test() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name1 = get_random_string();
            let host_name2 = get_random_string();
            let host_name3 = get_random_string();

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host(&host_name1)
                .create_host(&host_name2)
                .create_host(&host_name3);

            #[derive(Serialize)]
            struct Filter {
                pub host: Vec<String>
            }
```

```
            let request = GetHostsRequest {
                filter: Filter {
                    host: vec![host_name2.to_string()],
                },
            };

            match test_env.client.get_hosts(&test_env.session, &request) {
                Ok(hosts) => {
                    assert_eq!(hosts.len(), 1);

                    let host = hosts.first().unwrap();

                    assert_eq!(&host.host, &host_name2)
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("host get error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn get_items_test() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name1 = get_random_string();
            let host_name2 = get_random_string();
            let host_name3 = get_random_string();
            let item_name = get_random_string();
            let item_key = format!("test{}", get_random_string());

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host(&host_name1)
                .create_host(&host_name2)
                .create_host(&host_name3)
                .create_item(&item_name, &item_key);

            #[derive(Serialize)]
            struct Search {
                pub key_: String
            }

            let request = GetItemsRequestById {
                output: ZABBIX_EXTEND_PROPERTY_VALUE.to_string(),
                with_triggers: false,
                host_ids: test_env.latest_host_id.to_string(),
                search: Search {
                    key_: item_key.to_string(),
                },
                sort_field: "name".to_string(),
            };

            match test_env.client.get_items(&test_env.session, &request) {
                Ok(items) => {
                    assert_eq!(items.len(), 1);

                    let item = items.first().unwrap();

                    assert_eq!(&item.key_, &item_key)
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("host get error: {}", e);
```

```
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn get_triggers_test() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name = get_random_string();
            let item_name = get_random_string();
            let item_key = get_random_string();
            let trigger_description = get_random_string();

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host(&host_name)
                .create_item(&item_name, &item_key)
                .create_trigger(&host_name, &trigger_description, &item_key);

            let request = GetTriggerByIdRequest {
                trigger_ids: test_env.latest_trigger_id.to_string(),
                output: ZABBIX_EXTEND_PROPERTY_VALUE.to_string(),
                select_functions: ZABBIX_EXTEND_PROPERTY_VALUE.to_string(),
            };

            match test_env.client.get_triggers(&test_env.session, &request) {
                Ok(results) => {
                    assert_eq!(results.len(), 1);
                    let result = results.first().unwrap();

                    assert_eq!(&result.description, &trigger_description)
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("host get error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn get_webscenarios_test() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name = get_random_string();
            let item_name = get_random_string();
            let item_key = get_random_string();
            let trigger_description = get_random_string();
            let webscenario_name = get_random_string();

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host(&host_name)
                .create_item(&item_name, &item_key)
                .create_trigger(&host_name, &trigger_description, &item_key)
                .create_web_scenario(&webscenario_name);

            let request = GetWebScenarioByIdRequest {
                output: ZABBIX_EXTEND_PROPERTY_VALUE.to_string(),
                select_steps: ZABBIX_EXTEND_PROPERTY_VALUE.to_string(),
                httptest_ids: test_env.latest_webscenario_id.to_string(),
            };
```

```
            match test_env.client.get_webscenarios(&test_env.session, &request) {
                Ok(results) => {
                    assert_eq!(results.len(), 1);
                    let result = results.first().unwrap();

                    assert_eq!(&result.name, &webscenario_name)
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("host get error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn create_host_group_and_host() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name = get_random_string();

            test_env.get_session()
                     .create_host_group(&group_name)
                     .create_host(&host_name);

            assert!(test_env.latest_host_group_id > 0);
            assert!(test_env.latest_host_id > 0);
        }
    }

    #[test]
    fn create_item() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name = get_random_string();

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host(&host_name);

            let item_key = get_random_string();
            let item_name = get_random_string();

            let request = CreateItemRequest {
                key_: item_key,
                name: item_name,
                host_id: test_env.latest_host_id.to_string(),
                r#type: 7,
                value_type: 4,
                interface_id: "0".to_string(),
                tags: vec![],
                delay: "30s".to_string(),
            };

            match test_env.client.create_item(
                &test_env.session, &request
            ) {
                Ok(item_id) => {
                    assert!(item_id > 0);
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }
```

```
                    error!("item create error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn create_trigger() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name = get_random_string();

            let item_name = get_random_string();
            let item_key = format!("key{}", get_random_string());

            test_env.get_session()
                    .create_host_group(&group_name)
                    .create_host(&host_name)
                    .create_item(&item_name, &item_key);

            let trigger_description = get_random_string();

            let expression = format!("last(/{host_name}/{item_key})=0");

            let request = CreateTriggerRequest {
                description: trigger_description,
                expression: expression.to_string(),
                dependencies: vec![],
                tags: vec![],
            };

            match test_env.client.create_trigger(
                &test_env.session, &request
            ) {
                Ok(trigger_id) => assert!(trigger_id > 0),
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("trigger create error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }

    #[test]
    fn create_web_scenario() {
        init_logging();

        if are_integration_tests_enabled() {
            let mut test_env = TestEnvBuilder::build();

            let group_name = get_random_string();
            let host_name = get_random_string();

            test_env.get_session()
                .create_host_group(&group_name)
                .create_host(&host_name);

            let web_scenario_name = get_random_string();

            let step = ZabbixWebScenarioStep {
                name: "Check github.com page".to_string(),
                url: "https://github.com".to_string(),
                status_codes: "200".to_string(),
                no: "0".to_string(),
            };

            let request = CreateWebScenarioRequest {
                name: web_scenario_name,
```

```
                host_id: test_env.latest_host_id.to_string(),
                steps: vec![step],
            };

            match test_env.client.create_webscenario(
                &test_env.session, &request
            ) {
                Ok(web_scenario_id) => {
                    assert!(web_scenario_id > 0);
                }
                Err(e) => {
                    if let Some(inner_source) = e.source() {
                        println!("Caused by: {}", inner_source);
                    }

                    error!("web-scenario create error: {}", e);
                    panic!("{}", e)
                }
            }
        }
    }
}
```