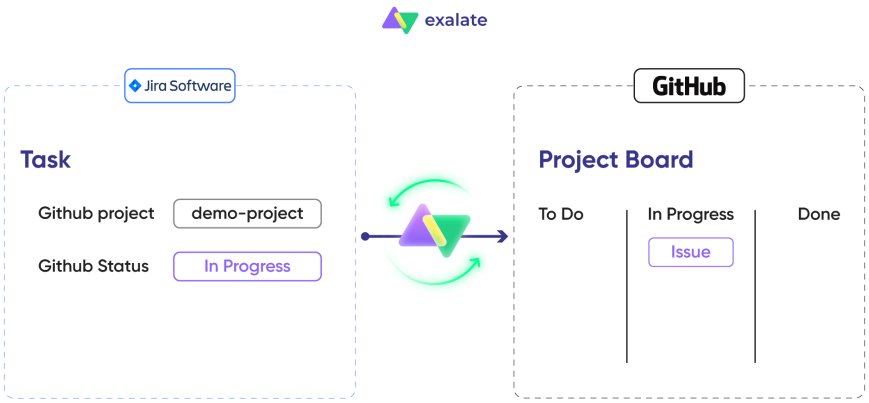
Jira GitHub: Controlling GH issue's project attributes from Jira

When you Exalate a Jira ticket to GitHub, it would not be assigned to any GitHub project (by default). In this use case, we will control the allocation of the GitHub issue to a project board, but we will base this selection from what the user selects on the Jira ticket i.e. user will select the target GitHub project for the issue from his own ticket.

The following depicts what we are trying to achieve:



We have used two custom fields within the Jira issue for the user to make the relevant selections. Once the project and status are chosen on these custom fields, Exalate would allocate the corresponding GitHub issue to the relevant project, and assign it the correct status. In order for all this to happen, the following steps are necessary:

1. Jira needs to ensure that the value of these fields is being sent to GitHub. This can be done using the following lines in Jira Outgoing Script:

Jira Outgoing

```
replica.customFields."GitHub Project" = issue.customFields."GitHub Project"
replica.customFields."GitHub Status" = issue.customFields."GitHub Status"
```

2. In GitHub, we would need to make several API calls in order to get the project assigned etc. So, we will start by adding the GroovyHttpClient code to our Incoming script on the GitHub side:

GitHub Incoming

```
import org.apache.commons.lang3.StringEscapeUtils
class GroovyHttpClient {
    // SCALA HELPERS
    public static <T> T await(scala.concurrent.Future<T> f) { scala.concurrent.Await$.MODULE$.result(f,
scala.concurrent.duration.Duration$.MODULE$.Inf()) }
    private static <T> T orNull(scala.Option<T> opt) { opt.isDefined() ? opt.get() : null }
    private static <T> scala.Option<T> none() { scala.Option$.MODULE$.<T>empty() }
    @SuppressWarnings("GroovyUnusedDeclaration")
    private static <T> scala.Option<T> none(Class<T> evidence) { scala.Option$.MODULE$.<T>empty() }
    private static <L, R> scala.Tuple2<L, R> pair(L l, R r) { scala.Tuple2$.MODULE$.<L, R>apply(l, r) }
    // SERVICES AND EXALATE API
    //private httpClient
    private nodeHelper
    def parseQueryString = { String string ->
        string.split('&').collectEntries{ param ->
            param.split('=', 2).collect{ URLDecoder.decode(it, 'UTF-8') }
        }
```

```
    }
    //Usage examples: https://gist.github.com/treyturner/4c0f609677cbab7cef9f
    def parseUri
    {
        parseUri = { String uri ->
            def parsedUri
            try {
                parsedUri = new URI(uri)
                if (parsedUri.scheme == 'mailto') {
                    def schemeSpecificPartList = parsedUri.schemeSpecificPart.split('\\?', 2)
                    def tempMailMap = parseQueryString(schemeSpecificPartList[1])
                    parsedUri.metaClass.mailMap = [
                            recipient: schemeSpecificPartList[0],
                            cc       : tempMailMap.find { it.key.toLowerCase() == 'cc' }.value,
                            bcc      : tempMailMap.find { it.key.toLowerCase() == 'bcc' }.value,
                            subject  : tempMailMap.find { it.key.toLowerCase() == 'subject' }.value,
                            body     : tempMailMap.find { it.key.toLowerCase() == 'body' }.value
                    ]
                }
                if (parsedUri.fragment?.contains('?')) { // handle both fragment and query string
                    parsedUri.metaClass.rawQuery = parsedUri.rawFragment.split('\\?')[1]
                    parsedUri.metaClass.query = parsedUri.fragment.split('\\?')[1]
                    parsedUri.metaClass.rawFragment = parsedUri.rawFragment.split('\\?')[0]
                    parsedUri.metaClass.fragment = parsedUri.fragment.split('\\?')[0]
                }
                if (parsedUri.rawQuery) {
                    parsedUri.metaClass.queryMap = parseQueryString(parsedUri.rawQuery)
                } else {
                    parsedUri.metaClass.queryMap = null
                }
                if (parsedUri.queryMap) {
                    parsedUri.queryMap.keySet().each { key ->
                        def value = parsedUri.queryMap[key]
                        if (value.startsWith('http') || value.startsWith('/')) {
                            parsedUri.queryMap[key] = parseUri(value)
                        }
                    }
                }
            } catch (e) {
                throw new com.exalate.api.exception.IssueTrackerException("Parsing of URI failed:
$uri\n$e", e)
            }
            parsedUri
        }
    }
    GroovyHttpClient(nodeHelper) {
        this.nodeHelper = nodeHelper
    }
    String http(String method, String url, String body, java.util.Map<String, List<String>> headers) {
        http(method, url, body, headers) { Response response ->
            if (response.code >= 300) {
                throw new com.exalate.api.exception.IssueTrackerException("Failed to perform the request
$method $url (status ${response.code}), and body was: \n```$body```\nPlease contact Exalate Support: ".
toString() + response.body)
            }
            response.body as String
        }
    }
    public <R> R http(String method, String _url, String body, java.util.Map<String, List<String>>
headers, Closure<R> transformResponseFn) {
        def unsanitizedUrl = _url
        def parsedUri = parseUri(unsanitizedUrl)
        def embeddedQueryParams = parsedUri.queryMap
        def allQueryParams = embeddedQueryParams instanceof java.util.Map ?
                ({
                    def m = [:] as java.util.Map<String, List<String>>;
                    m.putAll(embeddedQueryParams.collectEntries { k, v -> [k, [v]] } as java.util.
Map<String, List<String>>)
                    m
                })()
                : ([:] as java.util.Map<String, List<String>>)
```

```
        def urlWithoutQueryParams = { String url ->
            URI uri = new URI(url)
            new URI(uri.getScheme(),
                    uri.getUserInfo(), uri.getHost(), uri.getPort(),
                    uri.getPath(),
                    null, // Ignore the query part of the input url
                    uri.getFragment()).toString()
        }
        def sanitizedUrl = urlWithoutQueryParams(unsanitizedUrl)
        def response
        try {
            def request = nodeHelper.githubClient.ws
            //.ws()
                    .url(sanitizedUrl)
                    .withMethod(method)
            if (headers != null && !headers?.isEmpty()) {
                def scalaHeaders = scala.collection.JavaConversions.asScalaBuffer(
                        headers?.entrySet().inject([] as List<scala.Tuple2<String, String>>) {
List<scala.Tuple2<String, String>> result, kv ->
                            kv.value.each { v -> result.add(pair(kv.key, v) as scala.Tuple2<String,
String>) }
                            result
                        }
                )
                request = request.withHeaders(scalaHeaders)
            }
            if (body != null) {
                def writable = play.api.libs.ws.DefaultBodyWritables$.MODULE$.writeableOf_String
                request = request.withBody(body, writable)
            }
            if (!allQueryParams?.isEmpty()) {
                def scalaQueryParams = scala.collection.JavaConversions.asScalaBuffer(allQueryParams?.
entrySet().inject([] as List<scala.Tuple2<String, String>>) { List<scala.Tuple2<String, String>> result,
kv ->
                    kv.value.each { v -> result.add(pair(kv.key, v) as scala.Tuple2<String, String>) }
                    result
                })
                request = request.withQueryString(scalaQueryParams)
            }
         //  throw new Exception("REQUEST ${request.method()}, ${request.url()}, ${request.headers()},
${request.body()}")
            response = await(
                    request.execute()
            )
        } catch (Exception e) {
            throw new com.exalate.api.exception.IssueTrackerException("Unable to perform the request
$method $_url with body: \n```$body```\n, please contact Exalate Support: ".toString() + e.message, e)
        }
        java.util.Map<String, List<String>> javaMap = [:]
        for (scala.Tuple2<String, scala.collection.Seq<String>> headerTuple : scala.collection.
JavaConverters.bufferAsJavaListConverter(response.allHeaders().toBuffer()).asJava()) {
            def javaList = []
            javaList.addAll(scala.collection.JavaConverters.bufferAsJavaListConverter(headerTuple._2().
toBuffer()).asJava())
            javaMap[headerTuple._1()] = javaList
        }
        def javaResponse = new Response(response.body(), new Integer(response.status()), javaMap)
        return transformResponseFn(javaResponse)
    }
    public static class Response {
        final String body
        final Integer code
        final java.util.Map<String, List<String>> headers
        Response(String body, Integer code, java.util.Map<String, List<String>> headers) {
            this.body = body
            this.code = code
            this.headers = headers
        }
    }
}
```

3. Now we can make the relevant API calls to achieve our use case. The first call (following code) will extract the contentId of the GitHub issue in order to use it in the subsequent GraphQL calls:

Retrieve content_id of GitHub issue

```
    def res = new GroovyHttpClient(nodeHelper)
        .http(
            "GET",
            "https://api.GitHub.com/repos/majid-org/demo-repo/issues/${issue.key}",
            null,
            ["Content-type" : ["application/json"], "Authorization":["Bearer ${tok}".toString()]]
    )
    {
        response ->
        if (response.code >= 400)
            throw new com.exalate.api.exception.IssueTrackerException("Failed")
        else
            response.body as String
    }
    def json = js.parseText(res)
    def contentId = json.node_id
```

4. Next we run a GraphQL call to retrieve the projectId of the correct GitHub project. This will be done be retrieving a list of GitHub projects and then matching the names to see if there is a match for the project selected by the user from the Jira custom field:

Find project_id of correct GitHub project

```
    res = new GroovyHttpClient(nodeHelper)
        .http(
            "POST",
            "https://api.github.com/graphql",
            "{\"query\":\" {organization(login:\\\"majid-org\\\") {projectsV2(first: 20) {nodes {id
title}}}}\"}",
            ["Content-type" : ["application/json"], "Authorization":["Bearer ${tok}".toString()]]
    )
    {
        response ->
        if (response.code >= 400)
            throw new com.exalate.api.exception.IssueTrackerException("Failed")
        else
            response.body as String
    }
    json = js.parseText(res)
    json.data.organization.projectsV2.nodes.each {
    if (it?.title == "@${projectJira}")
        projectId = it.id
    }
```

5. We are now ready to add the GitHub issue to the correct project by utilizing the projectId and contentId (retrieved in the previous 2 steps). The API call also returns the itemId that corresponds to the issue's id within the project (and will be needed in sunsequent calls):

Add GitHub issue to project

```
    res = new GroovyHttpClient(nodeHelper)
        .http(
            "POST",
            "https://api.github.com/graphql",
            "{\"query\":\"mutation {addProjectV2ItemById(input: {projectId: \\\"${projectId}\\\"
contentId: \\\"${contentId}\\\"}){item {id}}}\"}",
            ["Content-type" : ["application/json"], "Authorization":["Bearer ${tok}".toString()]]
    )
    {
        response ->
        if (response.code >= 400)
            throw new com.exalate.api.exception.IssueTrackerException("Failed")
        else
            response.body as String
    }
    json = js.parseText(res)
    itemId = json.data.addProjectV2ItemById?.item?.id
```

6. To deal with the issue status within the GitHub project/board, we need to work with the GraphQL calls again, and in order to do so we now need the fieldId of the status field itself and also the id of the status that we are trying to change to in the project:

Retrieve the fieldId of Status field and id of the status

```
    res = new GroovyHttpClient(nodeHelper)
        .http(
            "POST",
            "https://api.github.com/graphql",
            "{\"query\":\" { node(id: \\\"${projectId}\\\") { ... on ProjectV2 { fields(first: 20) {
nodes { ... on ProjectV2Field { id name } ... on ProjectV2IterationField { id name configuration {
iterations { startDate id }}} ... on ProjectV2SingleSelectField { id name options { id name }}}}}}\"}",
            ["Content-type" : ["application/json"], "Authorization":["Bearer ${tok}".toString()]]
    )
    {
        response ->
        if (response.code >= 400)
            throw new com.exalate.api.exception.IssueTrackerException("Failed")
        else
            response.body as String
    }
    json = js.parseText(res)


    json.data.node?.fields?.nodes.each {
        if (it.name == "Status"){
            fieldId = it.id
            it.options.each{
               statusEntry ->
               if (statusEntry.name == statusJira)
                           statusId = statusEntry.id
            }
        }
    }
```

7. The final step is to just change the ticket status:

Assign Project status to GitHub issue

```
    res = new GroovyHttpClient(nodeHelper)
        .http(
            "POST",
            "https://api.github.com/graphql",
            "{\"query\":\"mutation {updateProjectV2ItemFieldValue( input: { projectId: \\\"${projectId}
\\\" itemId: \\\"${itemId}\\\" fieldId: \\\"${fieldId}\\\" value: { singleSelectOptionId: \\\"${statusId}
\\\" }}) { projectV2Item { id }}}\"}",
            ["Content-type" : ["application/json"], "Authorization":["Bearer ${tok}".toString()]]
    )
    {
        response ->
        if (response.code >= 400)
            throw new com.exalate.api.exception.IssueTrackerException("Failed")
        else
            response.body as String
    }
```

A video demonstration of the use case:

And you can download the entire code for the GitHub Incoming side here.