REST APIs - design patterns

> This page is used to collect and state general definitions to be applied in all REST APIs for AUTOMIC products!

Background Information

In the v12 Backlog there were topics around REST APIs driven from multiple domains

- ARA general REST API
- AE general REST API (postponed to v12.1)
- ARA UI development REST API

In the current products we have the Service API (S-API) as an REST API which is part of the ASO offering. For this API some research was already done and General Patterns were defined. There are also some learnings around it which should be considered in discussing the design aspects.

Design goals

- Optimize for development use
  - In case of doubt, development usability goes over standard conformance
- Simple and performant authentication
  - No need to remember how to do a complex OAuth handshake if you just want to try things out
- Reuse existing framework/solutions where applicable
- Avoid effort doubling/multiple locations for specification/documentation/test case definition
- Usable for building rich frontends on top of the API
  - We have to e.g. avoid to fall into an n+1 trap for displaying tables with rich information
- API should expose the original object model rather than adding a layer on top (like SAPI did).
- REST APIs should not necessarily replace APIs that are currently provided

Design aspects

	Automic API Standard Ideas

General Structure

What's the endpoints for the api.

Examples:

`https://srv001/ara/api/deployment/v2/myresource`

`https://ecc.mycompany.com/api/callbacks/v1/myresource.json`

General Form

`https://{hostname}/{application}/api/{api_name}/{version}/{resource}.{format}`

segment	Description.
"https://"	HTTPS access to APIs only. Secure by default and not otherwise.
hostname	The hostname of the api application.
application	The web application hosting the api.
"api"	Fixed name after the root path of the web application in order to avoid conflicts with application routes.
api_name	An application may provide special purpose APIs to simplify development with the API.
version	The version of the API. It should only be incremented if there are breaking changes. Adding a new reso adding additional additional URL parameters are normally not considered as breaking changes.
resource	The resource to make the request on.
format	Optional. If the API supports multiple serialization formats of the resource (e.g. json, xml, protocol buffe required). At first only JSON should be supported if other serialization formats cause additional effort.

HTTP Verbs

How to correctly use HTTP verbs in the API.

Use the following HTTP verbs on resources:

GET	Only get a resource. Never use GET for changing anything!

POST	Use for adding new resources or changing them:
	
	Example:
	
	`POST /users`
	
	Use for calling action url's if they can't be avoided
	
	Example:
	
	`POST /tasks/981d62/deactivate`
	
	POST operation in general has an additive behavior. If you POST a resource containing a collection, the entries for the resource before the request and the entries contained in the request.
	
	Example:
	
	current state of package with id 123
	
	`GET /packages/123`
	
	`{ "name": "packagefoo", "components": { [ {"name": "comp1" } ] } }`

`POST /packages/123`

```
{
"name": "packagefoo",
"components": { [ {"name": "comp2" } ] }
}
```

`GET /packages/123`

```
{
"name": "packagefoo",
"components": { [ {"name": "comp1"}, {"name": "comp2"} ]
}
```

results in a package with components comp1 and comp2 assigned.

"components":[{"id":"foo", "value":"bar"}] ... add to or update object in the collection

"components":[{"id":foo", "value2":"bar2"}] ... add value2 to the object with id=foo

"components":[] ... do not add any new entry to the collection

"components": null ... deletion

Deletion of value2 only is not possible directly. The only options to provide such functionality in the API a

- to completely wipe the components container object via setting it null (⚠ setting value2 to null, mus
- or represent it as subresource and implement DELETE on it.

There might be object attributes where additive behavior doesn't apply (e.g ENUM). In this case docume

DELETE	Use for deleting resources. Example: `DELETE /components/49851` Response on successful deletion returns status code 200 If the resource doesn't exist return status code 404 Background Information: There is plenty of discussion around response behavior to be found on the internet. These are our views - 200 allows for empty bodies. We strive to keep the variety of response codes low, that's why no e.g. - idempotency only talks about the server side state and "sideeffects". There is no statement that the each time - Returning 200 and 404 depending on the server state, rather than always 200, avoids race conditior want to provide feedback to the consumer what happened on server side. - We accept the potential security risk, as it means you need to have DELETE permissions and no GE

We do not use PATCH, PUT, or HEAD

There has been a lot of discussion around this topic. Overall, for now the conclusion is that it is very hard to get the se and POST right, so we decided to restrict the set of verbs. Originally we suggested to just use PUT, but due to the fac non-idempotent operations we reverted this decision to just use POST.

One reason to be that restrictive is, that if you decide to support PUT and POST, you also immediately require PATCH

The topic is to be revisited with work on the AWA REST API

HTTP status codes

How much do we leverage HTTP and what goes into the payload to communicate error situations back to the consumer.

Use the following HTTP status codes:

Code	Meaning
200	The request was successful
201	A new resource was created. The resource should be serialized in the response in its default representatio
400	Bad Request. The request could not be serialized by the server, or the data entered by the user is invalid. I business rule violations --> known error situations. In all cases, it will not help if the client just resends the r
401	Unauthorized access. The user has to sign in again. This gets returned if e.g. the token is not valid anymor
403	Forbidden. The user is not allowed to do what she planned to do. Eg. ACL check fails.
404	The requested resource was not found.
500	Some kind of unexpected server error occured. The message should try to give more details about the erro request.

For all error codes (4xx, 5xx) return an error object.

Error Object:

Example:

```
{
  "code": "A4123",
  "error": "invalid field name: message"
  "details": ""
 }
```

Field	Description
code	An application specific error code.
error	The localized error message (see localization) as it should be displayed to a user.
details	An optional details field that should not be shown to users in the default case, but can be helpful for debugg

Client implementation should print error objects they receive from APIs to the logfile. 4x errors are an indication, that t input before the request goes to the API.

Localization

Users should be able to specify the language in which they want to receive results.

Users can specify the optional Accept-Language header to identify in which language they want to receive API results should be English. If the API does not have a localization for this language, results should be in English. Accepted va

Example:

```
Accept-Language: de
```

Authentication

The OAuth authentication setup implemented for SAPI turned out to be very complex to setup and makes it hard to easily start to use the API.

There should be the possibility to run the API in a setup where you can access each resource via CLI call (e.g. curl with just Basic Authentication)

For the sake of simplictiy the authentication mechanisms to be supported are

- BASIC authentication
  - username format: <client>/<username>/<department>
- API key

For a comparison of different authentication approaches see REST API Authentication - design discussion compiled

Linking / API Navigation

SAPI is using HAL to allow easy API navigation. We should challenge this approach and reconsider if the HAL format is the right way to go.

how do we link to other resources? usage of href, rel, ...

We do not recommend linking or URL references in the API. Instead our users have to rely on a good plain old docun please use the HAL notation (http://stateless.co/hal_specification.html) as it is not intrusive.

Currently there are many competing conventions and standards. This means that an API consumer anyway has to do given that we are the primary consumer of our own APIs the expected benefits are less than the expected costs and

Resource Serialization

Every resource should have a default serialization, that includes the most important attributes of a given resource. Ad up to the API provider to decide which attributes should be included in the default serialization. Keep in mind that ban for mobile consumers).

Singular Resources

e.g. /users/5152, /objects/MY.OBJECT, /applications/584512

Dates are always returned in UTC.

Properties are returned in a snake_case format.

Properties must not start with an underscore (reserved for future use)

The "id" must always be included in an objects default representation. It's has to be a string.

Example:

```
GET /objects/MY.JOB
```

```
{
"id": "MY.JOB",
"type": "JOBP",
"title": "My Great Job",
"status": "unlocked",
"locked_by": null,
"created": "2015-12-09T15:22:00",
"version": 2
}
```

Complex properties should be embedded as regular JSON objects (can be nested as needed)

Example:

`GET /objects/MY.JOBP/tasks/12`

```
{
...
"post_conditions": [
 { "condition": "exit_code = 0", "action": {"setVariable": {"nar
} }
]
}
```

A linked object should be included in a default representation so that it can be used for e.g. displaying a user name w

Example (in this case our default representation includes an embedded locked_by user):

`GET /objects/MY.JOB`

```
{
"name": "MY.JOB",
"type": "JOBS",
"title": "My Great Job",
"status": "locked",
"locked_by": {"id": "SBB01/ECB", "type": "User", "displayname":
}
```

A resource representation may include multiple other resources. Linked resources should be embedded as standard limit, sorting, and offset information is given in the resource URL and not visible in the representation anymore. The A entries in the collection.

Example (components as part of the default representation of an application):

`GET /applications/51652`

```
{
"id": "51652"
"name": "My Application"
"components": {
 "hasmore": false,
 "data": [
  {"id": 56152, "name": "ARA Core", "type": "IIS"},
  {"id": 26154, "name": "ARA Database", "type": "DB"},
  {"id": 84652, "name": "ECC", "type": "Tomcat"}
 ]
}
```

Key discussion points and why we came to this representation:

- We do not include referenced resources in a container (e.g. _embedded) because it looks crappy and we do not and linked resources have to be avoided by the API author (an attribute must not have the same name as a linke
- We do not hypermedia patterns in the api (e.g. include a URL field) because:
  - we do not have use of this for our own consumers
  - we do not see an established approach/standard in public apis (many competing things such as hal, jsonapi
  - it can be difficult to serialize because you have to know the public urls that your objects get exposed to
- We decided for a "default representation" pattern where API authors decide on a default representation of a reso
  - we are the primary user of our own api and in many cases it just makes sense to include certain data (e.g. u

Collection Resources

If a resource is a collection of entities and not a single entitiy, it is wrapped in a data container and includes the total r

Example:

`GET /applications/98454/components`

```
{
"total": 3,
"hasmore": false,
"data": [
 {"id": 56152, "name": "ARA Core", "type": "IIS"},
 {"id": 26154, "name": "ARA Database", "type": "DB"},
 {"id": 84652, "name": "ECC", "type": "Tomcat"}
]
}
```

Param	Description
total	The total number of resources in the collection. The returned data can be a subset because e.g. only 10(
hasmore	"true" if the response does not contain all data in the collection.
data	An array of the objects. If only one object is contained in the collection, it must still be returned as an arra

POST operation on collections - unique resource attribute

If you POST a request on a collection resource with a unique resource attribute, the behavior is the same as you wou

Example object JOBS.WIN.1 with Idnr 1234

`POST /objects`

```
{
"name":"JOBS.WIN.1",
...
}
```

equals

`POST /objects/1234`

Field selection

An API **MAY** support field selection which allows you to specify the information that gets selected.

Example:

Limit the response to only the fields that you're interested in.

`GET /objects/THE.OBJECT.NAME?fields=name,title`

Example (simple user manager)

`GET /users/hch`

```
{
"id": "HCH",
"displayname": "Christopher Hejl"
}
```

`GET /users/hch/groups`

```
{
"total": 2,
"hasmore": false,
"data": [
 {"id": "grp1", "name": "Storytellers" },
 {"id": "grp2", "name": "Engineering Leadership"}
]
}
```

`GET /users/hch?fields=id,groups`

```
{
"id": "HCH",
"groups": {
 "hasmore": false,
 "data": [
  {"id": "grp1", "name": "Storytellers" },
  {"id": "grp2", "name": "Engineering Leadership"}
 ]
 }
}
```

`GET /users/hch?fields=id,manager,first_name,last_name,groups.id,groups.displayname,groups.`

```
{
"id": "HCH",
"manager": {
 "id": "PUJ"
},
"first_name": "Christopher",
"last_name": "Hejl",
"groups": {
 "hasmore": false,
 "data": [
  {
   "id": "grp1",
   "displayName": "Storytellers",
   "description": "The best storytellers in the company.",
   "users": {
    "hasmore": false,
    "data": [
     {"displayName": "Christopher Hejl"},
     {"displayName": "Benedikt Eckhard"}
    ]
  },
  {
   "id": "grp2",
   "displayName": "Engineering Leadership",
   "description": "Reporting to Josef Puchinger",
   "users": {
    "hasmore": false,
    "data": [
     {"displayName": "Christopher Hejl"},
     {"displayName": "Benedikt Eckhard"}
    ]
  }
  ]
 }
}
```

Param	Description
fields	A comma seperated list of fields (the user specifies the technical field name) that should be selected from • If a given field does not exist it will be silently ignored. • If a field is a resource reference, the API should return a default representation of the referenced resc • The API must only return fields explicitly selected by users • The API MAY support cascading field selection • The API can decide which fields can be expanded/selected and for how many levels

Remark:

We do not use "embed", "include", ... because it does not add value to just specifying attributes and linked resources

Resource expansion

Inline expansion of included resources in a response

There is no difference between resource expansion and field selection. You may include linked resources in your rep
you do that, you have to explicitly specify all other fields that you want to include in the representation.

Resource Levels

Consider to restrict the maximal number of resource levels and rather introduce new top level resources than nesting to many of them in one path

Instead of building up deep navigation graphs, flatten out resource trees where possible. A resource URL in REST is
Hierarchical nesting should therefore only be used if a resource does not have its own identity.
Hint: Mind that from a API consumer side, it also might make perfect sense to support both. The nested resource can
the parent. The dedicated endpoint can be used to query more than one of the subentities in one call and process the

Example (components can be identified by ID):

Instead of:

```
GET /applications/12312/components/12321/
```

Introduce a new top level resource:

```
GET /applications/12312/components
```

```
GET /components/12321
```

Example (tasks in a jobplan do not have their own identity --> nesting is perfectly fine):

```
GET /objects/MY.WORKFLOW/tasks/3/postconditions
```

Documentation & Design

Documentation should be close and part of the API itself. Avoid additional PDF artifacts describing the API. Documentation, Specification, Test of the API very close to each other or even defined at one point and rendered for different purposes.

(Updated 18.12.2017)

We are using **Swagger (OAS 2.0)** for documentation of our REST APIs. OAS 2.0 is used as at the time of writing OA
yet.

We switched away from RAML, as we ran out of sync between docu and implementation, had troubles in the buildcha
support for client generation, as well as CA uses swagger as the default format.

The approach is have **annotations in the source code** which are interpreted during build time to compile the actual

Important to consider

- For requests its important, to not only rely on the generated default examples. This is necessary as per default th
makes the documentation bloated and confusing to the reader.
- Pay attention to the naming of model elements. Per default they use a name generated from the implementing cl
edge case explicitly override the name of the class to define the model element.

How to use it

Java

- Swagger https://github.com/swagger-api/swagger-core/tree/v1.5.17 and have a look at the wiki there
https://github.com/swagger-api/swagger-core/wiki/Annotations-1.5.X
- swagger-maven-plugin: https://bitbucket.automic.com/stash/projects/TOOL/repos/swagger-maven-plugin/browse
  - forked because there is no 3.1.6 release available yet. Not required as soon as 3.1.6 is released.
  - output is generated as part of compile phase
- sample pom used in automation engine rest api -
https://bitbucket.automic.com/stash/projects/AUT/repos/automation-engine-java/browse/rest/pom.template.xml
- to see an example in our codebase have a look here
https://bitbucket.automic.com/stash/projects/AUT/repos/automation-engine-java/browse/rest/src/com/automic/res
- swagger-ui rendering the content of swagger.json https://bitbucket.automic.com/stash/projects/DOC/repos/swag
  - (only relevant if you want to ship the docu application with your LE. If you do so, make sure you can disable
  - forked due to the fact that we cant support the try out button on docs.automic.com

For .NET

- swashbuckle is used by ARA teams https://github.com/domaindrivendev/Swashbuckle

Please align with AS team (depman@automic.com) how to properly handover your swagger.json for inclusion in the d

Command resources
 	
In scenarios where it is not possible to model behavior via the defined HTTP verbs only the question arises how to expose some more complex kind of domain command in the API?
	1. subresource - consider if you can model it as a common subresource
	
	For example have a look at issue transitions in JIRA https://docs.atlassian.com/jira/REST/cloud/#api/2/issue-doT
	
	`POST /resource/{id}/subresource`
	
	You could expose some command subresource like described in 2), but it makes way more sense to have a sub
	
	It allows you to also perform GET operations on the subentity which might give you more information about the e
	
	`GET /resource/{id}/subresource`
	
	2. expose it as command subresource
	
	`POST /resource/{id}/command`
	
	We already applied this pattern for example in the SAPI - /services/{serviceName}/consume

Performance For topics like throttling or caching we should validate if there is any kind of middleware/framework available, to avoid a custom implementation.	TBD
Push Notifications Have a common approach to enable push notifications/callbacks for async response scenarios - how to solve (e.g. DDP protocol)	TBD
API scope Aligned scope of API - around which domain/topic/component do we scope the functionality of an API? 1 API with too many resources vs. few APIs with well defined scope	TBD
Asynchronous Requests How to issue and deal with asynchronous requests	TBD Common approach: Use status "202 Accepted" and create a queue to poll for status of request: https://www.adayinthelifeof.nl/2011/06/02

Pagination

How to limit the entries of a response containing a collection and communicate overall size of the collection and iteratively retrieve as many entries as required.

request

- requests that result in a result set of records have to support the optional query parameters "max_results" and "s returned at once and the offset
  - `GET /resource?max_results=100&start_at=200`
- in case the optional parameters are omitted the recommended behavior is to respond without an offset and "max use case (e.g. entries required for a screen at typical resolution plus some extra to avoid another immediate requ

response

- this is the structure a response containing a page of result records looks like.
- The "total" parameter is optional, as it might not always make sense/be expensive to determine.

```
{
 "hasmore": false,
 "total": 2,
 "data": [
  { ... },
  { ... }
 ]
}
```

TBD

- pagination of subresources included in the response e.g. /ara/api/data/v1/applications?max_results=n implicatior

Approaches looked at:

- Stripe API - https://stripe.com/docs/api/curl#pagination http://docs.stormpath.com/rest/product-guide/#pagination
- Stormpath - http://docs.stormpath.com/rest/product-guide/#pagination
- JIRA - https://docs.atlassian.com/jira/REST/latest/#pagination

Sorting

How to sort elements in a collection

Ordering can be ascending or descending. By default it's ascending. To specify the ordering use "-" or "+" sign. Defin Fields supporting sorting must mention that in the documentation.

Examples:

`?sort=name Order by "name" ascending`

`?sort=+name Order by "name" ascending`

`?sort=-name Order by "name" descending`

Suggestion (Brauchler, Christian):

Good approach for sorting is described in http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api:

Also see Atlassian JIRA https://docs.atlassian.com/jira/REST/latest/

Filtering
 	
Filter resource lists
	For now we only define the behavior for opertor "is equal"
	
	If operation is "equal", use a simple parameter like:
	
	`GET /packages?name=1.0.2`
	
	Wildcard support for filter criteria is optional and denoted by '*'.
	
	Field criteria may support '*' wildcard like classical 'like' queries eg. name like 'PCK_AUTOMIC.*'
	
	The '*' may also be used as 'all' operator e.g.
	
	`GET /approvals?recipient=*`
	
	to retrieve all approvals for all recipients
	
	Date Fields
	
	Date fields may support a special keyword called 'LATEST'. This returns a result with the latest entry corresponding t
	
	`GET /packages?application.name=foo&creation_time=LATEST`
	
	Background information: in our domain we see a lot of use cases for requests querying the most recent record of a re /packages?application.name=foo&sort=-creation_time&max_results=1 we decided to introduce the keyword LATEST e.g /packages/latest but the semantic is ambiguous in many cases (e.g. latest means creation time, execution start, e
	
	TBD for other queries than "is equal",
	
	Please get in touch with Eckhard, Benedikt and Leitich, Stefan if you require queries other than is equal.
	
	Suggestion (Brauchler, Christian):
	
	If operation is "equal", use a simple parameter like: GET /packages?name=1.0.2
	
	If other operations including wildcards are to be used, use parameters like this: GET /packages?name=like:Ara*. Pos (greater than), geq (greater than or equal), lt (lower than, leq (lower than or equal), like, unlike. Wildcards *
	
	Note: This approach is used for Sparkpay REST API: See https://support.sparkpay.com/hc/en-us/articles/202836800

Time, Date and duration information	All time and date information in request and responses are solely represented in UTC. Date and Time format is represented in the format specifiec by ISO 8601 (See https://en.wikipedia.org/wiki/ISO_8601 Durations are represented in seconds (this deviates from ISO 8601, as the ISO format only works in combination with **TBD other timezone information than UTC** If there is the need to request or repond in a certain timezone this needs to be discussed before being implemented!

Status endpoint
 	
Endpoint that can be used to verify that the service is up
	In case of multiple instances offering a REST endpoint, Proxies, API gateways make use of an endpoint to check if th exposed as
	
	`GET https://{hostname}/{application}/api/{api_name}/{version}/ping`
	
	e.g. for ae REST API (note: api_name is optional (not in use for AE REST API) and no client information in the url)
	
	`GET https://localhost/ae/api/v1/ping`
	
	and return HTTP response code 200 **without any payload** and are available **without authentication**.
	
	For security reasons its important that no information is returned as it is an unauthenticated endpoint.

Link to documentation of public REST APIs:

- https://docs.atlassian.com/jira/REST/latest
- https://stripe.com/docs/api
- Article with links to paypal REST API design guide - https://www.infoq.com/news/2017/09/paypal-api-guide

How is the competition doing it:

- BMC Control M
  - https://docs.bmc.com/docs/display/public/workloadautomation/Control-M+Automation+API+-+Services
  - https://ec2-54-191-85-182.us-west-2.compute.amazonaws.com:8443/automation-api/swagger-ui.html

How are our fellow CA colleagues work on this topic

- Mainframe BU - https://cawiki.ca.com/display/MFDEVWiki/Building+Code+Subject+page+for+RESTFul+APIs