Table of Contents

How It Works 1
Available Services 1
Administration 1
Storage Access (CRUD, Metadata) 3
Call Actions 9
Examples 10
Integration 10
JUnit Tests 12

We define the business logic with life cycle objects on the server side. The access authorization of an application is checked by a security manager. The business logic is usually available via master- or subconnections from the client.

In order to complete the technology independence, the complete business logic of an application is also available via REST.

For the use of the REST services, the authentication with username and password is necessary. BASIC is used as the authentication mechanism. The credentials are checked by the security manager of the application as usual. You do not need to change a source code line to integrate the REST services.

How It Works

The REST implementation in JVx has been implemented with Restlet. To use the REST services, the deployment descriptor must be configured as follows:

```
<servlet>
    <servlet-name>RestletServlet</servlet-name>
    <servlet-class>org.restlet.ext.servlet.ServerServlet</servlet-class>

    <init-param>
      <param-name>org.restlet.application</param-name>
      <param-value>com.sibvisions.rad.server.http.rest.RESTAdapter</param-
value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>RestletServlet</servlet-name>
    <url-pattern>/services/rest/*</url-pattern>
  </servlet-mapping>
```

With this configuration, the following services are available:

Available Services

Administration

Various services are available for the administration. These can only be used by POST requests in the standard case. However, if certain custom services have been registered, they can also be addressed via GET request.

The following services are available by default:

- Test authentication
- Change password
- Check database

Test Authentication

Test URL:
`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/testAuthentication`
or
`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/testAuthentication/parameter`

The request requires a HashMap in JSON format.

Example:

```
{ "username" : "admin",
  "password" : "adminpassword"
}
```

The username can also be omitted. In this case, the parameter from the URL will be used as the username.

POST-Response

If the login was successful, no response is generated and the status code is 204 (SUCCESS_NO_CONTENT).

Change Password

Test URL:
`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/changePassword`
or
`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/changePassword/parameter`

The request requires a HashMap in JSON format.

Example:

```
{ "username" : "admin",
  "oldpassword" : "oldpassword",
  "newpassword" : "newpassword"
}
```

The username can also be omitted. In this case, the parameter from the URL will be used as the username.

POST-Response

If the password has been changed, no response is generated and the status code is 204 (SUCCESS_NO_CONTENT).

Check database

Test URL:
`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/checkDB`

GET-Response

If the check was successful, no response is generated and the status code is 204 (SUCCESS_NO_CONTENT). If database is not available, status code 500 (SERVER_ERROR_INTERNAL) will be returned. It's also possible that the configuration can't be found. In this case, status code 503 (SERVER_ERROR_SERVICE_UNAVAILABLE) will be returned.

Custom Services

If you want to register your own service at runtime, this can be done by

```
UserService.register(String pApplicationName, String pAction,
ICustomServiceDelegate pDelegate);
UserService.unregister(String pApplicationName, String pAction)
```

The service can be addressed either via GET or POST request, depending on whether ***ICustomServiceGetDelegate*** or ***ICustomServicePostDelegate*** is used.

Test URL:

`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/ACTION`
or
`http://server:port/webapp/services/rest/APPLICATION_NAME/_admin/ACTION/parameter`

Storage Access (CRUD, Metadata)

- Select
- Insert
- Update
- Delete
- Metadata

GET-Request (Select)

Query all data:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME`

Query exactly one record:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME/PRIMARY_KEY`

If the PK is composed of several columns, the query parameters must be used:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME?PKCOLUMN=VALUE&PKCOLUMN2=VALUE2`

The query parameters can also be used to perform filtering with columns that are not PK columns.

Read more about complex query parameters.

GET-Response

The response always contains a list of HashMaps in JSON format. The column name is used as the key.

Example:

```
[ { "ID" : 0,
    "POST_ID" : 127,
    "POST_PLZ" : "1127",
    "STIEGE" : 8,
    "STRA_ID" : 68,
    "STRA_NAME" : "Strasse (69)",
    "HAUSNUMMER" : 37,
    "TUERNUMMER" : 79
  },
  { "ID" : 1,
    "POST_ID" : 50,
    "POST_PLZ" : "1050",
    "STIEGE" : 7,
    "STRA_ID" : 55,
    "STRA_NAME" : "Strasse (56)",
    "HAUSNUMMER" : 37,
    "TUERNUMMER" : 60
  },
]
```

POST-Request (Insert)

Insert a new record:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data`

/**STORAGE_NAME**

The request requires a HashMap in JSON format. The column name is used as the key.

Example:

```
{ "POST_ID" : "0",
  "STRA_ID" : "0",
  "HAUSNUMMER" : "9999"
}
```

POST-Response

The response returns the complete record in JSON format:

```
{ "ID" : 1008,
  "POST_ID" : 0,
  "POST_PLZ" : "1000",
  "STIEGE" : null,
  "STRA_ID" : 0,
  "STRA_NAME" : "Strasse (1)",
  "HAUSNUMMER" : 9999,
  "TUERNUMMER" : null
}
```

PUT-Request (Update)

Update a record with Primary Key:
`http://server:port/webapp/services/rest/`**APPLICATION_NAME**`/`**LIFECYCLE_CLASS**`/data/`**STORAGE_NAME**`/`**PRIMARY_KEY**

If the PK is composed of several columns, or if the records are not to be identified via the PK, the query parameters must be used:
`http://server:port/webapp/services/rest/`**APPLICATION_NAME**`/`**LIFECYCLE_CLASS**`/data/`**STORAGE_NAME**`?`**COLUMN=VALUE**`&`**COLUMN2=VALUE2**

The request requires a HashMap in JSON format. The column name is used as the key.

Example:

```
{ "ID" : "123",
  "HAUSNUMMER" : "0",
  "STIEGE" : "0",
  "TUERNUMMER" : "0"
}
```

It should be noted that PK columns are not updated.

PUT-Response

The response returns the complete record in JSON format:

```
{ "ID" : 0,
  "POST_ID" : 127,
  "POST_PLZ" : "1127",
  "STIEGE" : "0",
  "STRA_ID" : 68,
  "STRA_NAME" : "Strasse (69)",
  "HAUSNUMMER" : "0",
  "TUERNUMMER" : "0"
}
```

DELETE-Request (Delete)

Delete exactly one record:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`/`**`PRIMARY_KEY`**

If the PK is composed of several columns, the query parameters must be used:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`?`**`PKCOLUMN=VALUE`**`&`**`PKCOLUMN2=VALUE2`**

DELETE-Response

The response returns the number of deleted records in JSON format (as number):

```
42
```

OPTIONS-Request (Metadata)

Request Metadata:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**

OPTIONS-Response

The response returns the metadata in JSON format:

```
{ "autoIncrementColumnNames" : [ "ID" ],
  "columnMetaData" : [ { "allowedValues" : null,
        "autoIncrement" : true,
        "dataType" : 3,
        "defaultValue" : null,
```

```
        "label" : "Id",
        "linkReference" : null,
        "name" : "ID",
        "nullable" : false,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Post Id",
        "linkReference" : null,
        "name" : "POST_ID",
        "nullable" : false,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 12,
        "defaultValue" : null,
        "label" : "Plz",
        "linkReference" : { "columnNames" : [ "POST_ID", "POST_PLZ"],
                            "referencedColumnNames" : [ "ID", "PLZ"],
                            "referencedStorage" :
".subStorages.postleitzahlen"
                          },
        "name" : "POST_PLZ",
        "nullable" : false,
        "precision" : 2147483647,
        "scale" : 0,
        "signed" : false,
        "sqltype" : 12,
        "writable" : false
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Stra Id",
        "linkReference" : null,
        "name" : "STRA_ID",
        "nullable" : false,
        "precision" : 10,
```

```
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 12,
        "defaultValue" : null,
        "label" : "Name",
        "linkReference" : { "columnNames" : [ "STRA_ID", "STRA_NAME"],
                            "referencedColumnNames" : [ "ID", "NAME"],
                            "referencedStorage" : ".subStorages.strassen"
                          },
        "name" : "STRA_NAME",
        "nullable" : false,
        "precision" : 2147483647,
        "scale" : 0,
        "signed" : false,
        "sqltype" : 12,
        "writable" : false
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Hausnummer",
        "linkReference" : null,
        "name" : "HAUSNUMMER",
        "nullable" : false,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Stiege",
        "linkReference" : null,
        "name" : "STIEGE",
        "nullable" : true,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
```

```
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Tuernummer",
        "linkReference" : null,
        "name" : "TUERNUMMER",
        "nullable" : true,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      }
    ],
  "columnNames" : [ "ID",
      "POST_ID",
      "POST_PLZ",
      "STRA_ID",
      "STRA_NAME",
      "HAUSNUMMER",
      "STIEGE",
      "TUERNUMMER"
    ],
  "primaryKeyColumnNames" : [ "ID" ],
  "representationColumnNames" : [ "ID",
      "POST_ID",
      "POST_PLZ",
      "STRA_ID",
      "STRA_NAME",
      "HAUSNUMMER",
      "STIEGE",
      "TUERNUMMER"
    ]
}
```


Call Actions

The server-side actions can be called directly from the life cycle object as well as from available business objects. It's also possible to use parameters.

- Action without parameter
- Action with parameter

GET-Request

Call a server-side action (without parameter):
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/acti`

on/**ACTION_NAME**

Call a method from a business object (without parameter):
`http://server:port/webapp/services/rest/`**APPLICATION_NAME**`/`**LIFECYCLE_CLASS**`/object/`**OBJECT_NAME**`/`**ACTION_NAME**

GET-Response

The response returns the return value of the action in JSON format.

POST/PUT-Request

Call a serer-side action (with parameter):
`http://server:port/webapp/services/rest/`**APPLICATION_NAME**`/`**LIFECYCLE_CLASS**`/action/`**ACTION_NAME**

Call a method from a business object (with parameter):
`http://server:port/webapp/services/rest/`**APPLICATION_NAME**`/`**LIFECYCLE_CLASS**`/object/`**OBJECT_NAME**`/`**ACTION_NAME**

The request requires an array of objects populated with the parameters for the action.

Example:

Action:

```
public String calculate(Number pFirst, Number pSecond)
{
  return "" + pFirst.intValue() + pSecond.intValue();
}
```

JSON Request:

```
[123,1]
```

POST/PUT-Response

The response returns the return value of the action in JSON format.

Examples

Integration

Using php:

```
$ch = curl_init();
curl_setopt($ch,
CURLOPT_URL,'https://<server>/DB/services/rest/League/Standings/data/All');

curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_USERPWD, "user:password");
//curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 5);

$json = json_decode(curl_exec($ch), true);

curl_close($ch);
```

Using Javascript:

rest.html

```
<html>
<head>
<script>
function doRest() {
    const http = new XMLHttpRequest();
    const
url='https://<server>/DB/services/rest/League/Standings/action/getResul
ts';

    http.open("POST", url, true, 'user', 'password');
    http.withCredentials = true;
    http.send("[88]");

    http.onreadystatechange=(e)=>
    {
    if (http.readyState == 4)
    {
        console.log(atob(eval(http.responseText)));
    }
    }
}
</script>
</head>
<body>
<button type="button" onclick="doRest()">REST call</button>
</body>
</html>
```

AngularJS 4 with VisionX and JVx REST services
AngularJS with JVx in action
Oracle JET with VisionX/JVx

JUnit Tests

TestCallService for the Lifecycle objects: Session and Address

Note

For action calls, the correct data types must be used! In general, it is best to dispense with primitive data types, as parameters, and instead of using arrays, the List Interface should be used. It is also recommended to use Number for all numerical values. This avoids problems due to JSON serialization.

The life cycle name should be the fully qualified class name, with package. If only the simple class name is used, JVx will try to find a matching class. If several classes are considered, then no class is used. You can optionally define a search path in the config.xml of the application:

```
<application>
  ..
  <rest>
    <search>
      <path>/com/sibvisions/app/myapp</app>
      <path>/com/sibvisions/app/myapp/screens/sub/</app>
    </search>
  </rest>
</application>
```

Additional information about this feature is available in our Support System.

From:
https://doc.sibvisions.com/ - **Documentation**

Permanent link:
https://doc.sibvisions.com/jvx/common/util/rest

Last update: **2022/11/17 11:13**