Table of Contents

Funktionsweise 1
Verfügbare Dienste 1
Administration 1
Storage Zugriff (CRUD, Meta Data) 3
Aufruf von Actions 9
Anwendungsbeispiele 11
Integration 11
JUnit Tests 12

Wir definieren die Business Logik mit Life-Cycle Objekten am Server. Die Zugriffsberechtigung einer Applikation wird durch einen Security Manager geprüft. Der Zugriff auf die Business Logik erfolgt üblicherweise via Master- oder SubConnections vom Client.

Um die Technologie Unabhängigkeit zu vollenden, steht die Komplette Business Logik einer Applikation auch via REST zur Verfügung.

Für die Benützung der REST Dienste ist die Autentifizierung mit Benutzername und Passwort notwendig. Als Authentifizierungsmechanismus wird BASIC erwendet Die Überprüfung der Daten wird vom Security Manager der Applikation wie gewohnt durchgeführt. Sie müssen für die Integration der REST Dienste keine Zeile Source Code verändern.

Funktionsweise

Die REST Implementierung in JVx wurde mit Restlet umgesetzt. Um die REST Dienste zu nutzen, muss der Deployment Deskriptor wie folgt konfiguriert werden:

```
<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>
```

Mit dieser Konfiguration stehen folgende Services zur Verfügung:

Verfügbare Dienste

Administration

Für die Administration stehen unterschiedliche Dienste zur Verfügung. Diese können im Standardfall nur mittels POST Requests verwendet werden. Falls jedoch benutzerdefinierte Dienste registriert wurden, können diese auch mittels GET Request angesprochen werden.

Folgende Dienste stehen standardmässig zur Verfügung:

- Anmeldung testen
- Passwort ändern
- Datenbank prüfen

Anmeldung testen

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

Der Request benötigt eine HashMap im JSON Format.

Beispiel:

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

Der Benutzername kann auch weggelassen werden. In diesem Fall wird dann der Parameter aus dem URL als Benutzername verwendet.

POST-Response

Wenn die Anmeldung erfolgreich war, wird kein Response generiert und der Status Code ist 204 (SUCCESS_NO_CONTENT).

Passwort ändern

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

Der Request benötigt eine HashMap im JSON Format.

Beispiel:

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

Der Benutzername kann auch weggelassen werden. In diesem Fall wird dann der Parameter aus dem URL als Benutzername verwendet.

POST-Response

Wenn das Passwort geändert wurde, wird kein Response generiert und der Status Code ist 204 (SUCCESS_NO_CONTENT).

Datenbank prüfen

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

GET-Response

Wenn die Prüfung erfolgreich war, wird kein Response generiert und der Status Code ist 204 (SUCCESS_NO_CONTENT). Wenn die Datenbank nicht verfügbar ist, wird der Status Code 500 (SERVER_ERROR_INTERNAL) geliefert. Es ist ebenfalls möglich das die Konfiguration nicht gefunden werden konnte. In diesem Fall wird der Status Code 503 (SERVER_ERROR_SERVICE_UNAVAILABLE) geliefert.

Benutzerdefinierte Dienste

Wenn man einen eigenen Dienst zur Laufzeit registrieren möchte, kann dies mittels

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

erfolgen. Der Dienst kann entweder per GET oder POST Request angesprochen werden, je nachdem ob ***IACustomServiceGetDelegate*** oder ***IACustomServicePostDelegate*** verwendet wird.

Der Aufruf erfolgt mittels:

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

Storage Zugriff (CRUD, Meta Data)

- Select
- Insert
- Update
- Delete
- Meta Daten

GET-Request (Select)

Alle Daten abfragen:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME`

Genau einen Datensatz abfragen:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME/PRIMARY_KEY`

Wenn der PK aus mehreren Spalten zusammengesetzt ist, müssen die Query Parameter verwendet werden:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME?PKCOLUMN=VALUE&PKCOLUMN2=VALUE2`

Die Query Parameter können auch verwendet werden, um Filterungen mit Spalten, die nicht PK Spalten sind, durchzuführen.

Lesen Sie mehr über komplexe Abfrage Parameter?

GET-Response

Der Response enthält immer eine Liste von HashMaps im JSON Format. Als Key wird die Spaltenbezeichnung verwendet.

Beispiel:

```
[ { "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)

Einen neuen Datensatz einfügen:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME`

Der Request benötigt eine HashMap im JSON Format. Als Key wird die Spaltenbezeichnung verwendet.

Beispiel:

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

POST-Response

Der Response liefert den vollständigen Datensatz im 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)

Einen Datensatz per PK aktualisieren:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME/PRIMARY_KEY`

Wenn der PK aus mehreren Spalten zusammengesetzt ist oder der Datensatze nicht über den PK identifiziert werden sollen, müssen die Query Parameter verwendet werden:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME?COLUMN=VALUE&COLUMN2=VALUE2`

Der Request benötigt eine HashMap im JSON Format. Als Key wird die Spaltenbezeichnung verwendet.

Beispiel:

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

```
}
```

Es gilt zu beachten, das PK Spalten nicht aktualisiert werden.

PUT-Response

Der Response liefert den vollständigen Datensatz im 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)

Genau einen Datensatz löschen:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`/`**`PRIMARY_KEY`**

Wenn der PK aus mehreren Spalten zusammengesetzt ist, müssen die Query Parameter verwendet werden:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`?`**`PKCOLUMN=VALUE`**`&`**`PKCOLUMN2=VALUE2`**

DELETE-Response

Der Response liefert die Anzahl der gelöschten Datensätze im JSON Format (als Nummer):

```
42
```

OPTIONS-Request (Meta Daten)

Meta Daten abfragen:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**

OPTIONS-Response

Der Response liefert die Meta Daten im 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"
    ]
}
```


Aufruf von Actions

Die Server-side Actions können sowohl direkt vom Life-cycle Objekt als auch von verwendeten Business Objekten aufgerufen werden. Die Parameter-Übergabe ist ebenfalls möglich.

- Action ohne Parameter
- Action mit Parameter

GET-Request

Aufruf einer Server-side Action (ohne Parameter):
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/action/ACTION_NAME`

Aufruf einer Methode eines Business Objektes (ohne Parameter):
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/object/OBJECT_NAME/ACTION_NAME`

GET-Response

Der Response liefert den Rückgabewert der Action im JSON Format.

POST/PUT-Request

Aufruf einer Server-side Action (mit Parameter):
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/action/ACTION_NAME`

Aufruf einer Methode eines Business Objektes (mit Parameter):
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/object/OBJECT_NAME/ACTION_NAME`

Der Request benötigt ein Array von Objekten, befüllt mit den Parametern für die Action.

Beispiel:

Action:

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

JSON Request:

```
[123,1]
```

POST/PUT-Response

Der Response liefert den Rückgabewert der Action im JSON Format.

Anwendungsbeispiele

Integration

Mittels 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);
```

Mittels 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 für die Lifecycle Objekte: Session und Address

Hinweis

Bei Action calls müssen die korrekten Datentypen verwendet werden! Generell sollte auf primitive Datentypen, als Parameter, verzichtet und anstatt von Arrays sollte das List Interface verwendet werden. Weiters empfiehlt sich die Nutzung von Number für alle numerischen Werte. Dadurch werden Probleme aufgrund der JSON Serialisierung vermieden.

Als Life-Cycle Name sollte der Voll qualifiziert Klassenname, mit Package, verwendet werden. Wenn nur der simple Klassen Name verwendet wird, versucht JVx eine passende Klasse zu ermitteln. Sollten mehrere Klassen in Frage kommen, dann wird keine Klasse verwendet. Sie können optional einen Suchpfad in der config.xml der Applikation definieren:

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

Weitere Details über dieses Feature finden Sie im Support System.

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

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

Last update: **2022/11/17 11:12**