DB Portal Internal API Specifications

This document contains detailed and complete API specifications for how the AngularJS front end of the DB portal interacts with the backend via web-services. API's used by supporting softwares are also documented. API's opened to public for integrating with the portal, or using any of the portal features are outside the scope of this document.

Sandbox Location: http://sb-api.blobcity.net
Production Location: https://portal-api.blobcity.com

Quick Links / Table of Contents

Conventions

Request Conventions

`[METHOD] url-part?params`

A request URL is expressed in the format mentioned above. The `[METHOD]` part indicates the request method type; usually one of `[GET]` or `[POST]`.

The url-part corresponds to part of the URL after the domain. If full URL is:

`http://sb-api.blobcity.net/rest/user/login`

The url-part is:

`/rest/user/login`

The equivalent URL for production derived from the URL part would be:

`https://portal-api.blobcity.com/rest/user/login`

The params are the various parameters accepted by the request. An actual parameter string would look as shown below.

`param1=value1&param2=value2`

The above string has two parameters namely param1 and param2 having values value1 and value2 respectively. For sake of brevity where the values are not of importance to this document the params are simply written as

`param1=&param2=`

Sandbox Only Requests

`[SBX] [GET] /rest/user/delete-all`

Some APIs are valid on sandbox only and disabled on production. Such APIs are marked in the document with the `[SBX]` tag.

Response Conventions

`[value1, value2, value3]`

A response of the above form indicates a JSON Array as response. The elements are ordered and have to be processed assuming order guarantees. Any number of elements could be present in the array. The array could be an empty array. For sake of brevity in places where the values contained in the array are not of importance to this document, the array would be documented as empty, but actual values could be expected by the problem.

```
{“param1”:value1, “param2”: value2}
```

A response of the above form indicates a JSON Object as response. Any empty JSON Object is also treated as a valid response. The JSON Object could contain sub JSON objects and arrays within the object.

```
[STRING] some response
```

A response of the above form indicates a String response. Any string could be provided in such a response. For better readability the term `[STRING]` is used to mark responses that are of type String. The `[STRING]` term is not expected to be present in the actual response.

User Module

Login

```
[GET][POST] /rest/user/login?email=&password=
```

Success Response

```
[true, “session token”]
```

Failure Response

```
[false]
```

Registration

```
[GET][POST] /rest/user/register?name=&email=&password=
```

Success Response

```
[“success”]
```

Failure Response

```
[“failure”, “failure cause message”]
```

Verify Email

`[GET][POST] /rest/user/verify-email?token=`

Verification Confirmation

`[STRING] success`

Verification Failure

`[STRING] failure`

Request Password Reset / Forgot Password

`[GET][POST] /rest/user/request-new-password?email=`

This is to be called when the user remembers his email, but has forgotten the password. This request will result in password reset process initiation. An email will be sent to the user with further steps.

Success Response

`[STRING] success`

Failure Response

`[STRING] failure`

Reset Password

`[GET][POST] /rest/user/reset-password?token=&password=`

This is to be called from the reset password page the user lands on after clicking on the reset password link present in a reset password email. The token is to be the token captured from the email URL, which will be present as an HTTP parameter. The password will be the new password that is to be set to the users account.

The function succeed only if the token is valid. All issued tokens are for one time use and valid for 1 hour only.

Success Response

`[STRING] success`

Failure Response

`[STRING] failure`

Change Password

`[GET][POST] /rest/user/change-password?st=&oldpass=&newpass=`

This is to be called when the user is authenticated and wants to perform a password change. The `st` parameter needs to be a valid session token of the user for the request to succeed. The `oldpass` entered needs to be the valid current password. The `newpass` will be the value to which the password will be set if the `st` and `oldpass` combination is valid.

Success Response

`["success"]`

Failure Response

`["failure", "failure reason message"]`

Check Session Token Valid

`[GET][POST] /rest/user/token-valid?st=`

Used to check the validity of an existing session token.

Responses

```
["valid", validity time remaining in milliseconds]
["invalid"]
```

Fetch Session

`[GET][POST] /rest/user/session?st=`

Used to fetch details of the user's session. Details are returned only if the session token is valid. If not a login page should be displayed.

Valid Session Response

`[JSON] {"ack":1, "name":"name", "email": "test@test.com"}`

Invalid / Expired Session Response

`[JSON] {"ack":0}`

Delete All User Accounts

`[SBX] [GET] /rest/user/delete-all`

Deletes all users accounts, both verified and not-yet-verified.

No response provided. A HTTP 200 OK response should indicated that all accounts have been deleted.

Logged In Home Page

`[GET][POST] /rest/account/home?st=`

Gets all information to be displayed on the post login home page. The information is returned only as long as the session token belongs to a valid session.

Valid Session Response

```
[JSON]
{
      “ack”:1,
      “warnings”: [“warning 1”, “warning 2”],
      “info”: [“info 1”, “info 2”],
      “ds”: {
            “name-set”: true,
            “name”: “XXXXX-YOUR_NAME”
      },
      “usage”: {
            “usage”:0,
            “unit”: “MB”,
            “note”: “First 100MB free”
      },
      “rec-charge”: {
            “charge”:0,
            “currency-sym”: “$”,
            “note”: “$10/GB/month beyond free tier”
      },
      “unbilled”: {
            “charge”: 0,
            “currency-sym”: “$”,
            “note”: “Next charge on 30 June 2017 or $100”
      },
```

```
      "ftp": {
            "toggle-on":true,
            "ip": "xxx.xxx.xxx.xxx",
            "port": "xxxx",
            "user": "username"
      }
}
```

The `warnings` tag contains information to be displayed on the top red bar. These are important updates about the customer's account and need immediate action. There will be one bar shown per warning. The ordering of the bars should be as per the order of elements in the array, with the first element of the array being displayed top most on the page.

The `info` tag contains information similar to the `warnings` tag. However `info` level items are shown in an orange background dialog instead of a red background dialog.

Invalid Session Response

```
[JSON] {"ack":0}
```

Set DS Name

```
[GET][POST] /rest/db/set-do?st=&ds=
```

Used to set the name of the datastore. This URL will get called only once in the lifetime of each user's account with BlobCity. The session token must be valid and must not be associated with a users account where this URL has already been invoked. The name of the datastore to which the name is to be set is provided as the second parameter.

Success Response

```
[JSON] {"ack": "1"}
```

Failure Response

```
[JSON] {"ack": "0"}
```

Set FTP On

```
[GET][POST] /rest/db/set-ftp-on?st=&ds=
```

Sets the FTP to on and returns important information related to the FTP user account. The operation is a no-op if the ftp is already on and will return a success response in such a case. The operation returns an error if `st` is invalid or the `ds` cannot be found, or the user does not have access to the specified `ds` to control its ftp state.

Success Response

```
[JSON]
{
      "ack": "1",
      "ip": "xxx.xxx.xxx.xxx",
      "port": "xxxx",
      "user": "username"
}
```

Failure Response

```
[JSON] {"ack": "0"}
```

Set FTP Off

```
[GET][POST] /rest/db/set-ftp-off?st=&ds=
```

Sets the FTP to off and confirms with a success / failure response. The operation is a no-op if the FTP is already off. The operation returns an error if `st` is invalid or the `ds` cannot be found, or the user does not have access to the specified `ds` to control its ftp state.

Success Response

```
[JSON] {"ack": "1"}
```

Failure Response

```
[JSON] {"ack": "0"}
```

Fetch FTP password

```
[GET][POST] /rest/db/get-ftp-pass?st=&ds=
```

Gets the password for the FTP user account associated with the specified datastore. The operation will return the password only if the ftp is currently enabled, the `st` is valid and the `ds` is accessible to the user as identified by the `st`.

Success Response

```
[JSON]
{
      “ack”: “1”,
      “pass”: “password”
}
```

Failure Response

```
[JSON] {“ack”: “0”}
```

Reset FTP password

`[GET][POST] /rest/db/reset-ftp-pass?st=&ds=`

Sets a new password for the FTP user account associated with the datastore. The operation is successful only if the FTP on the specified `ds` is currently enabled, the `st` is valid and `ds` is valid and accessible to the user as identified by the `st`. A newly created password is returned in response.

Success Response

```
[JSON]
{
      “ack”: “1”,
      “pass”: “password”
}
```

Failure Response

```
[JSON] {“ack”: “0”}
```

Fetch User Cards

`[GET][POST] /rest/user/cards?st=`

Fetch all credit cards associated with the users account.

Success Response

```
[JSON]
{
```

```
      "ack": "1",
      "cards": [
            {"id": "id1", "card": "...1234", "primary": true},
            {"id": "id2", "card": "...9876", "primary": false}
      ]
}
```

Failure Response

```
[JSON] {"ack": "0"}
```

Set Card Primary

`[GET][POST] /rest/user/set-card-primary?st=&id=`

Makes the specified credit card the primary credit card associated with the users account. The operation is a no-op if the specified card is already the primary card or is primary by default on being the only card associated with the account. The operation will give error if `st` is not valid or no card with the specified `id` is found associated with the user's account.

Success Response

```
[JSON] {"ack": "1"}
```

Failure Response

```
[JSON] {"ack": "0"}
```

Delete Card

`[GET][POST] /rest/user/delete-card?st=&id=`

Deletes the specified card associated with the user's account. The operation will return error if `st` is invalid. The operation return a success response and will be a no-op if `st` is valid, but a card with the specified `id` is not found.

Success Response

```
[JSON] {"ack": "1"}
```

Failure Response

```
[JSON] {"ack": "0"}
```

Add Card

```
[POST] /rest/user/add-card

[JSON]
{
      “st”: “”,
      “stripe-token”: “”,
      “country”: “”,
      “pin-code”: “”
}
```

This service end point should be called as the final step of the process explained at https://stripe.com/docs/elements#create-token

Card information should be submitted to Stripe in a PCI DSS compliant manner using the stripe.js file and procedure explained in the URL. As a final step, the token returned by stripe should be passed to the BlobCity API’s so that BlobCity can know of a credit card addition and would be able to save and use the card in the future.

The operation returns a success if the passed stripe-token can be verified to be valid from Stripe and the st is valid. It will return a failure response otherwise.

The country and pin-code are additional parameters to be captured on the page. These parameters correspond to the users billing country associated with the card, and the address pin code of the billing address of the credit card. The form submission should not be accepted unless these details are provided.

Success Response

```
[JSON] {“ack”: “1”}
```

Failure Response

```
[JSON] {“ack”: “0”}
```

Set Account Details

```
[POST] /rest/account/set-details

[JSON]
{
```

```
      "st": "",
      "name": "full name",
      "billing" : {
            "company": "company name",
            "addr1": "address line 1",
            "addr2": address line 2",
            "city": "city",
            "state": "state",
            "pin": "pincode"
      }
}
```

Sets the details of the account to the details passed. Overwrites any existing values with the new values. The operation is successful only if the value of `st` is valid.

Success Response

```
[JSON] {"ack": "1"}
```

Failure Response

```
[JSON] {"ack": "0"}
```