PROJECT_NAME Documentation

Release 0.1.0

AUTHOR

Feb 07, 2018

Contents

JBConnect - a tightly integrated analysis framework for JBrowse

JBrowse does not require JBConnect to operate.

JBConnect is a sails.js application and provides a job execution engine (kue).

JBConnect can be extended with *jbh-hook* modules that extend both server and client-ends in a single package. (see: *Installing JBServer jbh-hooks* and *JBServer jbh-hooks*)

JBServer provides the following features:

Job queue execution engine (kue)
Tight Integration with JBrowse client
RESTful APIs (**Blueprints**)
Database ORM, Any Database, MySql, PostgreSQL Mongo, Redis, local (**Waterline**)
Express-based Compatible routes & Middleware
Socket.io - sub/pub, WebSockets, Auto Integrate Models
Passport.js - role-based security, access control, OAuth
Extensible plugin framework based on Sails.js Installable Hooks
Grunt Customizable asset workflow, LESS, SASS, Stylus

CHAPTER 1

Contents

1.1 Quick Start

The quick start instructions demonstrate installing JBServer with JBrowse loaded as a an NPM module (since JBServer is generally intended to be a companion of JBrowse. JBrowse may also be installed in a separate directory. (See *JBrowse Installed In Separate Directory*.)

1.1.1 Pre-Install

JBServer requires redis as a pre-requisite, which is only used by the queue framework (kue).

Install and run *redis*

```
yum install redis
redis-server
```

1.1.2 Install

Install the JBServer and JBrowse. jb_setup.js ensures the sample data is loaded.

```
git clone http://github.com/gmod/jbserver
cd jbserver
npm install
```

1.1.3 Run

Launch the server.

```
sails lift
```

From a web browser, access the application.

```
http://localhost:1337/jbrowse
```

The default username/password: juser/password

1.2 Features

1.2.1 JBConnect is a Sails.js application

JBConnect utilizes Sails.js, provideing the following features:

Javascript (NODE-based)
Blueprints Auto-generate CRUD APIs, ESTful APIs, Socket.io (events)
Waterline ORM, Any Database, MySql, PostgreSQL Mongo, Redis, local
Express-based Compatible routes & Middleware
Socket.io sub/pub, WebSockets, Auto Integrate Models
Passport.js role-based security, access control, OAuth
Installable Hooks (jbh-*), Extensible plugin framework
Grunt Customizable asset workflow, LESS, SASS, Stylus
Front-End Agnostic Angular, backbone, bootstrap, ember, …

Directory Layout

```
JBConnect project
├── api                         Standard sails API layout
├── assets                      contains client accessible assets
├── bin                         Utilities
├── config                      Configuration files.
│   ├── globals.js              Config file for module
│   └── libroutes.js            Library Routes
├── data                        Contains the local database file
│   └── localDiskDb.db          Local database file
├── docs                        Documentation
│   └── genapi-rst              jsdoc generated rst files
├── plugins                     Client-side Plugins
│   └── JBClient                Client plugin
├── test                        Test
├── views                       View pages
├── Gruntfile.js                Grunt config
├── jbutil                      JBServer Utility
└── package.json
```

jbutil Command

`jbutil` is a setup/configuration utility for JBConnect. jbh-hook can extend `jbutil` command options. (see: jbs-hooks-extend)

This example shows that `jbh-jblast` adds a number of commands to `jbutil`

```
$ ./jbutil --help
Usage: jbutil [OPTION]
      --config            display merged config
      --blastdbpath=PATH  (jblast) existing database path
      --setupworkflows    (jblast) [install|<path>] "install" project wf, or specify .
↪ga file
      --setuptools        (jblast) setup jblast tools for galaxy
      --setupdata         (jblast) setup data and samples
      --setupindex        (jblast) setup index.html in the jbrowse directory
      --setuphistory      setup history
  -h, --help              display this help
```

Queue Framework

JBConnect uses Kue as the basis for the queue framework. However, Kue is encapsulated in the Job model/controller. Since Kue requires redis database, redis server must be running. An integrated job panel is available when the JBClient plugin is active. (see: *JBClient Plugin*)

For diagnostic purposes, a Kue utility can be used to view/manage the Kue database content: `http://localhost:1337/kue`

This route can be disabled with in config/http.js.

Configuration

JBrowse configurations are in `config/globals.js`

```
jbrowse: {
    jbrowseRest: "http://localhost:1337",        // path accessible by web browser
    jbrowsePath: jbPath,                         // or point to jbrowse directory (ie.
↪"/var/www/jbrowse/")
    routePrefix: "jbrowse",                      // jbrowse is accessed with http://
↪<addr>/jbrowse
    dataSet: [
        {
            dataPath: "sample_data/json/volvox" // registered datasets.
        }
    ]
}
```

Client-Side Plugins

Client-side plugins are defined in the *plugins* directory. Plugins will automatically be accessible by the client side. However, they need to be configured in the *plugins:* section of the particular dataset in JBrowse *trackList.json*.

Plugin routes are virtual routes With respect to the client side, they appear in the client-side's plugin directory only when the server is lifted.

Library Routes

libroutes maps dependancy routes for client-side access. These provide access to modules that are required for use by the client-side plugins or other client-side code. The framework looks for libroutes.js in jbh- (hook modules), in their respective config directories

For example: for the module jquery, The module is installed with 'npm install jquery' The mapping the mapping 'jquery': '/jblib/jquery' makes the jquery directory accessible as /jblib/jquery from the client side.

Library Routes are virtual routes, in that they only exist when the server is lifted. They are virtually mapped to their respective locations in the node_modules directory.

`config/libroutes.js`:

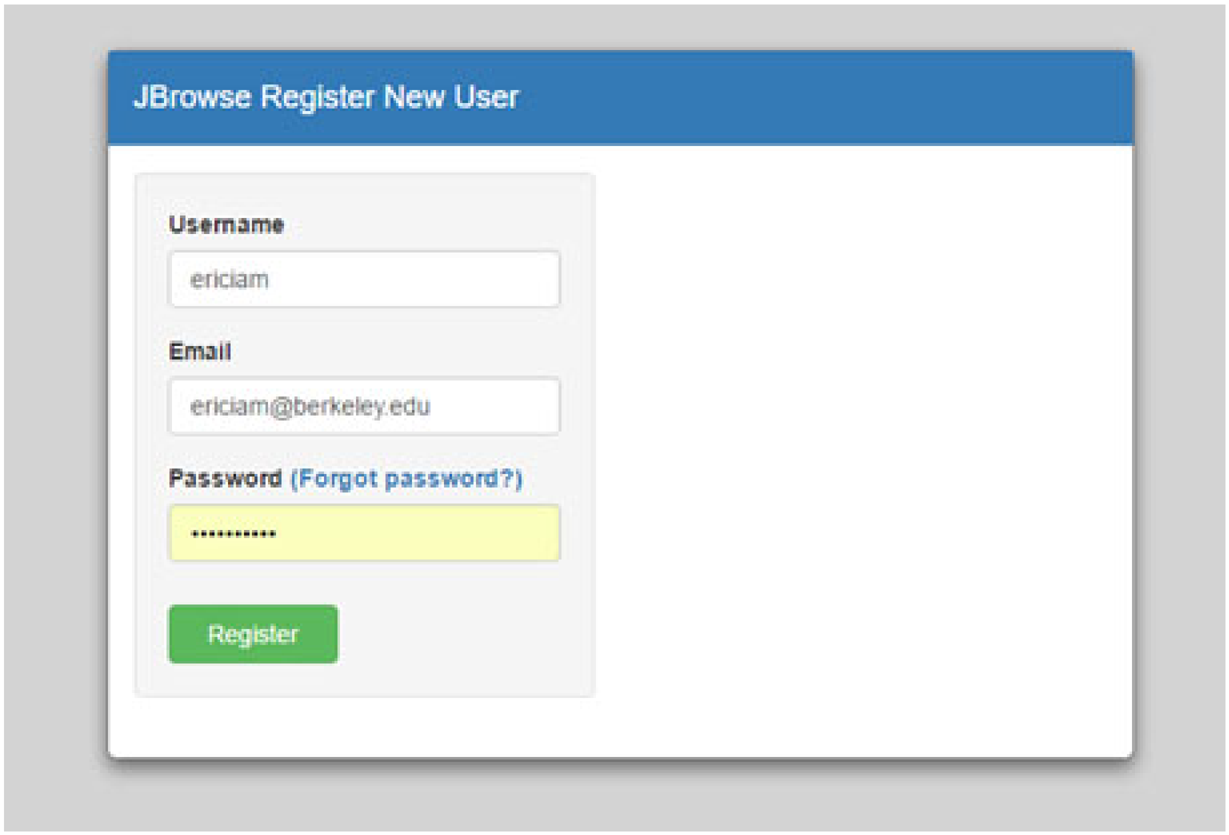
```
module.exports = {
    lib: {
            'jquery.mb.extruder':        '/jblib/mb.extruder',
            'jQuery-ui-Slider-Pips':     '/jblib/slider-pips',
            'jquery-ui-dist':            '/jblib/jquery-ui'
    }
};
```

Standalone Register / Login / Logout Routes

Stand-alone routes allow for basic register/login/logout functionality free from the JBrowse interface.

Stand-alone interfaces use bootstrap

Register: `http://<address>:1337/register`


JBrowse Register New User

Username

ericiam

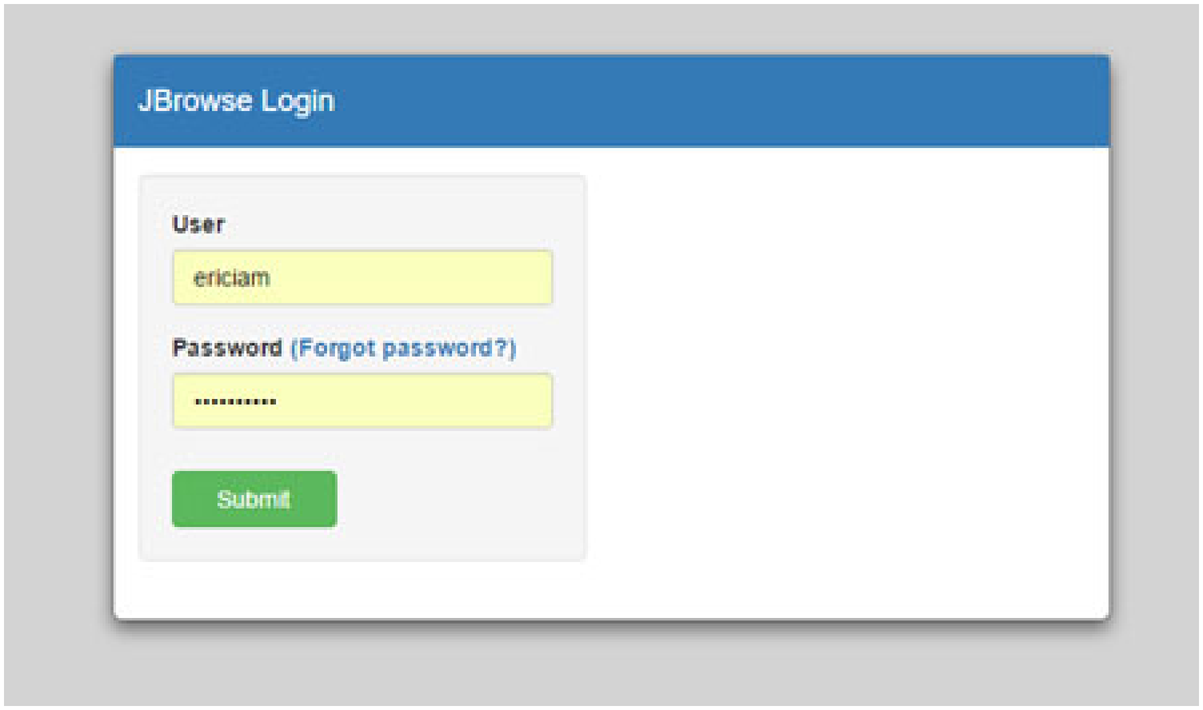
Email

ericiam@berkeley.edu

Password (Forgot password?)

•••••••••••

Login: `http://<address>:1337/login`



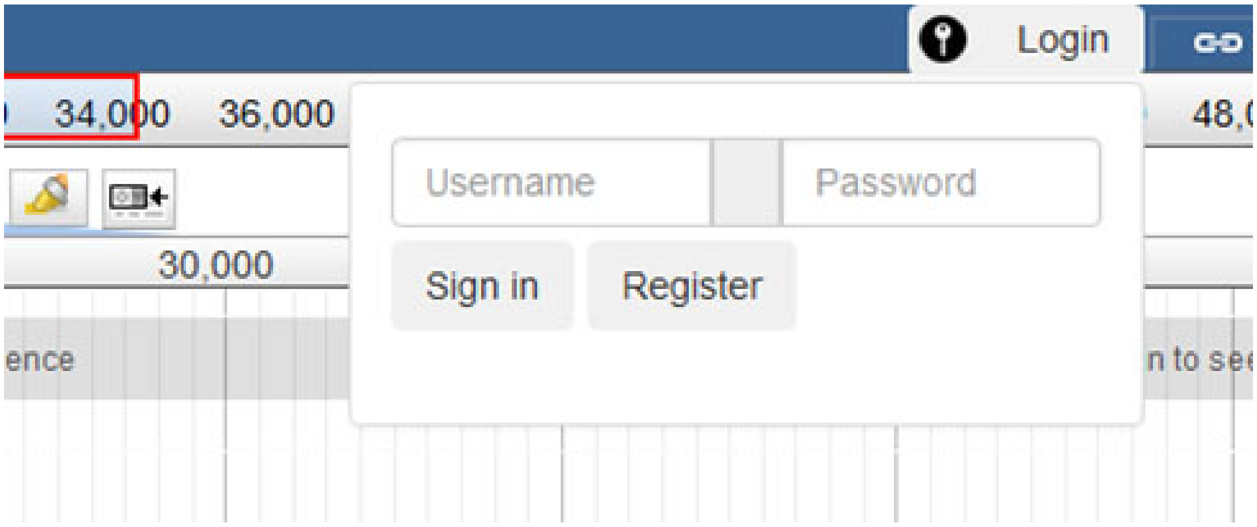
Logout: `http://<address>:1337/logout`

Get Login State: `http://<address>:1337/loginstate`

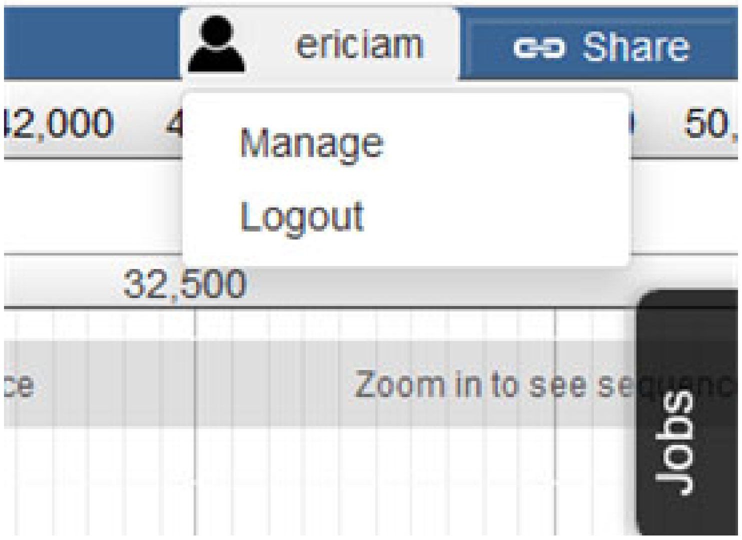
The routes are defined in *config/routes.js*.

Login/Logout Panel

Login Panel



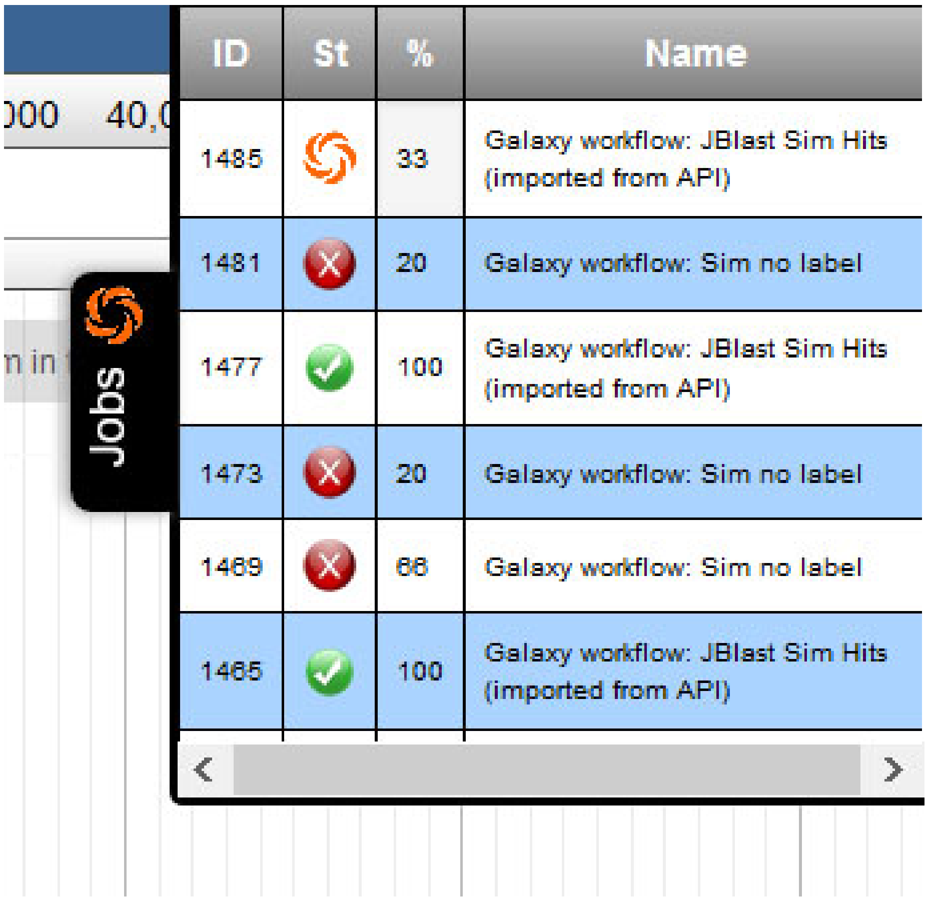
Loguot Panel



Job Queue Panel

JBConnect uses *Kue* as the queue framework. Since Kue requires *redis* database, redis server must be running. An integrated job panel is available when the JBClient plugin is active. (see: *JBClient Plugin*)

Integrated Job Panel:



ID	St	%	Name
1485	⟳	33	Galaxy workflow: JBlast Sim Hits (imported from API)
1481	✗	20	Galaxy workflow: Sim no label
1477	✓	100	Galaxy workflow: JBlast Sim Hits (imported from API)
1473	✗	20	Galaxy workflow: Sim no label
1469	✗	66	Galaxy workflow: Sim no label
1465	✓	100	Galaxy workflow: JBlast Sim Hits (imported from API)

Jservice Framework

todo

Test Framework

Test framework uses

- Mocha for unit test
- Nightwatch for end-to-end, supporting phantomjs, selenium and online service such as browserstack.
- Istanbul for coverage

To execute

`npm test`

by default nightwatch is setup for phantomjs. Selenium requires running an additional selenium server

`package.json`:

```
"scripts": {
  "test": "nyc node ./node_modules/mocha/bin/mocha test/bootstrap.test.js test/
↪integration/**/*.test.js test/e2e/**/*.test.js --nightwatch-test phantomjs",
},
```

The option `--nightwatch-test` can be:

- `phantomjs` - runs client tests with phantomjs
- `selenium` - runs client tests with selenium
- `browserstack` - runs client test with selenium through remote browserstack account.

Documentation Framework

For integrated documentation, JSdoc3 is used to generate API docs from code with jsdoc-sphinx, a jsdoc template that generates RestructuredText (RST) and Sphinx. This enables support for readthedocs.

See: RST/Sphinx Cheatsheet

`npm run gendocs`

1.3 Setup Options

1.3.1 JBrowse Installed In Separate Directory

The JBrowse directory can also be configured manually. (See jbs-globals-js)

1.3.2 Configuration Files

A number of configuration files are in the `./config` directory. A few of the more important ones (ones that JBSserver touches) are described mentioned in the table below. See Sails Configuration for a better description of the configuration framework.

jbs-globals-js	global configuration file
http.js	custom middleware and /jbrowse route is setup here.
passport.js, policies.js	passport framework and auth policies config
routes.js	various route config
connections.js	choice of database - local, mongo, mysql, . . . (we use local by default.) The DB file is in the `./data/localDiskDb.db`.

globals.js

Modify the configuration file as necessary.

To view aggregate configuration: `./jbutil --config`

The aggregate config file is the merged config of JBServer and its installed jbh- (hook) modules.

Edit config file: `nano config/globals.js`

```
jbrowse: {
    jbrowseRest: "http://localhost:1337",       // path accessible by web browser
    jbrowsePath: jbPath,                        // or point to jbrowse directory (ie.
↪"/var/www/jbrowse/")
    routePrefix: "jbrowse",                     // jbrowse is accessed with http://
↪<addr>/jbrowse
    dataSet: [
        {
            dataPath: "sample_data/json/volvox" // datasets.
        }
    ]
}
```

1.3.3 Installing JBServer jbh-hooks

A 'JBServer Hook' is basically an *installable sails hook* with specific methods for extending JBServer. JBServer hooks must have the prefix `jbh-` prepended to the name. For example: jbh-jblast. When the hook is installed (i.e. `npm install jbh-jblast`). JBServer will automatically integrate a number of features of the hook directly into JBServer upon `sails lift`.

The jbh- hook can extend JBServer in the following ways:

- Extend models, controllers, policies and services
- Integrated client-side JBrowse plugins injection
- Integrated client-side npm module injection
- Integrated configuration tool (jbutil)
- Aggregated configurations

Installing a hook:

`npm install jbh-<hook name>` (i.e. jbh-jblast)

For detailed info on jbh-hooks, see: *JBServer jbh-hooks*

1.3.4 JBClient Plugin

JBrowse GUI intetrated interfaces are available when the `JBClient` plugin is configured on in the JBrowse client.

To enable integrated features within the JBrowse app, modify the dataset's `trackList.json`, adding `JBClient` plugin to the configuration.

Note: the JBClient plugin is not physically in the JBrowse plugin directory. It is available as a route.

```
"plugins": [
  "JBClient",                   <-----
  "NeatHTMLFeatures",
  "NeatCanvasFeatures",
  "HideTrackLabels"
],
```

1.3.5 Jservice Configuration

Jservices are a special type of service that are used to extend RESTful api service and serve processing for job operations.

Configuration is defined in config/globals under the jbrowse section under service.

A definition: <indexname>: {name: <servicename>, type:<type>, alias:<alias> }

where:

- indexname - is the reference name service (generally the same as servicename)
- servicename - is the name of the service reference the service code in api/services.
- type - is the type of service. either "workflow" or "service"
- alias - (optional) if specified, the service can also be referenced by the alias name.

jservice type:

- workflow - service can serve job execution
- service - service only serves RESTful interfaces

```
// list of services that will get registered.
services: {
    'basicWorkflowService':     {name: 'basicWorkflowService',  type: 'workflow',␣
↪alias: "jblast"},
    'filterService':            {name: 'filterService',         type: 'service'},
    'entrezService':            {name: 'entrezService',         type: 'service'}
},
```

1.4 JBServer jbh-hooks

A 'JBServer jbh-hook' is an installable sails hook with specific methods for extending JBServer. jbh-hooks must have the prefix `jbh-` prepended to the name. For example: jbh-jblast. When the hook is installed (i.e. `npm install jbh-jblast`). JBServer will automatically integrate a number of features of the hook directly into JBServer upon `sails lift`.

jbh-* Integration * Models, Controller, Policies, Services (via marlinspikes) * jbutil command extensions * client-side plugin exposure, and client module dependencies * routes via jservice framework or via controllers * job jservice framework

These features are described below.

1.4.1 Directory Layout

This is the standard directory layout of a jbh- module

```
jbh- hook project
├── api                              Standard Sails modules layout
│   ├── controllers
│   ├── hooks
│   │    └── Myhook
│   │         └── index.js            The main hook
│   ├── models
│   ├── policies
│   └── services
├── bin
│   └── jbutil-ext.js                jbutil extension module
├── config
│   ├── globals.js                   Config file for module
│   └── libroutes.js                 Library Routes
├── plugins                          Client-side Plugins
│   └── PluginA
├── Gruntfile.js
└── package.json
```

1.4.2 package.json

Standard sails hooks should contain the following section in the package.json

```
"sails": {
  "isHook": true,
  "hookName": "jblast"
}
```

1.4.3 Configurations (globals.js)

This file contains config options that are specific to the hook module. These config options are merged with other jbh-hooks and the JBServer globals.js.

From JBServer, use `./jbutil --config` to see the aggregated config.

1.4.4 Configuration (config.js)

This config file resides in the root of the app and can override config/globals.js and any config/globals.js from hooks that may be installed.

1.4.5 Library Routes (libroutes)

libroutes maps dependancy routes for client-side access. These are routes to modules that are required for use by the client-side plugins or other client-side code. The framework looks for libroutes.js in jbh- (hook modules), in their respective config directories

For example: for the module jquery, The module is installed with 'npm install jquery' The mapping the mapping 'jquery': '/jblib/jquery' makes the jquery directory accessible as /jblib/jquery from the client side.

libroutes.js

```
module.exports = {
    lib: {
            'jquery.mb.extruder':        '/jblib/mb.extruder',
            'jQuery-ui-Slider-Pips':     '/jblib/slider-pips',
            'jquery-ui-dist':            '/jblib/jquery-ui'
    }
};
```

1.4.6 Client-Side Plugins

Client-side plugins in this directory are made available on the JBrowse client side as routes in the JBrowse plugin directories upon `sails lift`.

1.4.7 Extending jbutil

jbutil-ext.js is the file that is read by JBServer and integrates additional command options into jbutil (the JBServer utility).

- it extends new command line options
- it extends the help (i.e. `./jbutil --help`)

1.4.8 Sails Module Layout

This is the standard sails directory layout for modules of a sails hook. The framework uses marlinspike to integrate controllers, models, policies, and services into JBServer.

ref: marlinspike

```
jbh- project
├── api                         Standard Sails modules layout
    ├── controllers
    ├── hooks
    ├── models
    ├── policies
    └── services
```

1.4.9 The Main Hook

index.js should not be modified.

This core fragment starts the initialization of JBConnect.

1.4.10 config Directory

This directory contain config files for the hook. If the name matches it's counterpart file in JBServer's config directory, the configurations similar files will be merged.

1.4.11 JService Framework

todo

1.5 API

1.5.1 Module: `TrackController`

Local Navigation

Description

REST interaces for TrackController

this is a inline code: `a = b + c` and txt after it.

Test *italics* and **bold** and `monspaced` text.

Function: `get`

enumerate tracks or search track list.

Get all tracks:

`GET /track/get`

Get filtered tracks by dataset:

`GET /track/get?id=1` where id is the dataset id

`GET /track/get?pat=sample_data/json/volvox`, where path is the dataset path

`get` (*req*, *res*)

Arguments

- **`req`** (*`object`*) – punctuary
- **`res`** (*`object`*) – postuary

Function: `add`

This is a description wonderful eggs hatch into lovely swans that lay golden eggs.

A	B
C	D

This is a *italics* in a function description:

```
{
    var x = 1;
    function abc(z) {
        console.log("z=",z);
    }
}
```

`add` (*req*, *res*)

Arguments

- **`req`** (*`object`*) – zingle mingle

Code block in param description:

```
{
    wifi: "sparkle",
    swindle: true
}
```

Arguments

- **`res`** (*`object`*) – sizzle

codeblock 2:

```
{
    apple: "crunch",
    pear: "green"
}
```

Function: `modify`

Description of Modify

```
var x = 1;
```

`modify` (*req*, *res*)

Arguments

- **`req`** (*`object`*) – very interesting

A	B
C	D

Arguments

- **`res`** (*`object`*) – nothing else follows

Function: `remove`

A fine thing to remove

A link to remember: Stack Overflow.

method 2 - Test hyperlink: SO.

method 3 - A hyper link in a new tab: .

`remove`(*req*, *res*)

Arguments

- **`req`** (*`object`*) – fizzle
- **`res`** (*`object`*) – frazzle

Index