Software setup components library
Technical documentation

Ver. 1.1

Table of Contents

Overview

The purpose of this document is to provide developers with an informative piece of documentation that will allow them to start any development project using a well-defined set of kick-start common functions.

Library contains components to work with database, prepare and track database versioning by executing migration SQL scripts. Functionality to manage users in the system via Web API (back-office) and via REST API provided by the system. In addition there are components for system monitoring including health check, thread dumps, current system configuration and environment properties that help to easily investigate problems on production systems.

Project structure and goals

Along with this document a developer will be able to find a development project called:
components.library
This project aims to give developers a common ground to start from when starting a new project and make sure that development methodologies are followed from project kick-off to end.
This project includes the following functions:

1. The ability to setup a new Data Base
2. The ability to setup and define all Data Base connections
3. The ability to define schema migrations of the Data Base
4. The ability to create and maintain users in the system
5. The ability to monitor the system
6. The ability to test and view all API’s defines in the system

In order to properly use this starter code this document describes each functionality separately using examples and screen shots in necessary

Technologies used for this project

- **Spring Framework:** main system framework. Used to control object lifecycle, dependency injection and integration with other frameworks and systems.

- **Spring Boot:** provides pom.xml structure with wide set of dependencies setup with versions that guarantee that 3rd party libraries not in conflict with each other.

- **Actuator:** provides system health check and other useful monitoring \ debug information via REST endpoints

- **Swagger:** provides auto-generated REST documentation and playground (web interface) for system REST interfaces

- **Togglz:** provides feature flags functionality in order to control system business flows without need to restart or redeploy the application. Supports large set of activation strategies (gradual rollout, user based, ip-based and many more)

- **Flyway:** db migration tool. Allows to control data base state, execute developer sql scripts in defined order and maintain migrations history in DB table

- **Vaadin:** back-office framework. Allows to translate java code (swing style) to HTML (supports ajax, push notifications, responsive UI, custom components etc). Vaadin is very powerful tool for quick building of back-office solutions for enterprise systems

- **JUnit:** tests framework. Allows to write and execute system and unit tests for the system.

Project components

Configuration

Application main class:
com.it.server.Application
Starts the application with default users (dev/111111 and others)

Security config:
Placed in class com.it.server.SecurityConfig

Configures login url (to which user is redirected if not logged in), urls which do not require authentication etc.
In addition, contains configuration of which UserDetailsService is responsible for finding users in db to be validated during login process.

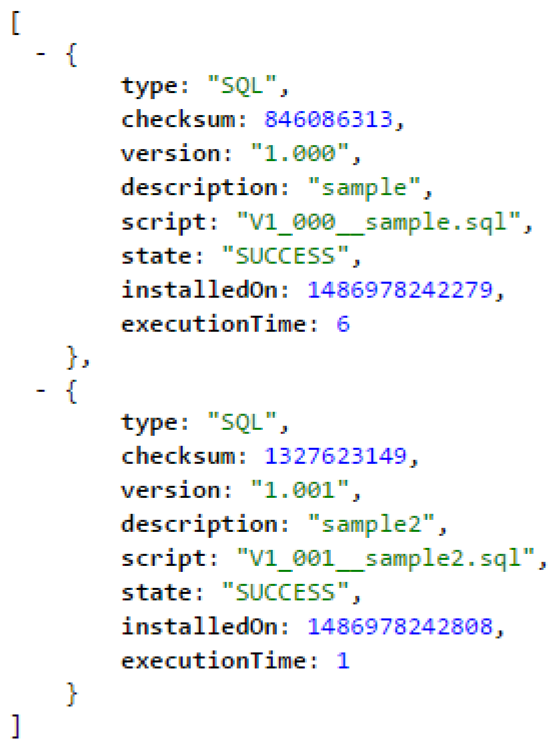
DB

Currently used inmemory hsqldb.
To use different database add application.yml file and configure the properties:

```
spring:

    datasource:
        name: <name>
        url: <db url>
        username: <db user>
        password: <db pass>
        driver-class: <driver>
```

Example db migration script located under classpath: db\migration.
/flyway – shows currently applied migrations



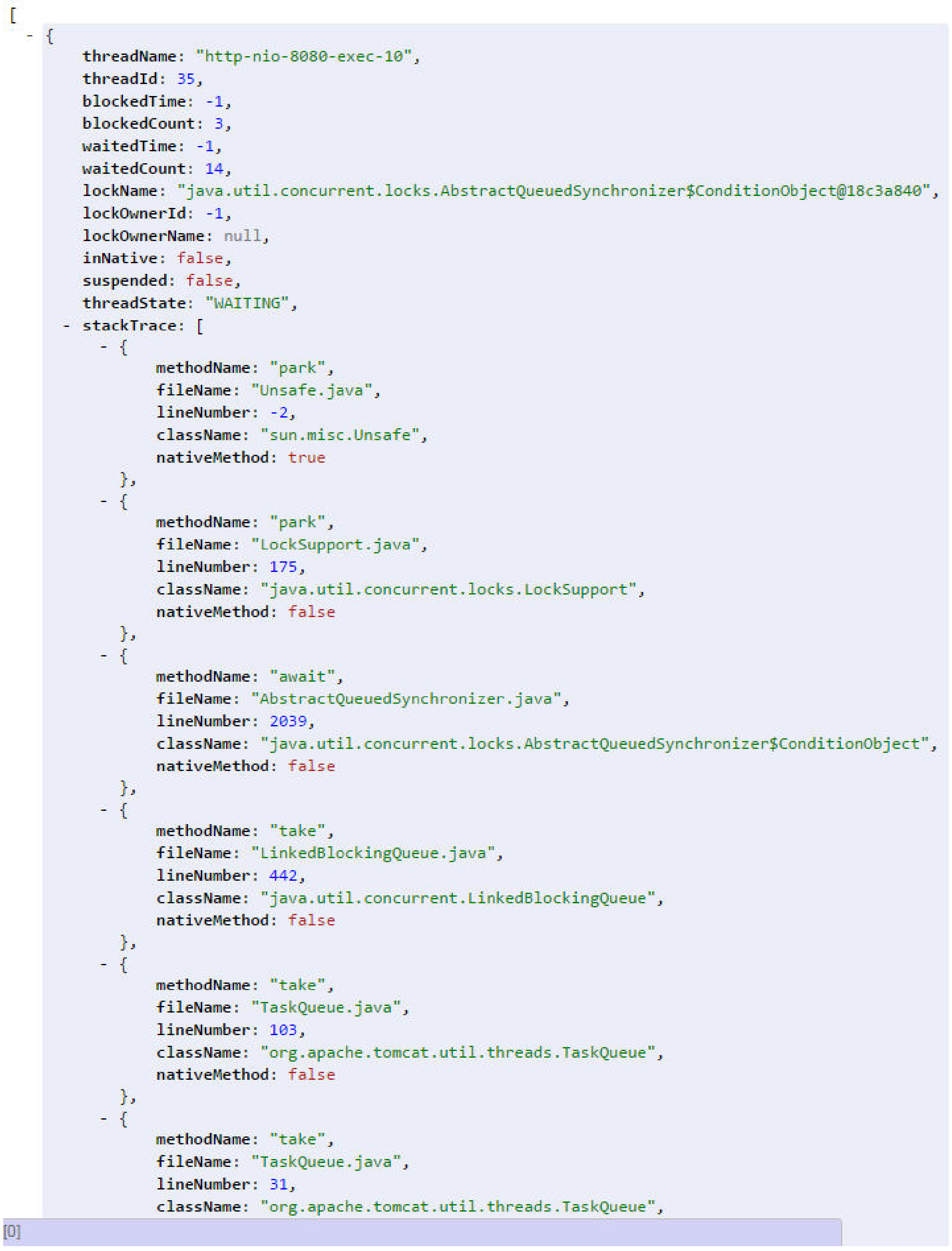
```
[
  - {
        type: "SQL",
        checksum: 846086313,
        version: "1.000",
        description: "sample",
        script: "V1_000__sample.sql",
        state: "SUCCESS",
        installedOn: 1486978242279,
        executionTime: 6
    },
  - {
        type: "SQL",
        checksum: 1327623149,
        version: "1.001",
        description: "sample2",
        script: "V1_001__sample2.sql",
        state: "SUCCESS",
        installedOn: 1486978242808,
        executionTime: 1
    }
]
```

com.it.repository.UserRepository - class for DB queries for User

System health check and status

Following endpoints provided to support system health check:

/health – Shows application health (shows that server is up, disk space, db connection status)
/metrics – Shows ‘metrics’ information for the current application (memory usage, threads, gc info etc)
/trace – Displays trace (history of http requests)
/dump – Displays dump of currently running threads



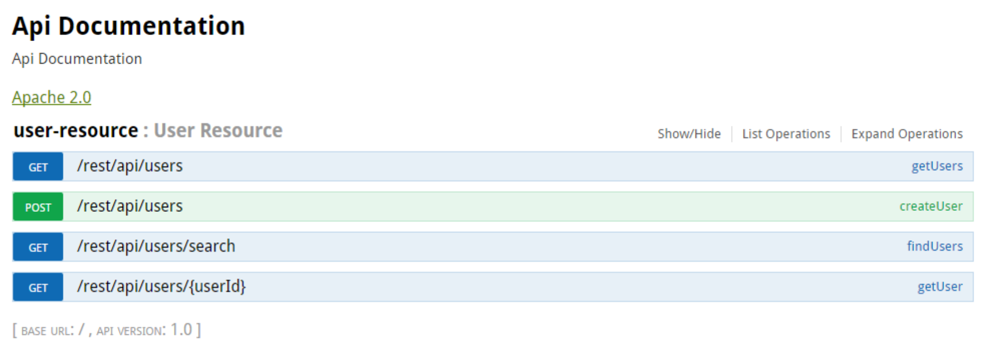
```
[
  - {
        threadName: "http-nio-8080-exec-10",
        threadId: 35,
        blockedTime: -1,
        blockedCount: 3,
        waitedTime: -1,
        waitedCount: 14,
        lockName: "java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@18c3a840",
        lockOwnerId: -1,
        lockOwnerName: null,
        inNative: false,
        suspended: false,
        threadState: "WAITING",
      - stackTrace: [
          - {
                methodName: "park",
                fileName: "Unsafe.java",
                lineNumber: -2,
                className: "sun.misc.Unsafe",
                nativeMethod: true
            },
          - {
                methodName: "park",
                fileName: "LockSupport.java",
                lineNumber: 175,
                className: "java.util.concurrent.locks.LockSupport",
                nativeMethod: false
            },
          - {
                methodName: "await",
                fileName: "AbstractQueuedSynchronizer.java",
                lineNumber: 2039,
                className: "java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject",
                nativeMethod: false
            },
          - {
                methodName: "take",
                fileName: "LinkedBlockingQueue.java",
                lineNumber: 442,
                className: "java.util.concurrent.LinkedBlockingQueue",
                nativeMethod: false
            },
          - {
                methodName: "take",
                fileName: "TaskQueue.java",
                lineNumber: 103,
                className: "org.apache.tomcat.util.threads.TaskQueue",
                nativeMethod: false
            },
          - {
                methodName: "take",
                fileName: "TaskQueue.java",
                lineNumber: 31,
                className: "org.apache.tomcat.util.threads.TaskQueue",
[0]
```

```
{
    mem: 955069,
    mem.free: 682193,
    processors: 4,
    instance.uptime: 1384494,
    uptime: 1393595,
    systemload.average: -1,
    heap.committed: 860672,
    heap.init: 262144,
    heap.used: 178478,
    heap: 3719168,
    nonheap.committed: 95888,
    nonheap.init: 2496,
    nonheap.used: 94399,
    nonheap: 0,
    threads.peak: 24,
    threads.daemon: 20,
    threads.totalStarted: 26,
    threads: 22,
    classes: 11946,
    classes.loaded: 11946,
    classes.unloaded: 0,
    gc.ps_scavenge.count: 8,
    gc.ps_scavenge.time: 208,
    gc.ps_marksweep.count: 3,
    gc.ps_marksweep.time: 344,
    httpsessions.max: -1,
    httpsessions.active: 1,
    datasource.primary.active: 0,
    datasource.primary.usage: 0,
    gauge.response.flyway: 95,
    gauge.response.trace: 15,
    gauge.response.rest.api.users.userId: 17,
    gauge.response.dump: 47,
    gauge.response.health: 38,
```

Auto generated API documentation with web interface

/swagger-ui.html shows available REST API endpoints that system provides. Also provides functionality to send requests to server and see the JSON responses

Additional configuration can be done in com.it.server.SwaggerConfig



Api Documentation

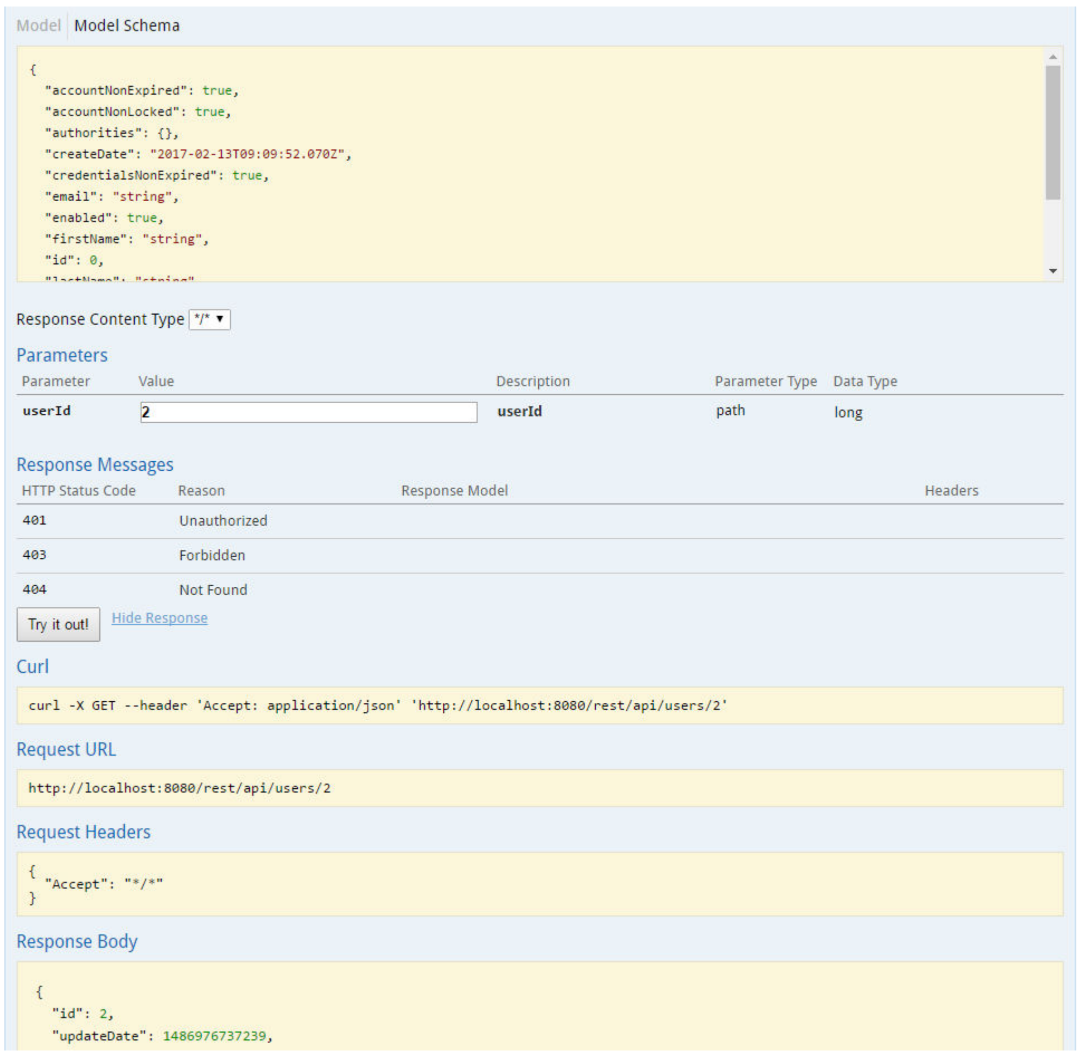
Api Documentation

Apache 2.0

user-resource : User Resource — Show/Hide | List Operations | Expand Operations

GET	/rest/api/users	getUsers
POST	/rest/api/users	createUser
GET	/rest/api/users/search	findUsers
GET	/rest/api/users/{userId}	getUser

[BASE URL: / , API VERSION: 1.0]



Model | Model Schema

```
{
  "accountNonExpired": true,
  "accountNonLocked": true,
  "authorities": {},
  "createDate": "2017-02-13T09:09:52.070Z",
  "credentialsNonExpired": true,
  "email": "string",
  "enabled": true,
  "firstName": "string",
  "id": 0,
```

Response Content Type */*

Parameters

Parameter	Value	Description	Parameter Type	Data Type
userId	2	userId	path	long

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

Try it out! Hide Response

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8080/rest/api/users/2'
```

Request URL

```
http://localhost:8080/rest/api/users/2
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

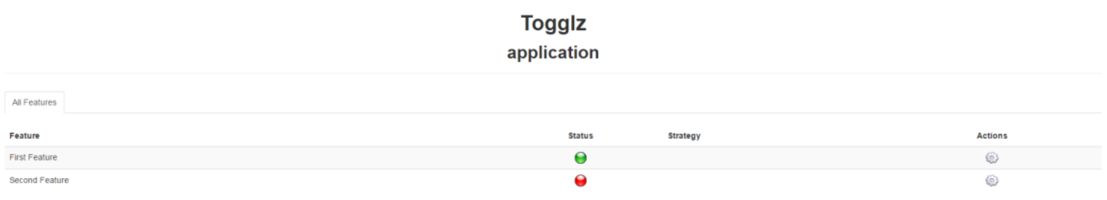
```
{
  "id": 2,
  "updateDate": 1486976737239,
```

Feature flags (Toggles)

To Add/remove feature flags see com.it.bo.server.enums.MyFeatures
To disable/enable features based on feature flags:

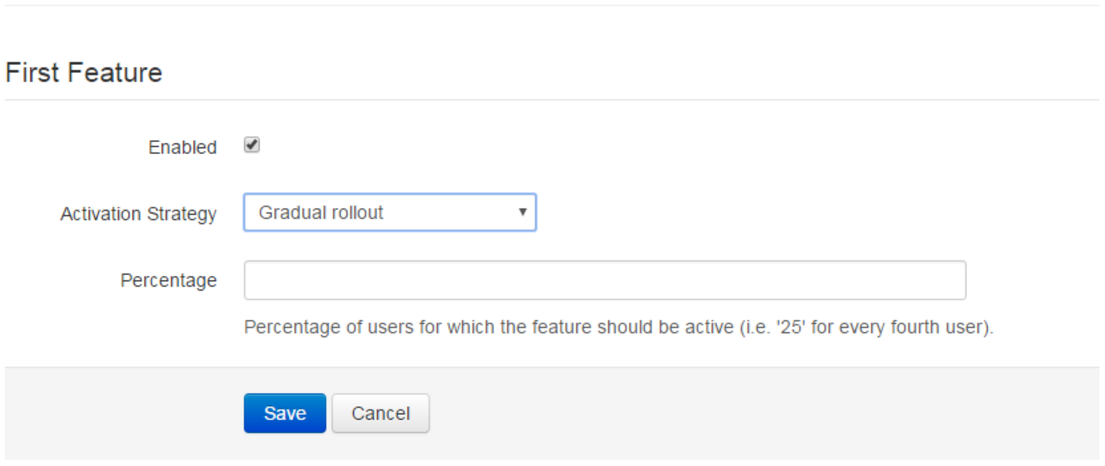
```
if (MyFeatures.<Feature name>.isActive()) {
        // put code here
}
```

Feature flags can be viewed and configured while server running using the following endpoints:
/togglz - Feature flag status
/togglz-console - Feature flag configuration


Togglz
application

All Features

Feature	Status	Strategy	Actions
First Feature	🟢		⚙
Second Feature	🔴		⚙


First Feature

Enabled	✓
Activation Strategy	Gradual rollout
Percentage	

Percentage of users for which the feature should be active (i.e. '25' for every fourth user).

Reference implementations

UserService

com.it.service.UserService – service which supports CRUD methods for user management
Used for both REST API and Back-office user management.

Rest

com.it.rest.UserResource supports rest CRUD operations for User

Get users:
GET: /rest/api/users
Returns json containing pageable list of users like:

```
{
        "content": [{
                "id": 1,
                "updateDate": 1486965599965,
                "createDate": 1486965599965,
                "username": "dev",
                "firstName": "Dev",
                "lastName": "user",
                "email": "dev@user.com",
                "password": "96e79218965eb72c92a549dd5a330112",
                "enabled": true,
                "authorities": [{
                        "authority": "ROLE_ADMIN"
                }],
                "accountNonExpired": true,
                "accountNonLocked": true,
                "credentialsNonExpired": true
        }],
        "totalPages": 1,
        "totalElements": 1,
        "last": true,
        "size": 2147483647,
        "number": 0,
        "sort": null,
        "first": true,
        "numberOfElements": 1
}
```

Page size is currently max integer but can be easily changed in UserResource.getUsers().

Get specific user details:

GET: /rest/api/users/{userId}
Returns details of the required user in json format:

```
{
	"id": 1,
	"updateDate": 1486965599965,
	"createDate": 1486965599965,
	"username": "dev",
	"firstName": "Dev",
	"lastName": "user",
	"email": "dev@user.com",
	"password": "96e79218965eb72c92a549dd5a330112",
	"enabled": true,
	"authorities": [{
		"authority": "ROLE_ADMIN"
	}],
	"accountNonExpired": true,
	"accountNonLocked": true,
	"credentialsNonExpired": true
}
```

Create new user:
POST: /rest/api/users
Expected input is json representing user.
For example:

```
{
	"username": "username",
	"firstName": "firstname",
	"lastName": "lastname",
	"email": "user@gmail.com",
	"password": "123456",
}
```

Search users:
GET: /rest/api/users/search?q=<search term>
Returns users matching search term.

Tests

Sample test can be found in /src/test/java/com/it/tests

com.it.tests.UserServiceTest provides reference implementation for test. Test will initialize Spring context and will run queries to DB with results validation

com.it.tests.BaseTest provides common test functionality and configurations

Back Office

com.it.bo.server.views.BaseView provides common logic for views (title etc)

For actual views extend baseView and implement buildViewContent().

com.it.bo.server.views.UserManagementView provides User management screen with few basic operations for users management (create new users, view existing etc)

com.it.bo.server.controllers.BaseController providers common logic for controllers
com.it.bo.server.controllers.UserManagemgentController controlling UserManagementView

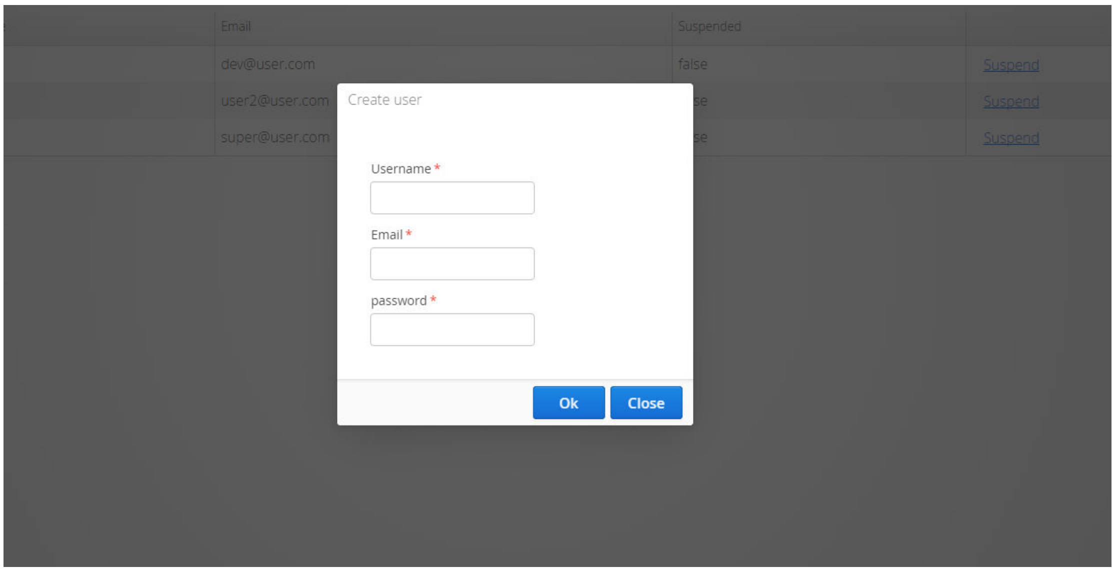
@ControllingView - annotation to provide link between view and controller. In order to link view to controller, only need to put this annotation on controller class. This will also provide navigation functionality

com.it.bo.server.views.AdminMenuItem enum for left-side menus. In order to support navigation to newly created view, it should be added to this enum.

com.it.bo.server.framework.ControllersFacade - infrastructure of View-Controller linkage, navigation and view initialization

UI components

com.it.bo.server.components.BaseWindow provider common functionality for modal windows
com.it.bo.server.components.AddEditUserWindow provides functionality to add\edit user
com.it.bo.server.components.CustomTable generic table implementation for quick table creation and easy population of new data. Contains also some basic filtering functionality



	Email	Suspended	
	dev@user.com	false	Suspend
	user2@user.com	se	Suspend
	super@user.com	se	Suspend

Create user

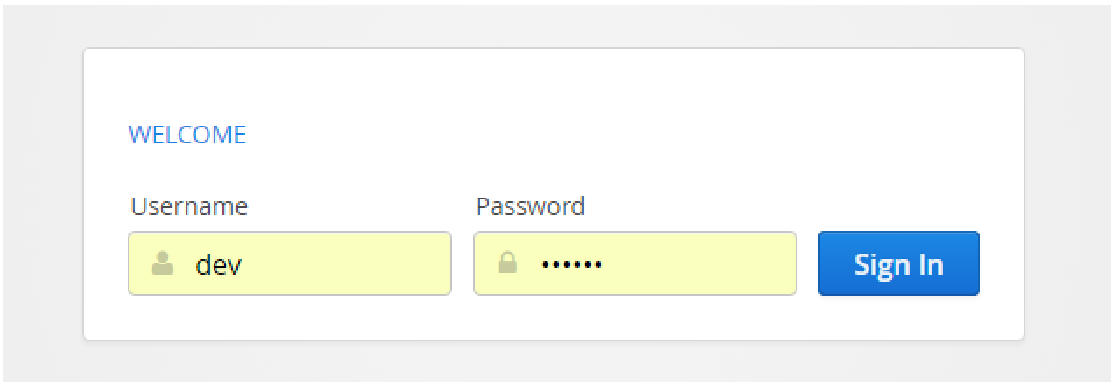
Username *

Email *

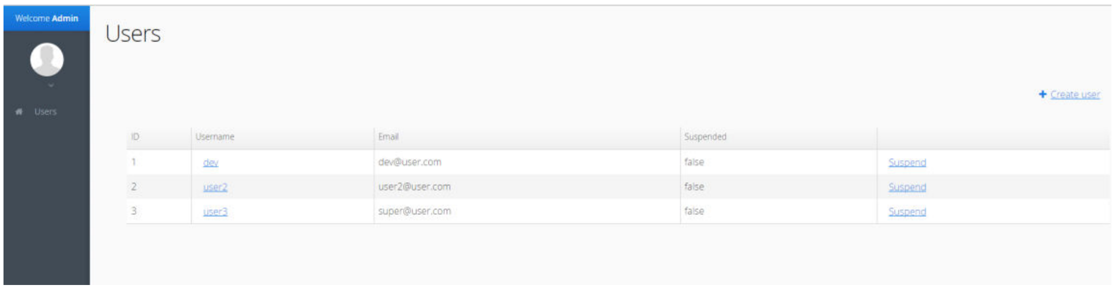
password *

Login View

com.it.bo.server.ui.LoginUI – This is the UI shown to user when not logged-in.
Defines the view of the login form and handling the login event.



Main View:
com.it.bo.server.ui.AdminUI – The UI user is redirected to after login.
Creates the navigator and the content of the default view.



Users

ID	Username	Email	Suspended	
1	dev	dev@user.com	false	Suspend
2	user2	user2@user.com	false	Suspend
3	user3	super@user.com	false	Suspend

Others

com.it.bo.server.servlet.AdminSessionInitListener in case some basic html customization needed (google analytics etc, it can be handled in this class)