**Software setup components library**

**Technical documentation**

**Ver. 1.1**

Table of Contents

[Version control 3](#_Toc474750285)

[Overview 4](#_Toc474750286)

[Project structure and goals 4](#_Toc474750287)

[Technologies used for this project 5](#_Toc474750288)

[Project components 6](#_Toc474750289)

[Configuration 6](#_Toc474750290)

[DB 7](#_Toc474750291)

[System health check and status 8](#_Toc474750292)

[Auto generated API documentation with web interface 10](#_Toc474750293)

[Feature flags (Toggles) 11](#_Toc474750294)

[Reference implementations 12](#_Toc474750295)

[UserService 12](#_Toc474750296)

[Rest 12](#_Toc474750297)

[Back Office 13](#_Toc474750298)

[UI components 14](#_Toc474750299)

[Login View 14](#_Toc474750300)

[Others 15](#_Toc474750301)

# 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



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

## System health check and status

Following endpoints provided to support system health check:

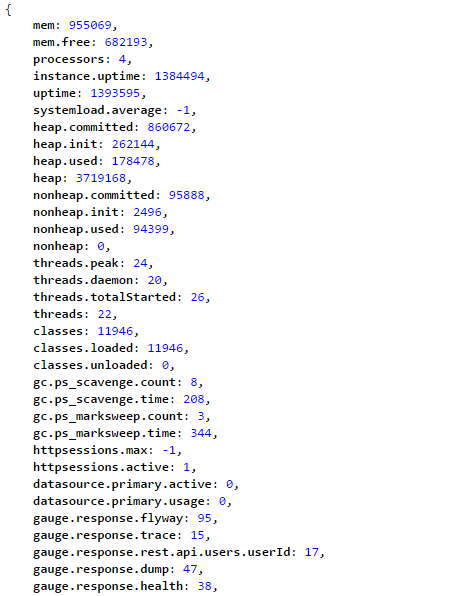
/health – Shows application health (shows that server is up, disk space, db connection status)

/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

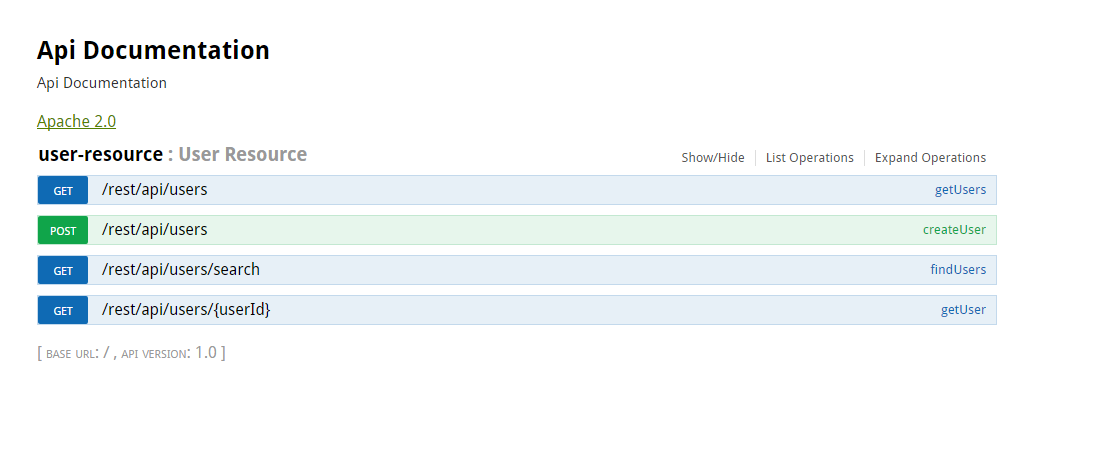


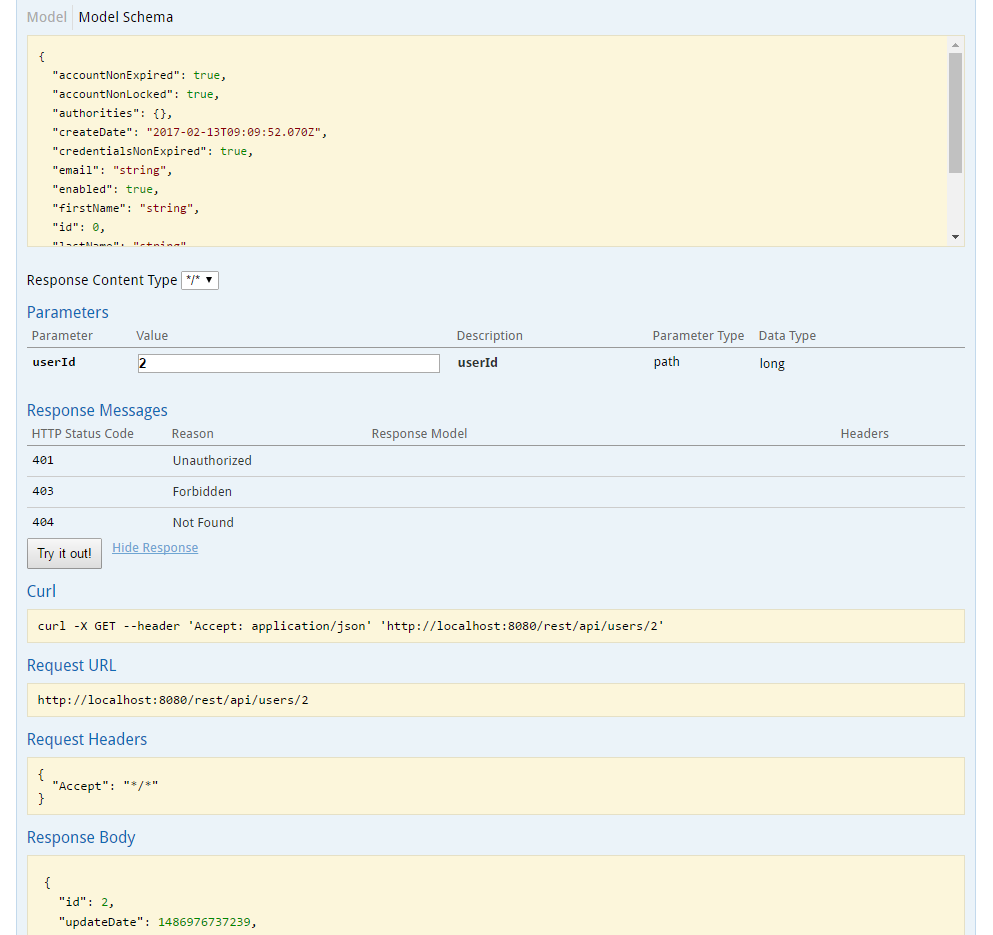


## Auto generated API documentation with web interface

/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





## 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()) {

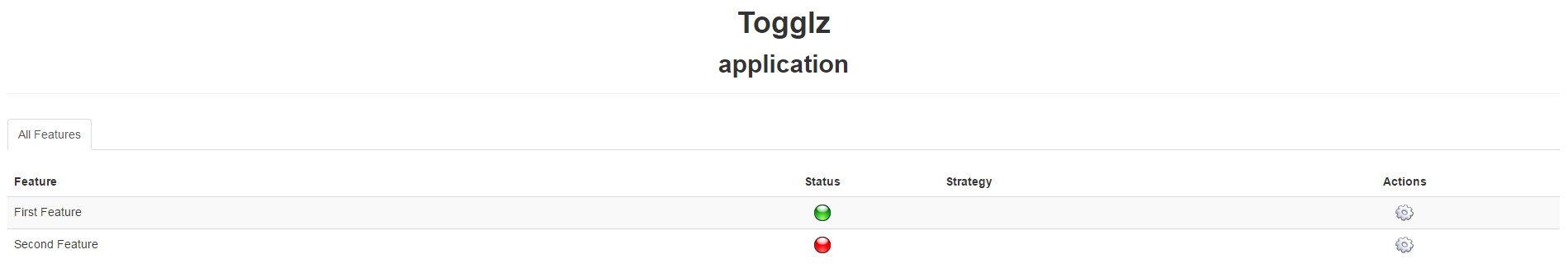
// put code here

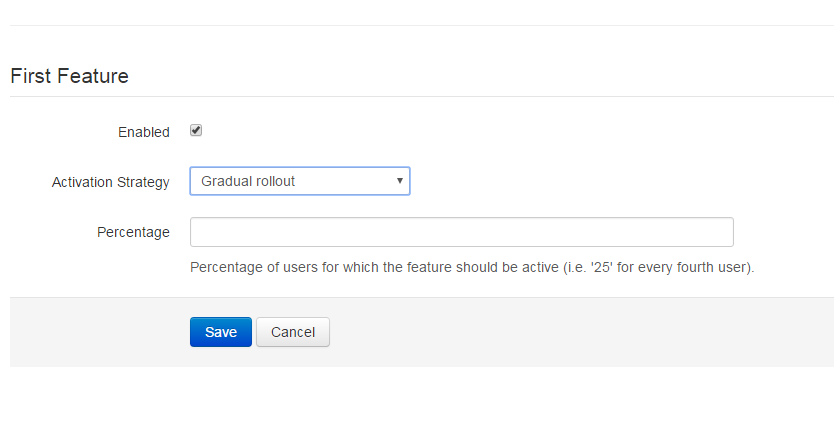
}

Feature flags can be viewed and configured while server running using the following endpoints:

/togglz - Feature flag status

/togglz-console - Feature flag configuration





# 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.

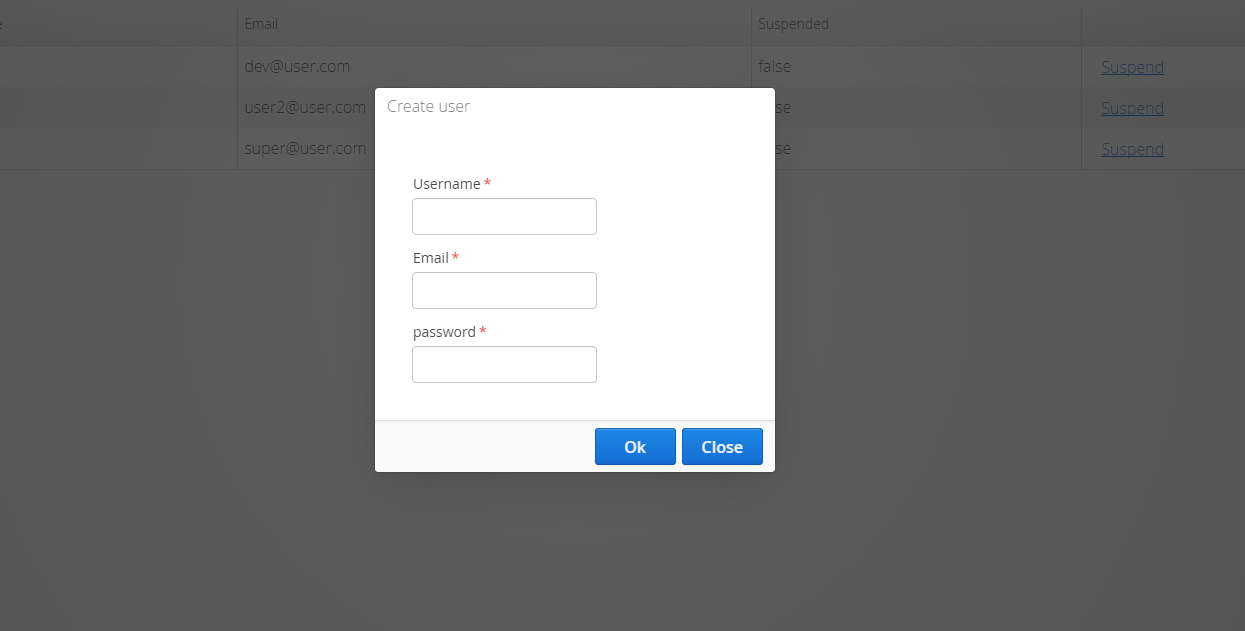
com.it.bo.server.framework.ControllersFacade - infrastructure of View-Controller linkage, navigation and view initialization

## 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

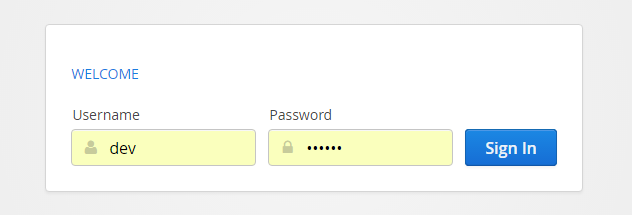
com.it.bo.server.components.CustomTable generic table implementation for quick table creation and easy population of new data. Contains also some basic filtering functionality



## Login View

com.it.bo.server.ui.LoginUI – This is the UI shown to user when not logged-in.

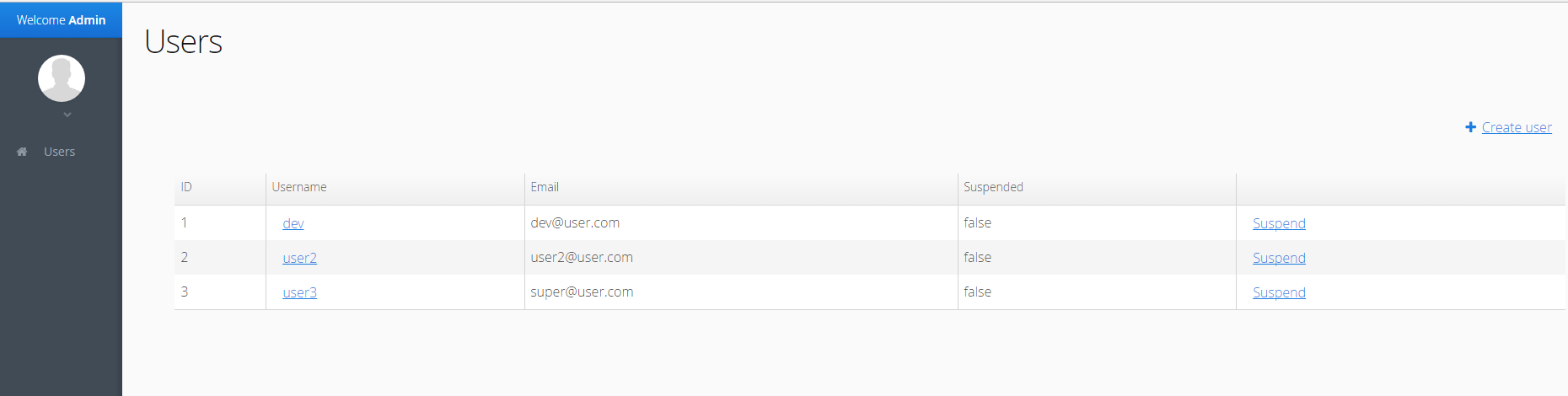
Defines the view of the login form and handling the login event.



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.



## Others

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