Documentation

Release 5.1.2.

Developer Guide XSQUARE- RAD 5.1.2

Feb. 11, 2025

Table of contents

CHAPTER 1

General information

This guide describes how to use the XRAD development environment to build web applications.

1.1 Who this guide is for

This guide is intended for developers whose goal is to build a data-centric (database-driven) web application using XSQUARE RAD development tools, hereinafter referred to as XRAD. The manual describes how to use the tools to build, debug, manage and deploy the application.

1.2 Requirements to the developer

Basic knowledge of relational database concepts, SQL, basic HTML and JavaScript (js) is required for application development.

1.3 Localization

All our products do support all PostgreSQL locales. Very important at the beginning create your PostgreSQL database with compatible localization to correct order of sorting rows, for this the LC_COLLATE and LC_CTYPE parameters are responsible. We recommend setting your locale as primary on the operating system so that PostgreSQL inherits the OS locale by default.

Attention! During installation, the default English locale is specified, it is highlighted in red font in the text. If you want to set a different locale, change the highlighted value to the desired locale.

Example:

```
echo "UTC" > /etc/timezone && ¥
```

CHAPTER 2

Quick start

This section introduces you to the general concept of XRAD. This section describes the main components and features of the development environment.

2.1 Introduction to XRAD

XRAD provides the developer with all the tools to build an application in a single, extensible platform based on PostgreSQL database server.

2.1.1 What is XRAD?

In today's world, developing a web application often requires multiple developers, each responsible for separate components of the system.

Typically, a web application is divided into the following components:

- user interface (frontend),
- component for interaction with the database (backend),
- a linking component that is responsible for various application logic (middleware).

All these components are interdependent on each other and, without clearly defined responsibility boundaries, can disrupt the logic of the application. These boundaries and areas of responsibility are defined by the application architect, a person who requires a high level of competence in understanding the work of each component. One should also not forget about the administration of such a system and maintaining its performance, which requires a system administrator.

Maintaining such a team noticeably increases the costs and decreases the speed of application development.

The XRAD platform offers a completely different approach to development.

XRAD is a rapid application development platform developed by XSQUARE LLC (XSQUARE Rapid Application Development).

XRAD is a declarative environment for developing and deploying database-centric web applications. Thanks to built-in features such as user interface themes, items of navigation controls, form handlers and flexible reports,

XRAD considerably accelerates the application development process.

2.1.2 How does XRAD work?

XRAD uses a standard 3-tier architecture in which requests are sent from the browser through the application server to the database. All processing, data manipulation and business logic is done in the database. After the database processes the code, the results are passed by the web server as a JSON structure, based on which the web application will render the page.

This architecture guarantees near-zero-latency data access, superior performance and horizontal scalability out-of-the-box.

CHAPTER 3

Quick installation

3.1 Quick installation on DEB-based OS

Below one can find the steps of the quick installation of XRAD+PGHS using Debian OS as an example. All commands must be run with root privileges

1. Create a directory for the distribution

```
mkdir /root/xsquare
```

Go to the directory

```
cd /root/xsquare
```

2. Download/receive the distribution to the created directory

```
wget https://lcdp.xsquare.dev/files/pghs/xsquare.lcdp.v5/xsquare.lcdp.5.0_latest_
↪ release.zip
```

3. Unzip distribution

```
apt -y install unzip
unzip xsquare.lcdp.5.0.0.0.0.0_release.zip
```

4. Go to the directory with the distribution files

```
cd xsquare.lcdp.5.0.0.0.0.0_release
```

5. Configure the time zone and OS localization

```
echo "UTC" > /etc/timezone && ¥
dpkg-reconfigure -f noninteractive tzdata && ¥
sed -i -e 's/# en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/' /etc/locale.gen && ¥
echo 'LANG="en_US.UTF-8"'>/etc/default/locale && ¥
```

(continued on next page)

(continued from previous page)

```
dpkg-reconfigure --frontend=noninteractive locales && ¥

export LANG=en_US.UTF-8
```

6. Install PostgreSQL

```
apt -y install postgresql
```

7. Prepare PostgreSQL

switch to the postgres user

```
su - postgres
```

create database users xrad_user and app_user

```
psql -c "create user xrad_user with encrypted password 'xrad_user';"
psql -c "create user app_user with encrypted password 'app_user';"
```

create appdb and xraddb databases

```
psql -c "CREATE DATABASE \"appdb\" WITH OWNER \"app_user\" ENCODING 'UTF8' LC_COLLATE␣
↪= 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';"
psql -c "CREATE DATABASE \"xraddb\" WITH OWNER \"xrad_user\" ENCODING 'UTF8' LC_
↪COLLATE = 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';"
```

assign maximum privileges to users xrad_user and app_user

```
psql -c "ALTER USER xrad_user WITH SUPERUSER;"
psql -c "ALTER USER app_user WITH SUPERUSER;"
```

log out of the postgres account session

```
exit
```

8. Import databases

```
export PGPASSWORD='xrad_user';
psql -U xrad_user -h 127.0.0.1 xraddb< db/xraddb.xsquare.pgsql

export PGPASSWORD='app_user';
psql -U app_user -h 127.0.0.1 appdb< db/appdb.xsquare.pgsql
```

9. Install nginx

```
apt -y install nginx
```

disable the default site

```
rm -f /etc/nginx/sites-enabled/default
```

copy the web controller files for PGHS and XRAD from the distribution kit

```
cp -R ./var /
```

copy nginx configuration files from the distribution kit

```
cp -R ./etc/nginx /etc/
```

10. Restart nginx

```
systemctl restart nginx
```

check its condition

```
systemctl status nginx
systemctl enable nginx
```

11. Copy executable and configuration files XRAD, PGHS

```
cp -R ./etc/systemd /etc/
cp -R ./usr  /
```

12. Start XRAD as the service and check the status

```
systemctl start xsquare.xrad.service
systemctl enable xsquare.xrad.service
systemctl status xsquare.xrad.service
```

13. Start PGHS as the service and check the status

```
systemctl start xsquare.pghs.service
systemctl enable xsquare.pghs.service
systemctl status xsquare.pghs.service
```

14. Check the availability of the default web application and XRAD builder in the browser

Note: in case of problems with http access one should check nginx settings and firewall permissions.

3.2 Quick installation on RPM-based OS

Below one can find the steps of the quick installation of XRAD+ PGHS using Fedora as an example. All commands should be executed with root privileges

1. Create a directory for the distribution

```
mkdir /root/xsquare
```

Go to the directory

```
cd /root/xsquare
```

2. Download the distribution to the created directory

```
wget https://lcdp.xsquare.dev/files/pghs/xsquare.lcdp.v5/xsquare.lcdp.5.0.0.0.0.0_
↪release.zip
```

3. Unzip distribution

```
dnf install -y unzip
unzip xsquare.lcdp.5.0.0.0.0.0_release.zip
```

4. Go to the directory with the distribution files

```
cd xsquare.lcdp.5.0.0.0.0.0_release
```

5. Configure the time zone and OS localization

```
timedatectl set-timezone UTC
localectl set-locale LANG=en_US.UTF-8
export LANG=en_US.UTF-8
```

6. Install and start PostgreSQL

```
dnf install -y postgresql
postgresql-setup --initdb
systemctl start postgresql
systemctl enable postgresql
```

7. Prepare PostgreSQL

switch to the postgres user

```
su - postgres
```

create database users xrad_user and app_user

```
psql -c "create user xrad_user with encrypted password 'xrad_user';"
psql -c "create user app_user with encrypted password 'app_user';"
```

create appdb and xraddb databases

```
psql -c "CREATE DATABASE \"appdb\" WITH OWNER \"app_user\" ENCODING 'UTF8' LC_COLLATE␣
↪= 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';"
psql -c "CREATE DATABASE \"xraddb\" WITH OWNER \"xrad_user\" ENCODING 'UTF8' LC_
↪COLLATE = 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';"
```

assign maximum privileges to users xrad_user and app_user

```
psql -c "ALTER USER xrad_user WITH SUPERUSER;"
psql -c "ALTER USER app_user WITH SUPERUSER;"
```

log out of the postgres account session

```
exit
```

8. Import databases

```
export PGPASSWORD='xrad_user';
psql -U xrad_user -h 127.0.0.1 xraddb < db/xraddb.xsquare.pgsql

export PGPASSWORD='app_user';
psql -U app_user -h 127.0.0.1 appdb < db/appdb.xsquare.pgsql
```

9. Install nginx

```
dnf install -y nginx
```

disable the default site

```
rm -f /etc/nginx/sites-enabled/default
```

copy the web controller files for PGHS and XRAD from the distribution kit

```
cp -R ./var /
```

copy nginx configuration files from the distribution kit

```
cp -R ./etc/nginx /etc/
```

10. Restart nginx

```
systemctl restart nginx
```

check its condition

```
systemctl --no-pager status nginx
```

11. Copy executable and configuration files XRAD, PGHS

```
cp -R ./etc/systemd /etc/
cp -R ./usr /
```

12. Start XRAD as the service and check the status

```
systemctl start xsquare.xrad.service
systemctl enable xsquare.xrad.service
systemctl --no-pager status xsquare.xrad.service
```

13. Start PGHS as the service and check the status

```
systemctl start xsquare.pghs.service
systemctl enable xsquare.pghs.service
systemctl --no-pager status xsquare.pghs.service
```

14. Check the availability of the default web application and XRAD builder in the browser

Note: in case of problems with http access you should check nginx settings and firewall permissions.

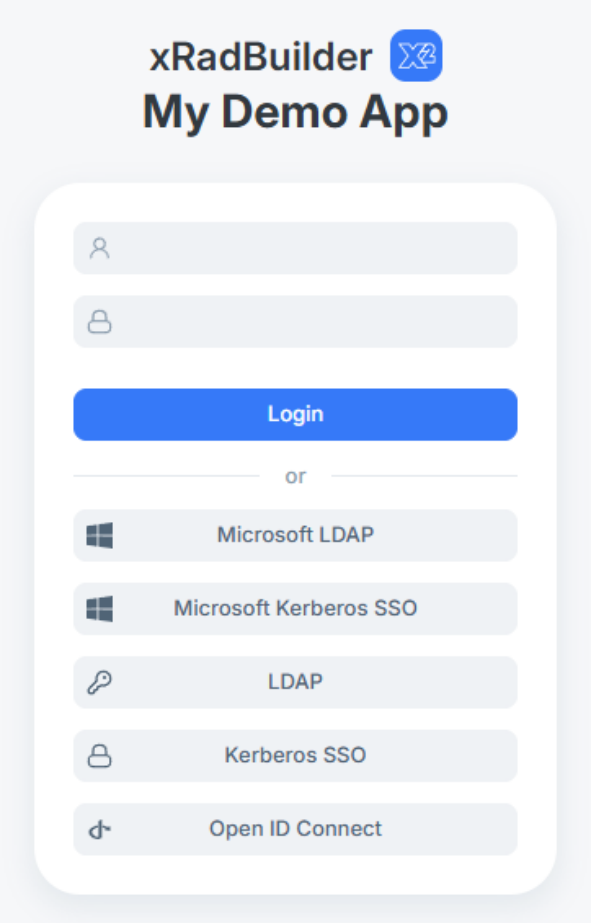
3.3 Introduction to the development environment

The XRAD development environment provides the developer with a flexible and intuitive interface for developing and managing the application.

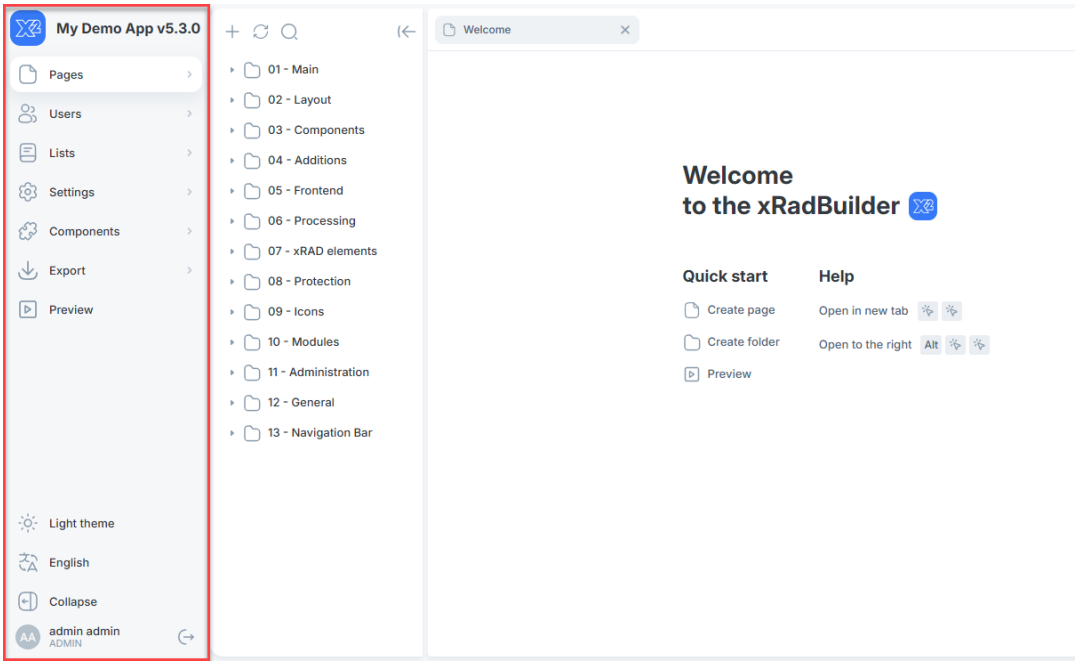
Getting started with the environment begins with the user authentication process on the page:

http://hostname:8080

Where hostname is the name of the host with XRAD installed.



The main page of the development environment opens after successful authentication. The development environment interface can be divided into 3 areas (from left to right):



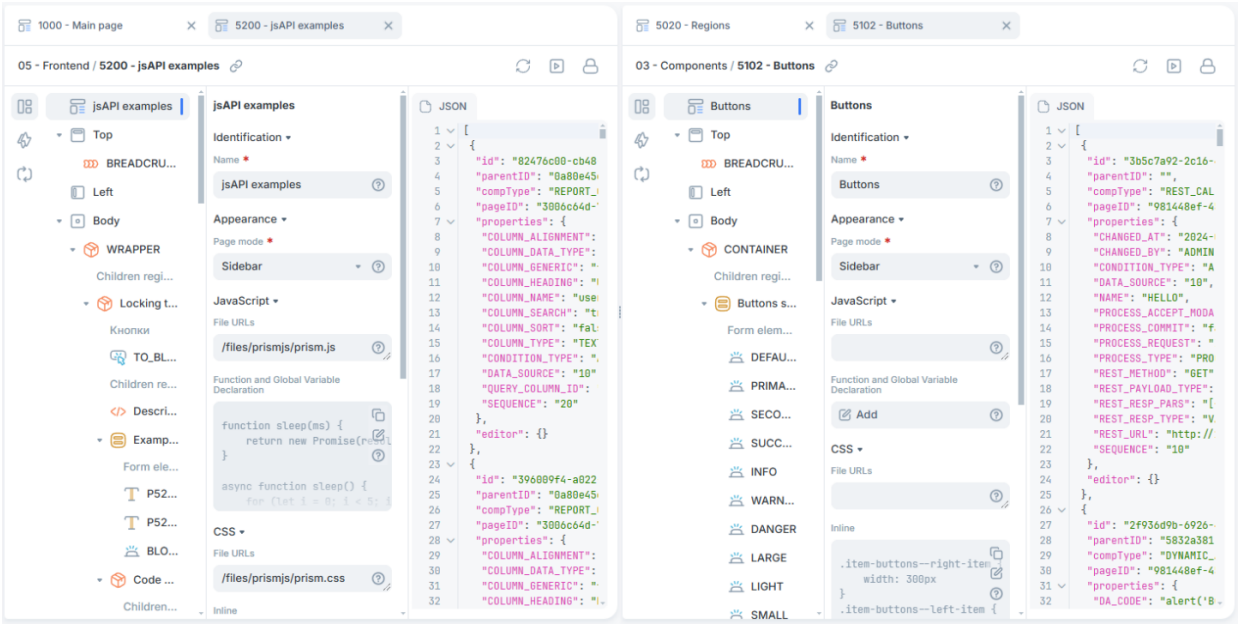
1. Main menu - contains the key elements of the application being developed, as well as information about the current user and the interface language.
2. List of items - this area displays the data sets for each main menu item, as well as buttons for creating new items, updating the data set and context search on the data set. This area supports a contextual menu accessible by right click.
3. Editor - here one can see the tabs where one can create and edit the selected elements.

The main menu contains the following sections:

- Pages - a list of pages in the web application.
- Users - a list of users of the development environment.
- Lists - predefined lists used for the operation of application components and navigation.
- Settings - settings of various parameters of the application.
- Preview – a button to switch to viewing the application in a new window.

In the main menu one can also change the interface language and end the current user's session.

The development environment supports drag&drop, and the editor supports split-window mode to speed up development.



The following chapters will provide a detailed description of the development environment and its components.

CHAPTER 4

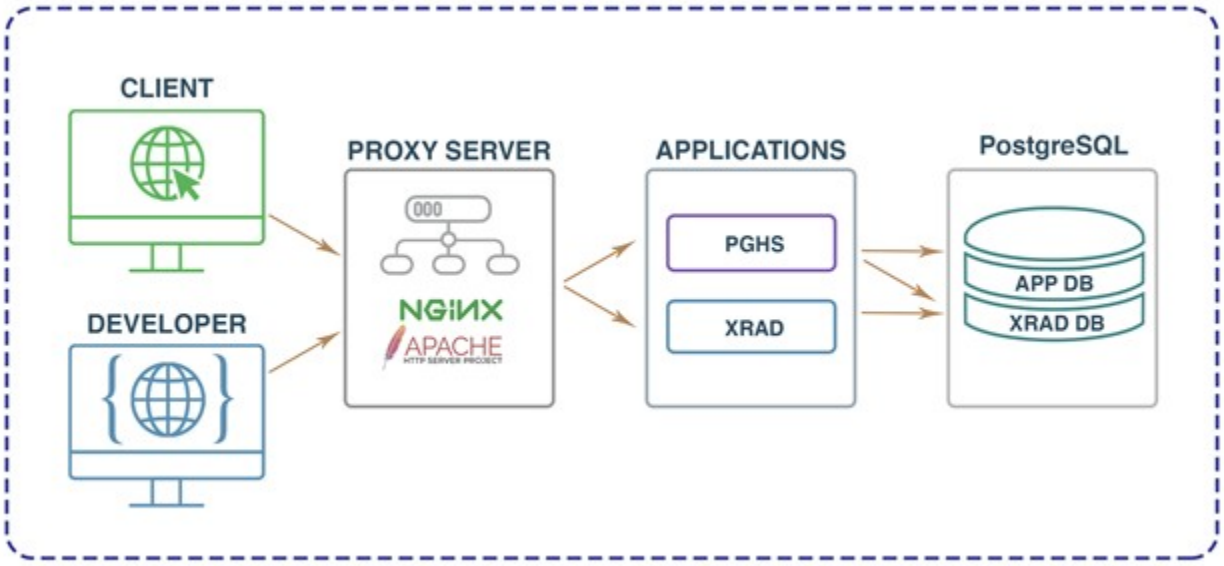
Architecture and system requirements

4.1 Architecture

The basic architecture of XRAD consists of 4 components:

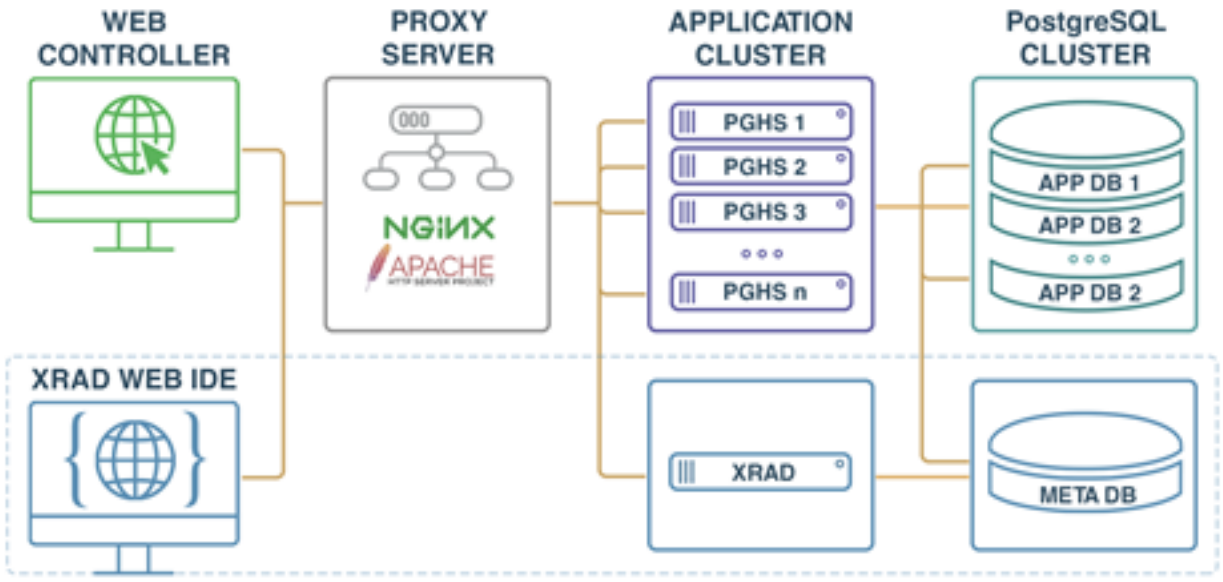
- XRAD database is a database with application metadata. It stores information about the components of the requested page, processes that should occur during page processing.
- The XRAD application server, which is responsible for processing developer actions and preparing the page structure for rendering.
- The XRAD web controller, which processes the page structure received from the web server and displays it.
- HTTP Proxy Server, which links the web server and the XRAD web controller.

The PGHS application server is required to display the web application under development, so XRAD and PGHS work in conjunction and share the same 4 component architecture. Whether you are running an XRAD development environment or an application created using XRAD, the process is the same. Your browser sends a request, which is converted into a corresponding code call on the database side. After the database processes the code, the results are passed back to your browser in the form of a JSON structure, based on which the web application will generate a page. This cycle happens every time you request or submit a page.



All components can be deployed either within a single server or distributed across different physical or virtual servers.

Horizontal scaling can be easily performed for highly loaded systems. An example of a highly loaded architecture is as follows:



4.2 System requirements

4.2.1 Performance environment

Supported architecture:

- x86-64

- ARM
- Loongson

Supported OS:

- **DEB**-based - any
- **RPM**-based - any
- Debian 12 - recommended

Database:

- PostgreSQL 13+
- PostgreSQL 15 - recommended

HTTP/Proxy Server:

- Apache 2.4+
- NGINX 19+

4.2.2 System requirements

XRAD - Server:

- CPU - 1 Core
- RAM - 100 MB
- HDD - 100 MB+ Logs

XRAD DB:

- CPU - 1 Core
- RAM - 50 MB
- HDD - 10 Mb PostgreSQL database

Installation of virtualization/containerization system, operating system, database is carried out at the discretion of the Administrator if needed.

CHAPTER 5

Installation and customization

5.1 XRAD installation

Detailed OS configuration and installation of all components except XRAD are covered in the PGHS documentation. This documentation covers the actual deployment of XRAD.

Note: the examples are given for the case when the user is in the distribution directory and all actions are performed with root privileges.

To install XRAD, copy the executables from the distribution to the /usr/local/xsquare.xrad directory

For example:

1. Copy all the components

```
cp -R ./usr /
```

or 2. Copy the XRAD distribution directly

```
cp -R ./usr/local/xsquare.xrad /usr/local/
```

Note: assign execution rights

```
chmod +x /usr/local/xsquare.xrad/xrad
```

Creating a service

```
vi /etc/systemd/system/xsquare.xrad.service

[Unit]
Description=XRAD Services
After=syslog.target network.target
After=postgresql.service

[Service]
```

(continued on next page)

(continued from previous page)

```
Type=simple
ExecStart=/usr/local/xsquare.xrad/xrad
WorkingDirectory=/usr/local/xsquare.xrad
Restart=on-failure
RestartSec=3

[Install]
WantedBy=default.target
```

Note: The default service file can be copied from the distribution. For example:

```
cp -R ./etc/systemd /etc/
```

Start XRAD as the service and check the status

```
systemctl start xsquare.xrad.service
systemctl enable xsquare.xrad.service
systemctl --no-pager status xsquare.xrad.service
```

Next, configure HTTP proxy server using NGINX and Apache2

Copy the XRAD web controller files from the distribution kit.

```
cp -R ./var/www/xrad.xsquare/* /var/www/xrad.xsquare/
```

5.2 Configuring NGINX

Copy nginx configuration files from the distribution kit

```
cp -R ./etc/nginx /etc/
```

and edit file /etc/nginx/conf.d/xrad.xsquare.conf, making necessary changes
vi /etc/nginx/conf.d/xrad.xsquare.conf

```
server {
    listen 8080;
    server_name xrad.xsquare;
    root /var/www/xrad.xsquare;
    index index.html;
    location /ds{
            proxy_pass http://127.0.0.1:8889/ds;
    }
    location / {
        try_files $uri $uri/ =404;
    }
}
```

To apply the new settings, restart nginx

```
systemctl restart nginx
systemctl enable  nginx
```

check its condition

```
systemctl --no-pager status nginx
```

5.3 Configuring Apache2 on DEB-based OS

Copy apache2 configuration files from the distribution kit

```
cp -R ./etc/apache2 /etc/
```

and edit the VirtualHost configuration file /etc/apache2/sites-available/xrad.xsquare.conf, making the necessary changes:

```
 vi /etc/apache2/sites-available/xrad.xsquare.conf

 <VirtualHost *:80>
ServerAdmininfo@xsquare.dev
ServerName   xrad.xsquare
ServerAlias xrad.xsquare
DocumentRoot /var/www/xrad.xsquare

Alias /files "/var/www/xrad.xsquare.files.local"
<Directory
        /var/www/xrad.xsquare.xsquare.files.local>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>.
ProxyPass         /ds http://127.0.0.1:8889/ds
ProxyPassReverse /ds http://127.0.0.1:8889/ds

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
 </VirtualHost>.
```

Apply the new configuration

```
a2ensite xrad.xsquare.conf
```

and restart apache2

```
systemctl restart apache2
```

5.4 Configuring Apache2 on RPM-based OS

Copy httpd configuration files from the distribution kit

```
cp -R ./etc/httpd /etc/
```

and edit the VirtualHost configuration file /etc/httpd/conf.d/xrad.xsquare.conf, making the necessary changes:

```
 vi /etc/httpd/conf.d/xrad.xsquare.conf
 <VirtualHost *:80>
ServerAdmin info@xsquare.dev
ServerName   xrad.xsquare
ServerAlias xrad.xsquare
DocumentRoot /var/www/xrad.xsquare

Alias /files "/var/www/xrad.xsquare.files.local"
<Directory
        /var/www/xrad.xsquare.files.local>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>
ProxyPass         /ds http://127.0.0.1:8889/ds
ProxyPassReverse /ds http://127.0.0.1:8889/ds

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
 </VirtualHost>
```

Disable Security-Enhanced Linux for HTTP requests

```
setsebool -P httpd_can_network_connect 1
```

and restart apache

```
systemctl restart httpd
```

5.5 Customizing XRAD

To configure XRAD, edit the config.json file:

```
    vi /usr/local/xsquare.xrad/config.json
    {
"app": {
    "port": "8889"
},
"XRAD": {
    "login": "xrad_user",
    "password": "xrad_user",
    "host": "localhost",
    "port": 5432,
    "minCons": 1,
    "maxCons": 15,
    "dbName":"xraddb",
    "runtimeOptions":{
        "LC_NUMERIC":"en_US.UTF-8"
    }
},
"datasources": {
    "login": "app_user",
```

(continued on next page)

(continued from previous page)

```
    "password": "app_user",
    "host": "localhost",
    "port": 5432,
    "minCons": 1,
    "maxCons": 15,
    "dbName":"appdb",
    "runtimeOptions":{
        "LC_NUMERIC":"en_US.UTF-8"
    }
}
    }
```

CHAPTER 6

Configuration files

6.1 Configuration file .json

For XRAD to operate, the json configuration file must be present in the directory with the executable file.

The configuration file contains 3 sections: The "app" descriptor, where you can define the basic server settings:

```
{
"app": {
"port": "8889"
},
```

"port" - string. Defines the number of the network port on which the server will be started (by default - 8889)

Descriptor "XRAD", where the settings for working with the XRAD database are defined:

- "login" - string. Username to connect to the XRAD database.
- "password" - string. User password for connection to XRAD database.
- "host" - string. IP address of the XRAD database server.
- "port" - number. The number of the port on which the XRAD database server is running.
- "dbName" - string. The name of the database to which XRAD needs to connect.
- "minCons" - number. Minimum number of simultaneous connections to the database.
- "maxCons" - number. Maximum number of simultaneous connections to the database.

runtimeOptions descriptor contains local settings:

- "LC_NUMERIC" - string. Local settings of the numeric format used to work with the database.

The "datasources" descriptor defines an array of data sources used by the application server. The data source description block contains the same set of fields as the XRAD database description and an additional field:

- "name" - string. The name of the data source.

For example, the following block defines two data sources named DEFAULT_APP and DEFAULT_APP_TEST

```
{ "datasources":
        [
    {
        "login": app_user",
        "password": "app_user",
        "host": "10.100.117.219",
        "name":"DEFAULT_APP",
        { "port": 5432,
    "minCons":1,
    "maxCons":15,
        "dbName": "pghs",
        "runtimeOptions":{
            "LC_NUMERIC":"en_US.UTF-8"
        }
    },
    {
        "login": app_user",
        "password": "app_user",
        "host": "10.100.117.219",
        "name":"DEFAULT_APP_TEST",
        { "port": 5432,
        "minCons":1,
        "maxCons":15,
        "dbName": "pghs",
        "runtimeOptions":{
            "LC_NUMERIC":"en_US.UTF-8"
        }
    }
    ]
```

6.2 Authentication scheme configuration file auth_config.json

XRAD supports authentication and authorization using the following schemes:

- Microsoft LDAP,
- Microsoft Kerberos SSO
- LDAP
- Kerberos SSO
- Open ID Connect

At startup, the XRAD development server loads schemes from the auth_config.json file and uses them when authenticating developers.

The auth_config.json file contains an array of authentication scheme descriptors in the following format:

```
[
    {
        "name": "",
```

(continued on next page)

(continued from previous page)

```
        "label": "",
        "type": "",
        "enabled": ,
        "order": ,
"options": {}
   }
```

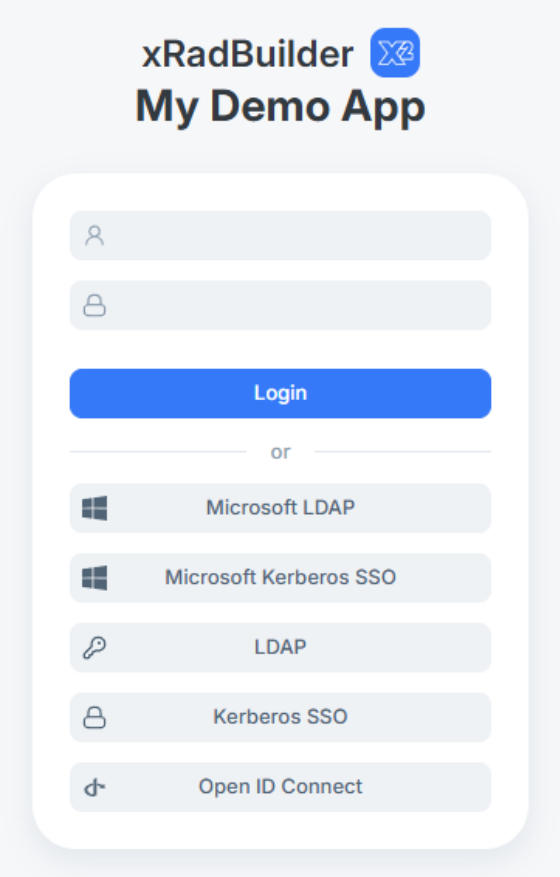
- "name" - string. The name of the authentication scheme.
- "label" - string. The name of the authentication scheme displayed on the button.
- "type" - string. The type of the authentication scheme.

Supported values:

- ldap
- kerberos_sso
- microsoft_ldap
- microsoft_kerberos_sso
- oidc

- "enabled" - Boolean value. This flag determines whether the authentication scheme is available for use in the application.
- "order" - number. The order number in the general list of schemes when displayed on the authentication page.
- "options" is a block of circuit parameters that depends on the type of circuit.

The schemes loaded from the configuration file will be used on the XRAD authorization page according to the enabled flag and the order number.



Below we review the *options* block for different authentication schemes.

6.2.1 LDAP, MICROSOFT_LDAP

For the ldap and microsoft_ldap scheme type, the "options" block describes the LDAP server connection parameters that are used to authenticate and retrieve user information.

- "host" - string. IP address of the LDAP server.
- "port" - number. The port used to communicate with the LDAP server.
- "base" - string. It defines the base entry point to the LDAP directory.
- "encryption" - string. Specifies the type of encryption used to communicate with the LDAP server. Supported values:
  - start_tls - establishes a secure TLS connection after initial authentication over an non-encrypted channel. Typically used on port 389.
  - simple_tls - establishes a fully encrypted TLS connection from the beginning. Usually used on port 636.
  - plain - does not use encryption and works over an unprotected channel. Usually used on port 389.
- "bind_dn" - string. Distinguished Name (DN) of the user used for authentication on the LDAP server.
- "password" - string. The password associated with the specified bind_dn.

- "request_user_groups" - Boolean value. It defines the need to request the groups to which the user belongs.

For example:

```
{
       "name": "User Auth LDAP",
       "type": "ldap",
       "options": {
           "host": "10.100.117.229",
           "port": 389,
           "base": "DC=ald,DC=dom",
           "encryption": "start_tls",
           "bind_dn": "uid=ldap,cn=users,cn=accounts,dc=ald,dc=dom",
           "password": "password",
           "request_user_groups": true
       }
}
```

6.2.2 KERBEROS_SSO, MICROSOFT_KERBEROS_SSO

For the kerberos_sso and microsoft_kerberos_sso scheme types, the "options" block describes the LDAP server connection parameters that are used to authenticate and retrieve user information.

- "keytab" - string. BASE64 encoded content of keytab file generated for end-to-end domain authentication.
- "request_user_groups" - Boolean value. It defines the need to request the groups to which the user belongs.

For example:

```
{
     "name": "User Auth Kerberos SSO",
     "type": "kerberos_sso",
     "options": {
         "keytab": "BQIAAABVAAIABOFMRC5ETOOABEhUVFAAEXBnaHMOLWRldi5hbG",
         "request_user_groups": true
     }
},
```

6.2.3 OIDC

For the oidc (Open Id Connect) scheme type, the "options" block describes the configuration of user authentication and authorization using OpenID Connect (OIDC).

- "scope"-string. A list of requested scopes that need to be accessed during authentication.
- "issuer"-string. The URL of the authentication server (Issuer).
- "uid_field" - string. It defines a field containing a unique user identifier (UID).
- "pkce" - Boolean value. The flag indicating the need to use Proof Key for Code Exchange (PKCE) to enhance security.
- "client_options" - descriptor of the block of parameters of the client using OIDC authentication.
- "id"-string. Client ID.

- "secret" - string. Secret client key (Client Secret).
- "redirect_uri"-string. The URL to which the user will be redirected after successful authentication.
- "request_user_groups" - Boolean value. It defines the need to request the groups to which the user belongs.

CHAPTER 7

Application development

7.1 Basic concepts

To effectively work and use XRAD, developers must understand these key concepts:

- How user interface design is managed,
- how pages are processed and rendered,
- what sessions and transactions are and how to work with them.

7.1.1 Application concept

What is an application?

A web application that is developed with XRAD is an HTML interface that exists on top of database objects: tables and procedures. The application is logically and physically divided into two databases: the database responsible for the business logic (APP_DB) and the database with the set of metadata of the application interface (XRAD_DB). PGHS Application Server combines the data from the two databases and provides an end-user web interface that displays business data and processes as described by the developer. One PGHS instance can access an unlimited number of APP_DB databases and only one XRAD_DB database.

For the end user, an application is a set of pages that display and allow data entry through various components.

What is a page?

A page is the "building block" of an application. An application may contain only one, but usually many such blocks. Each page can contain many simple elements, such as buttons and input fields, as well as more complex elements, such as various reports, graphs, tables. Elements on a page are grouped using a special container - region. Pages can include the logic of data processing, called processes. Links or branches are used to connect pages with each other.

7.1.2 Page rendering and input processing

To view the pages of an application under development, one must call the XRAD application page from the link that hosts the application. When you launch the application, XRAD invokes two key processes:

showPage is a page rendering process that collects all page attributes (including regions, buttons, and elements) into a single json control file. The runtime environment then passes this control structure to the client's web browser, which renders the described application elements. The XRAD client runtime is responsible for rendering the components on the client. When you request a page by clicking on a link, XRAD calls the showPage process.

processPage - page processing. It is responsible for processing the entered data, including the execution of processes, data checks and transitions to other pages. The process is called after sending the page to the server, by pressing a button or calling the appropriate method jsAPI. After submission, all data specified in the page elements are written to the session and substituted into the corresponding processes.

Transactions

When the showPage or processPage processes are called, a transaction in the database with business data is initiated. This transaction exists for the entire duration of execution of the showPage or processPage processes. If **all** processes are successfully executed, the transaction is terminated by the commit call, except for the case when the **TX commit** attribute is set at the process level. If an error occurs in the data processing processes - the transaction will be terminated by a rollback call, which will undo all changes at the moment of the process execution start. In this regard, the developer should take into account that the data that will be transferred to the database through the execution of one or another process will be available only at the successful completion of the transaction. The availability of data at the moment of executing this or that procedure completely depends on the transaction isolation level configured in the database with business data. By default, the isolation level in PostgreSQL is set to Read committed.

Due to limitations related to subtransactions in PostgreSQL, calling commit in a business process will result in an error. To ensure the execution of a separate process regardless of the success of the whole page processing, it is suggested to specify in the process settings to call commit after its execution.

The session values passed during the call of the showPage or processPage processes will also be written to the database only after the page processing is successfully executed. However, their values are available during the execution of these processes. To pass the values of input elements to business processes, see Page Processing.

7.1.3 Sessions and their status

What is a session

A session is a period of user interaction with an application. It starts when the user accesses the application and ends when the user closes the session, logs out or a certain period of inactivity passes (timeout). During a session, the application stores information about the user and their actions, which allows to maintain the context of interaction and personalize the information displayed.

Each session is assigned a unique identifier. XRAD uses this identifier to store and retrieve a working set of application data before and after each page view. Because sessions are completely independent of each other, any number of sessions can exist in the database at the same time. A user can also run multiple instances of an application simultaneously in different browsers.

The value of the session ID is stored in a browser cookie. The lifetime of this cookie is determined by the application settings and is additionally controlled by the PGHS runtime environment.

Sessions are logically and physically different from the database sessions used to serve page requests. A user runs an application in a single XRAD session from login to logout with a typical duration measured in minutes or hours. Each page requested during a session can initiate the creation of a new database session or use reopened database sessions to access database resources.

These database sessions often last only a fraction of a second.

What is a session state

Session state is a mechanism that allows developers to store and retrieve values for a user even as the user navigates through different pages of the application.

Hypertext Transfer Protocol (HTTP) - the protocol most commonly used to deliver HTML pages, is a stateful protocol. The web browser connects to the server only for the time it takes to load a complete page. Each page request is handled by the server as an independent event, unrelated to any page requests that have occurred previously or may occur in the future. In order to access form values entered on one page on the next page, the values must be saved to the session state. XRAD provides developers with the ability to retrieve and set session state values from any page in the application.

7.1.4 Managing session values

When creating interactive, data-driven web applications, the ability to access and manage session state values is critical. In XRAD, session state is automatically managed for each page and is easily referenced in static HTML or logical controls such as processes or checks.

7.2 Settings

The "Settings" section of the development environment contains:

1. Main - the main settings of the application being developed.
2. Data sources - list of databases that can be used as data sources for the application.
3. Global variables - list of global variables.
4. Global Processes - list of global processes of the application.
5. Authorization schemes - list of schemes available for user authorization during development.
6. Authentication schemes - list of schemes available for user authentication during development.

7.2.1 The main settings

The main application-level settings are mandatory and are divided into the following:

- Basic,
- Lists,
- Session,
- Static resources.

Parameter	Description
App name	Application name. Displayed in the page title.
Short name	The name of the application displayed in the development environment.
App URL	URL of the application.
Home page	The main page that an authorized user lands on. In the input field one needs to enter, or select from the list, the page number to be used as the homepage.
Login page	The main public page that an unauthorized user lands on. Typically, the authorization page. In the input field one needs to enter, or select from the List, the page number to be used as the login page.

Lists

This parameter block defines the application-level lists.

Parameter	Description
Navigation bar	Defines a list for the menu in the page header. Select from the pre-created lists in the appropriate section.
Navigation menu	Defines a list for the main menu in the left block of the page Select from the pre-created lists in the appropriate section.

Session

This block defines the parameters of user sessions.

Parameter	Description
Session lifetime	Defines the lifetime of a user session, i.e. the period of time, during which a user session of interaction with the application is active.
Session lifetime (units of measure)	Defines the unit of measure for the number specified in the "Session lifetime" parameter. There are 4 options to choose from: seconds, minutes, hours and days.
Inactive session lifetime	Specifies the period, in units of time, after which the user session will be deleted if the user does not perform any actions in the application.
Inactive session lifetime (units of measure)	Defines the unit of measure for the number specified in the "Session lifetime" parameter. There are 4 options to choose from: seconds, minutes, hours and days.

Static resources

This block defines external static resources that allow changing the display of the application user interface.

Parameter	Description
CSS files	A list of .css files containing style sheets. Each line defines a relative or absolute path to an internal or external resource.
JS files	A list of .js files containing JavaScript code. Each line defines a relative or absolute path to an internal or external resource.

7.2.2 Data sources

Data sources are one of the most important components of XRAD, which provides an opportunity to scale the developed application horizontally. The application allows creating an unlimited number of data sources. The data source is a database, the data of which is used to display the application components. By specifying different data sources, the developer can combine data output from different databases on one page.

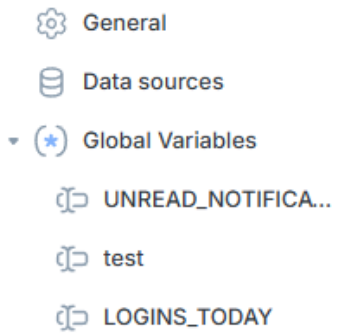
To add a new data source, specify it in the application settings. In the settings, only the name of the source is specified, which will be used to insert it into the corresponding fields of the page editor components.

The data source connection settings are specified in the json PGHS and XRAD configuration files, with the source name in the configuration file matching the one entered in the application settings.

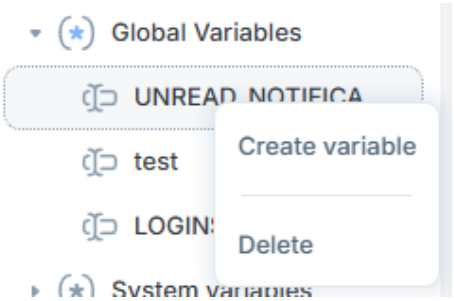
7.2.3 Global variables

This settings block defines the global variables of the application.

Global variables are variables that are available and ubiquitous in an application to store some value.

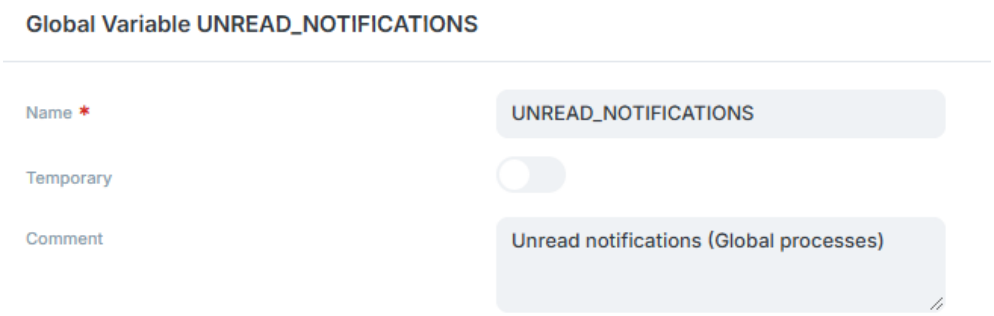


A new global variable can be declared by clicking on the "Global " list or on any element of this list.



When you create a global variable, you must fill in its attributes:

Parameter	Description
Name	Is the only mandatory attribute that defines the name of the variable to be created.
Temporary	The flag that determines whether the variable will be temporary. A temporary variable is not stored in the session state and accessed only while processes are running.
Comment	Text comment



After clicking the "Save" button, the variable will be entered into the XRAD database and will be available for viewing, editing and deleting. Existing variables are edited in the same way.

To delete a global variable, one should right click the necessary variable in the list and select "Delete" from the menu.

Preset variables

Preset variables are variables that are present in every XRAD application. These variables cannot be edited or deleted.

Reserved global variables:

Temporary:

- **REQUEST** - the request code to execute the required process,
- **PAGE** - page number,
- **RESPONSE** - the output parameter of the executed Ajax process,
- **G01** - **G10** - built-in variables for passing values to processes. Can be used when calling processes with js.
- **CGI_FORWARDED_FOR** is a variable that stores the value of the X-Forwarded-For header sent by the web server.
- **CGI_FORWARDED_IP** is a variable that stores the value of the X-Forwarded-IP header passed by the web server.
- **CGI_REAL_IP** - A variable that stores the value of the X-Real-IP header passed by the web server.

Constants:

- **APP_USER** - id of the currently authorized user,
- **SESSION** - unique session code,
- **APP_USER_GROUPS** - domain groups of the currently authorized user,

7.2.4 Global processes

This settings block defines the global processes for the entire application.

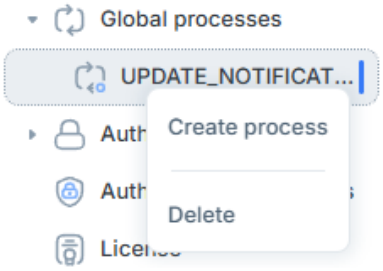
An application process is an XRAD process that is not bound to a page. Such processes, as well as processes on pages, are executed upon request from the application (Request).

The order in which application processes are executed

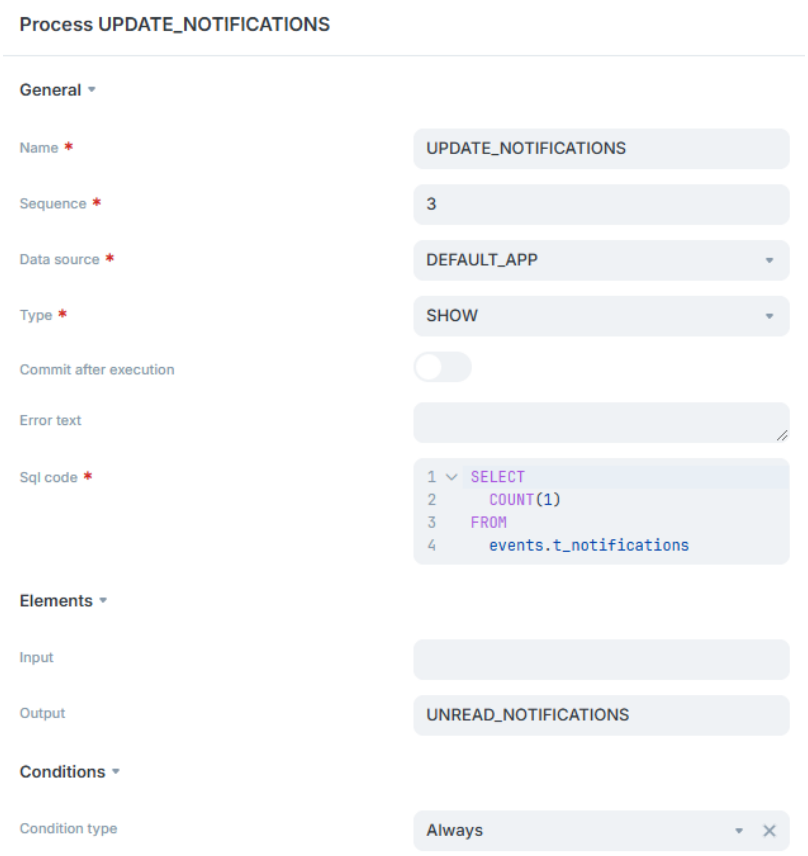
When a request from an application is made, the system selects all processes (global and page) that meet the conditions of the request, sorts them according to their sequence numbers, and executes them.

Process management

To create or delete a process, one should call the context menu by right clicking the process list and selecting the appropriate action.



After filling in or editing the process parameters, confirm the changes by clicking the "Save" button.



The process parameters are divided into 3 groups:

1. General

Parameter	Description
Name	A mandatory attribute that defines the process name.
Sequence number	A mandatory attribute that defines the order in which the process is executed.
Data source	A mandatory attribute that specifies in which data source the process will be executed.
Type	A mandatory attribute that defines the type of process: • SHOW • PROCESS • AJAX
Post-execution commit	The flag determining the necessity to terminate the transaction
Error text	A string of text that will be displayed if the process terminates with an error.
Sql code	A mandatory attribute that defines the code of the SQL query.

2. Elements

Parameter	Description
Input	Names of variables, separated by commas, containing input parameters
Output	Names of variables, separated by commas, where the results will be stored

3. Conditions

Condition type - defines the condition under which the process, action or event will be executed.

Types of conditions

There are 11 types of conditions available to the developer:

1. Always - is executed always.
2. Exists (SQL query returns at least one row) - allows to execute a validation SQL query. The condition is considered to be met if the query returns at least one row.
3. Value of Item / Column in Expression 1 Is NOT NULL - built-in condition for filling a field or a table cell. The condition is considered to be met if the input field or table cell contains any value.
4. Value of Item / Column in Expression 1 != Zero - built-in condition for the value different from 0 (zero) of the input field or table column. The condition is considered to be met if the checked object contains a value different from 0 (zero).
5. Value of Item / Column in Expression 1 Is NULL - built-in condition for NULL value (empty value) of input field or table column. The condition is considered to be met if the checked object contains NULL (empty value).
6. Value of Item / Column in Expression 1 Is NULL or Zero - built-in condition on the value of NULL (empty value) or 0 (zero) of the input field or column of the table. The condition is considered to be met if the checked object contains NULL (empty value) or is equal to 0 (zero).
7. Value of Item / Column in Expression 1= Zero - built-in condition for the value 0 (zero) of the input field or column of the table. The condition is considered to be met if the check object will contain the value 0 (zero).
8. Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero - a built-in condition for filling and value other than 0 (zero) of an input field or table column. The condition is considered to be met if the checked object does not contain NULL (empty value) and is not equal to 0 (zero).
9. NOT Exists (SQL query returns no rows) - allows to execute a verification SQL query. The condition is considered to be met if the query returns no rows.
10. SQL Expression - allows to execute an SQL query that returns TRUE or FALSE. The query input field contains only the body of the query, without SELECT and/or FROM keywords. If some subquery is to be executed, it must be wrapped in "(" ")". The condition is considered to be met if the query returns TRUE.
11. Never - never to be performed.

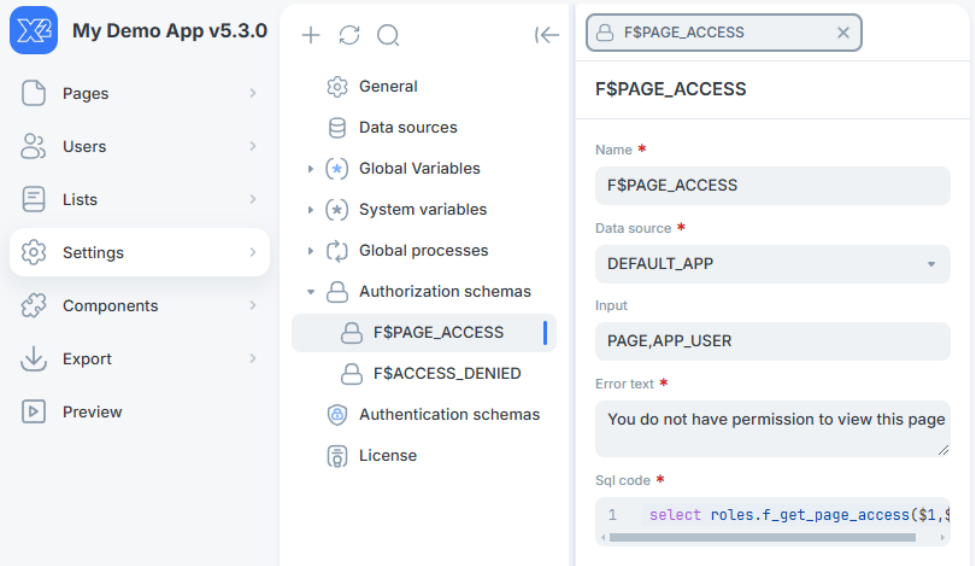
7.2.5 Authorization schemes

General description

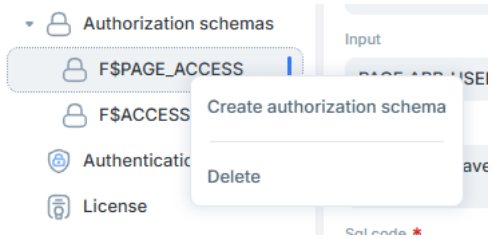
The authorization scheme determines whether a page is available to a particular user. If authorization is successful, the requested page is displayed to the user. If an error occurs during authorization, an error message is displayed.

Creating an authorization scheme

The list of authorization schemes is located in the "Settings" -> "Authorization schemas" section:

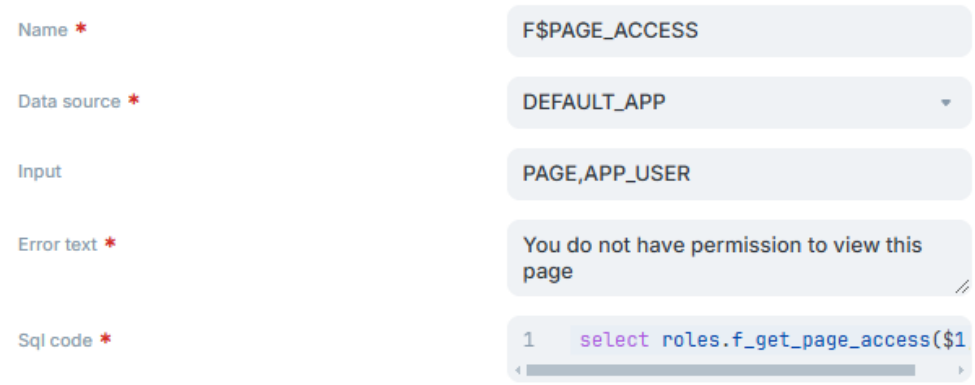


Right click the items of the "Authorization schemas" list, and a context menu with buttons for creating a new scheme and deleting the selected one will be displayed:



Note: To access the page, you must be logged into the account with the access role above VIEW.

A new authorization scheme is created by pressing the "Create" button. The form to create the authorization scheme will open:



Attributes:

- Name is a mandatory attribute that specifies the name of the authorization scheme to be created.
- Data source is a mandatory attribute defining the data source for the created authorization scheme.
- Input is a mandatory attribute that defines a list of input parameters for authorization.

- Error Text is a mandatory attribute that defines the text of the authorization error.
- SQL is a mandatory attribute that defines the SQL query for authorization.

After clicking the "Save" button and confirming the entered data, the authorization scheme will be entered into the database and will be available for viewing, editing and deleting.

Editing the authorization scheme

To edit the authorization scheme, select the needed one. After clicking on the scheme, a form (similar to the form for creating a new authorization scheme (see "Creating an authorization scheme" above) will open with the information about the selected scheme.

Edit the attributes you need and save their new values by clicking the "Save" button.

To delete the scheme, right click on the scheme, click the "Delete" button in the menu and confirm the action.

7.2.6 Authentication schemes

General information

Authentication is the process of verifying the identity of each user who accesses the application.

The authentication process requires the user to provide certain login credentials, such as a username and password. If the credentials pass, the user is granted access to the application, if not, access is denied.

After successful user authentication, the value of the global variable APP_USER is set (see Global variables). When the user moves from page to page, the APP_USER value is used to identify the user.

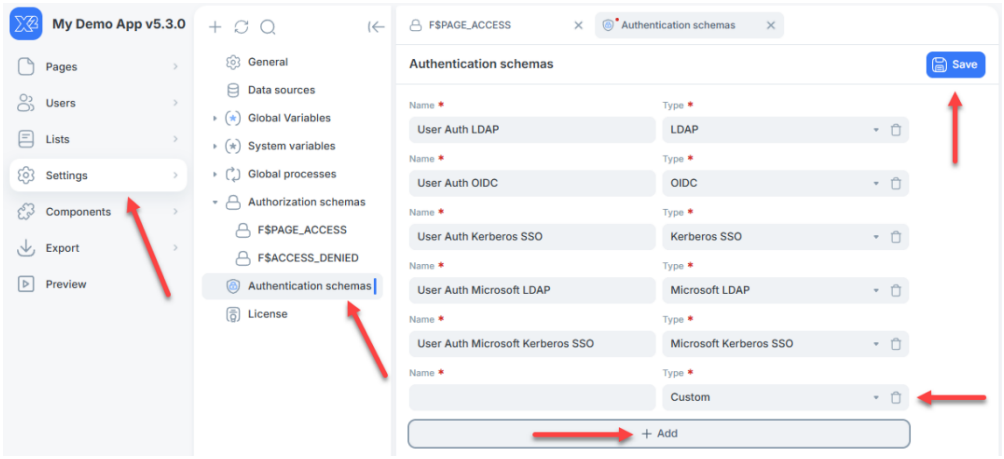
The PGHS Application Server supports authentication and authorization using the following schemes:

- By login and password - CUSTOM
- Microsoft LDAP
- Microsoft Kerberos SSO
- LDAP
- Kerberos SSO
- Open ID Connect

When developing an application in the XRAD builder, the developer defines possible authentication schemes by specifying the scheme name and type. Thus, the application can have many schemes of the same type, which allows flexible customization of user authentication. For example, it is possible to implement authentication of users of different Active Directory domains.

Creating an authentication scheme

The page with authentication schemes is located in the "Settings" -> "Authentication schemes" tab, one can add a new scheme by pressing the "Add" button. Besides, any scheme can be deleted by pressing the button with the trash can icon, to apply all changes you should press the "Save" button. The parameters of each authentication scheme are set directly in the auth_config.json file of the PGHS application server, the description of this file and parameters can be found in the PGHS documentation.

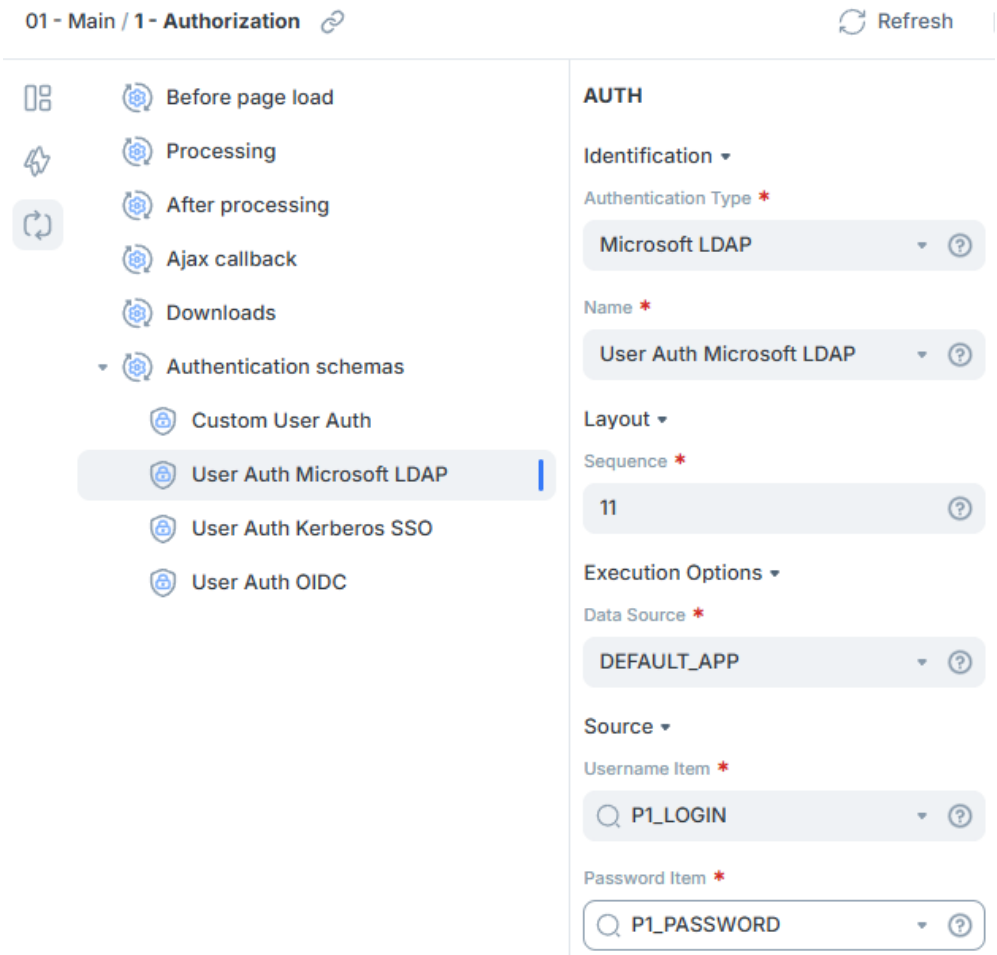


Note: To access the page, you must be logged into the account with the access role above VIEW.

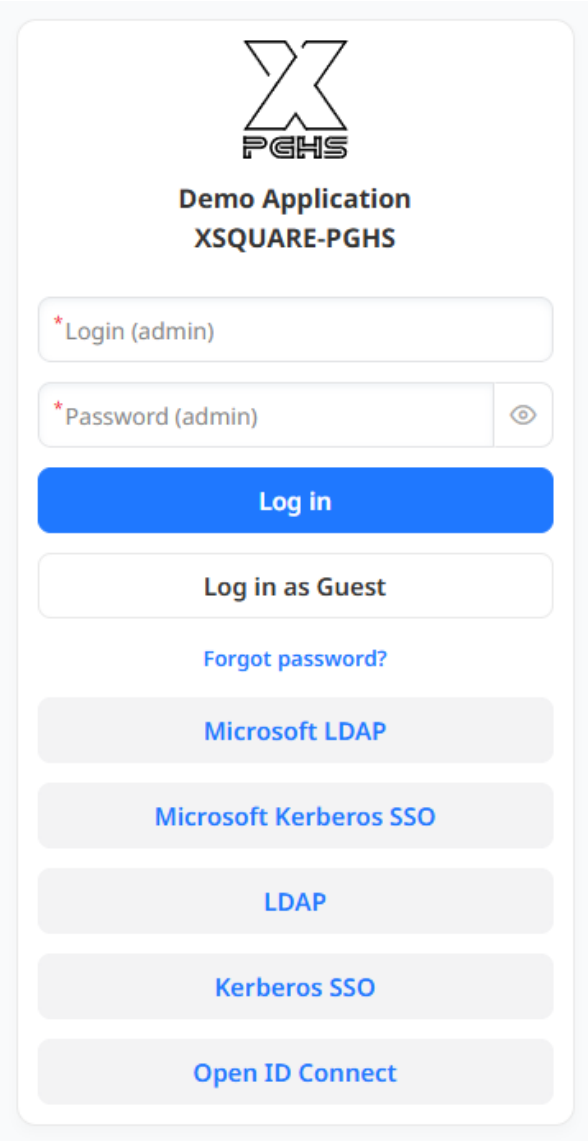
At startup, the PGHS Application Server loads schemas from the auth_config.json file and maps them by name and type to parameters in the XRAD database.

For each authentication page of the application under development, you can create a different set of authentication methods:

Example 1:

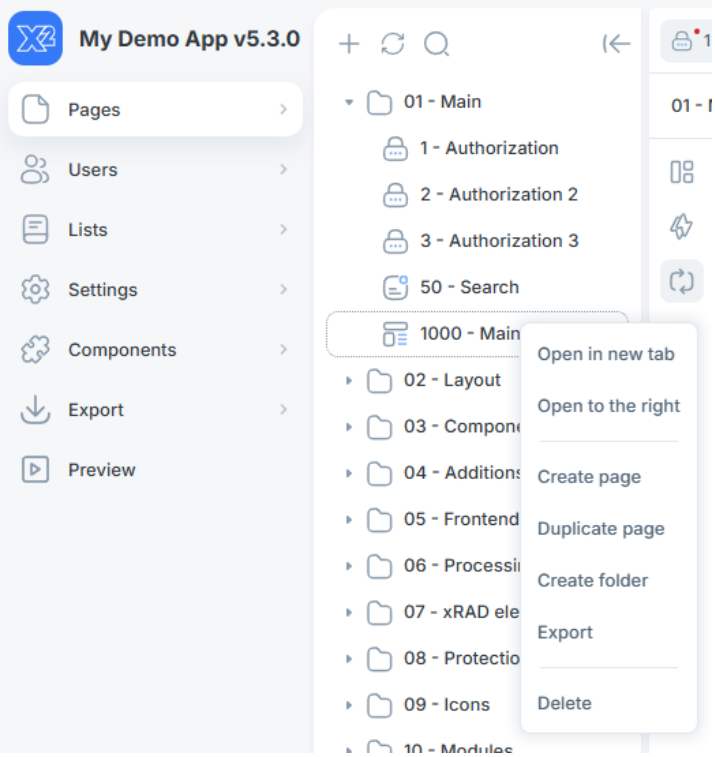


Example 2:



7.3 Working with pages

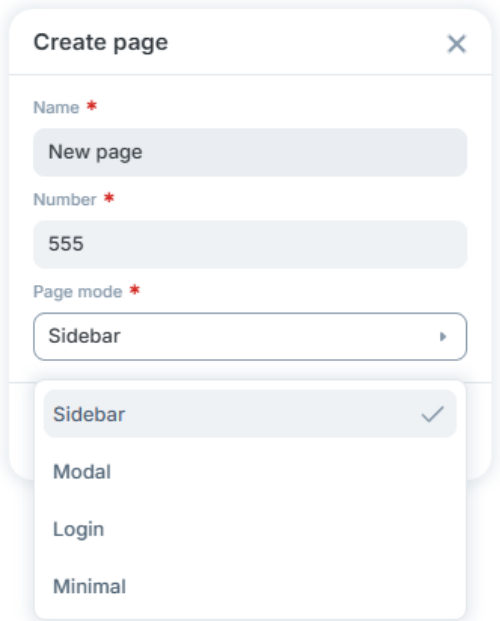
A page is the main block of the web application which is being developed. To display the list of available pages, select the "Pages" section in the main menu. All possible actions with pages are available from the context menu opening by the mouse right click. The developer can create a new page, duplicate, export and delete the current page, as well as create folders to group pages.



7.3.1 Creating a new page

One can create a new page using the context menu or the "+" button. In the page creation window one needs to specify the attributes of the new page:

- Name - string name of the page
- Number - integer page identifier
- Page mode - the page view template.



The following types of pages are available:

- Standard - a page with a side menu and navigation panel in the page header.
- A modal window is a page that is displayed on top of the parent (calling page).
- Login - page for authentication
- Minimalistic - a simple page without a side menu and with a header without a navigation panel.

The "Page editor" will open after the page has been created.

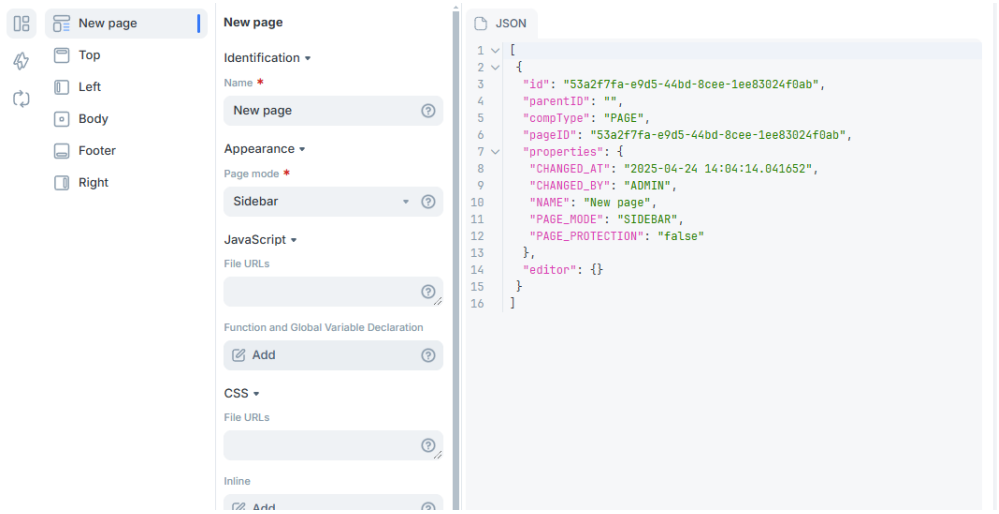
7.3.2 Page Editor

All work on forming a new page takes place in the Page editor.

At the top of the Page editor is a toolbar that displays the page ID and name, a button for copying the page address to the clipboard, and buttons for refreshing, previewing, locking and saving page changes.

Below the toolbar there are three vertical areas of the Page editor, which logically follow the concept of the entire development environment:

- Main menu
- Parameter list
- Code editing area



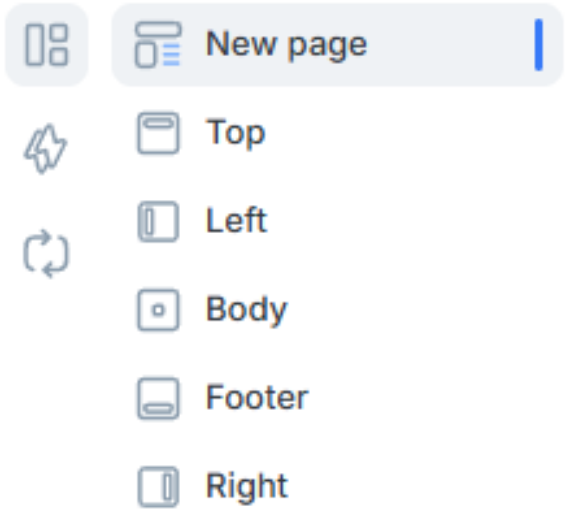
Main menu of the page editor

The main menu consists of 3 tabs:

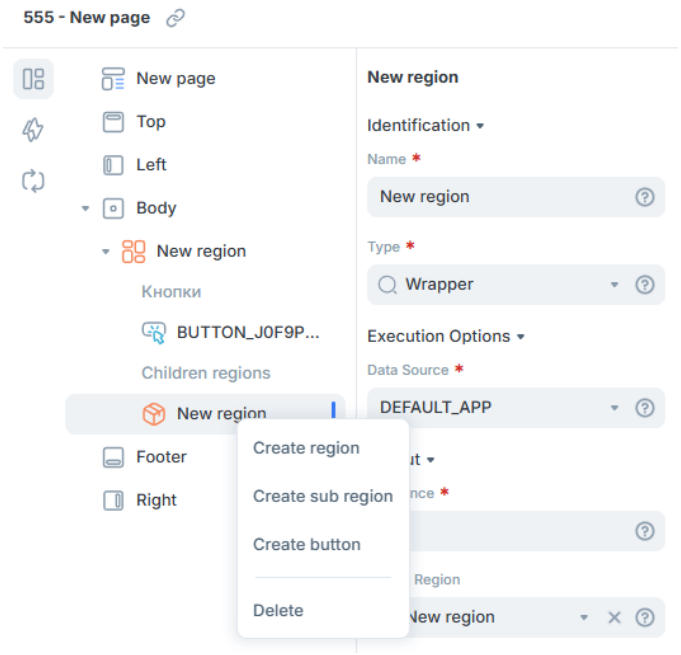
- Elements
- Events
- Processing

Elements

The Elements tab displays the components of the page - regions, form elements, and buttons.



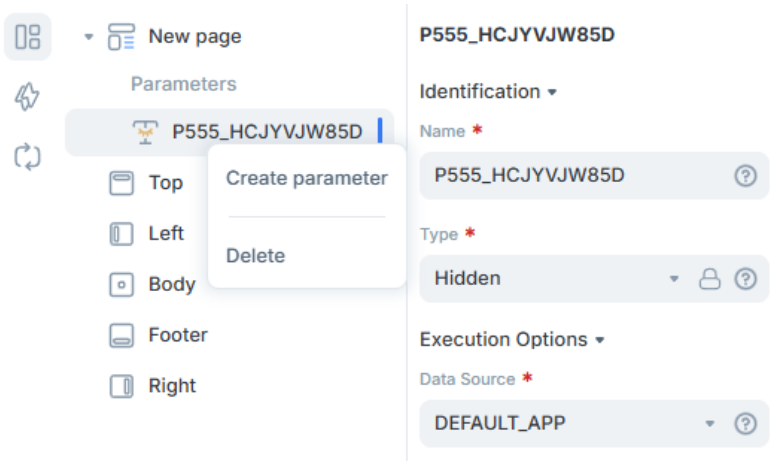
Components can be placed at the page level in a specific page region (Top, Left, Body, Footer, Right) or can be nested in the regions in a specific region position. The basic construction that is created from the context menu is a region. A region can include child regions, buttons, and other visual components that are defined by the region type.



If no page region is selected, items are created in the center region (Body) by default.

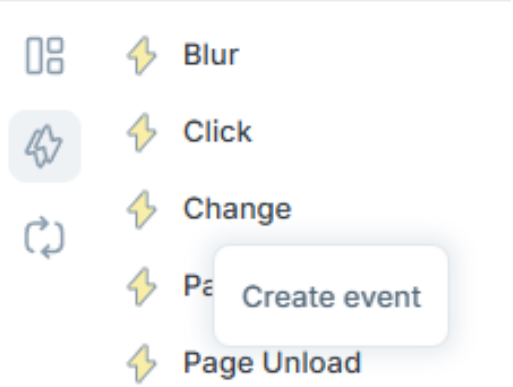
Note: buttons can only be created within a region.

Besides, it is possible to create page parameters from the context menu of the page name. These parameters, unlike global application parameters, are available only within a specific page.



7.3.3 Events

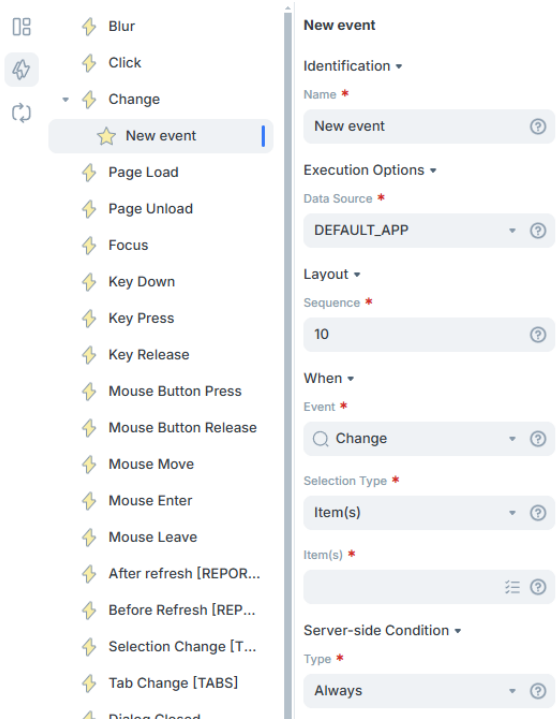
The **Events** tab displays a list of events and actions grouped by types of possible events. To create a new event, the developer should call the context menu by right click.



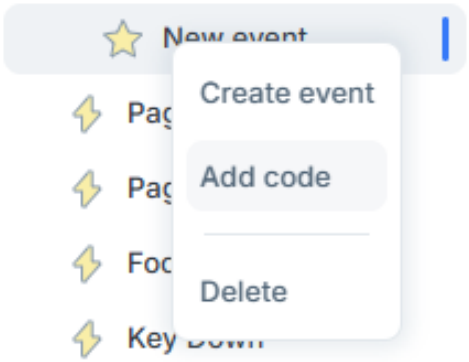
After creating an event, you should configure its parameters in the parameters area:

- Name is a mandatory attribute that defines the name of the event.
- Data source is a mandatory attribute that defines the data source for storing the list item.
- Sequence is a mandatory attribute, a sequence number according to which the action will be executed.
- Event is a mandatory attribute that defines the type of event.
- Selection type is a mandatory attribute that defines the type of page element to trigger the event.
- Display condition type is a mandatory attribute that specifies the type of condition that must be met to trigger the event.

To apply the changes, one must save the changes via the button in the toolbar.



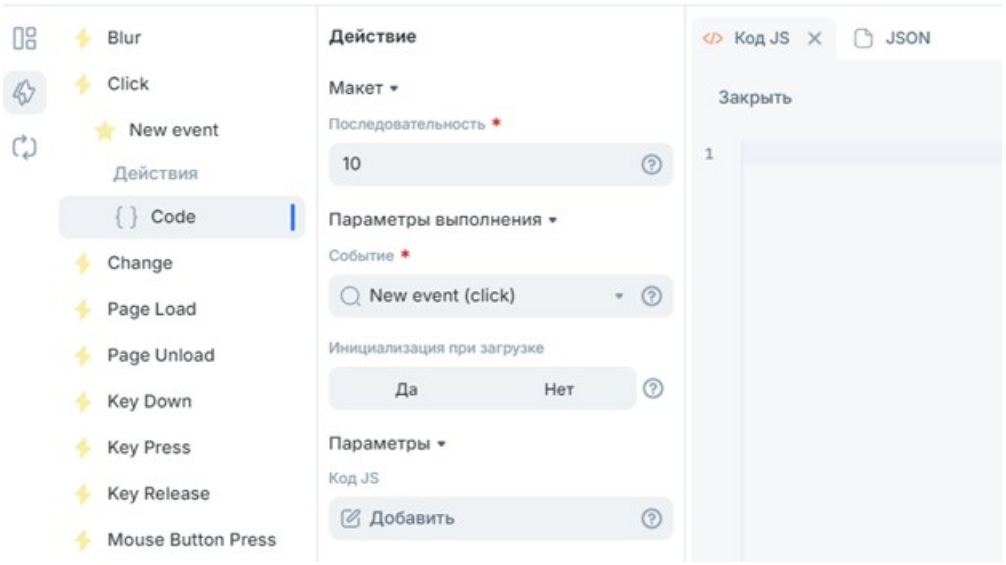
One can bind an Action to the event in the form of JavaScript code. The action can be created from the context menu using the "Add code" button.



It is also necessary to define parameters for the action to be created:

- Sequence is a mandatory attribute, a sequence number according to which the action will be executed.
- Event is a mandatory attribute, defines the name of the event to which the action code is bound.
- Initialize on load - defines the execution of the action when the page is loaded.
- JS code is a mandatory attribute, defines the JavaScript code that will be executed when the event occurs.

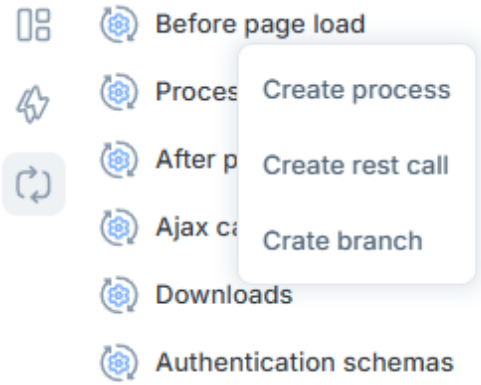
Clicking the "Add" button in the tab area of the editor will open the "JS Code" tab with an interactive code editor. Saving JS code changes is done in the code editor, and saving parameters is carried out using the button in the toolbar.



7.3.4 Processing

When running an application, it is often necessary to perform requests to the database. For this purpose, XRAD provides a mechanism for **processing** requests for each page - **processing**.

The **Processing** tab displays the list of request handlers grouped by process type or handler call type. The composition of the context menu available at right click is determined by the group type:



Processing provides the following types of request handlers:

- A process is the main mechanism of interaction between an application and a database. Processes are executed when various requests (Request) are received from the application. When a request is received from the application, all processes of the corresponding type are executed in the Sequence order.
- A branch is a process that redirects to another application page or link when a Server-side Condition is met.
- Validation is the process of verifying that the data entered by the user is correct.
- REST call is a process that allows to call a third-party REST API. Allows the application to process and make requests to third-party services. The call will be made by the PGHS application server.
- The Data Grid process is the process for handling changed data in DATAGRID.

- Download is the process that allows to upload a file from the database to the client.
- Authentication schema - the process of authenticating a user.

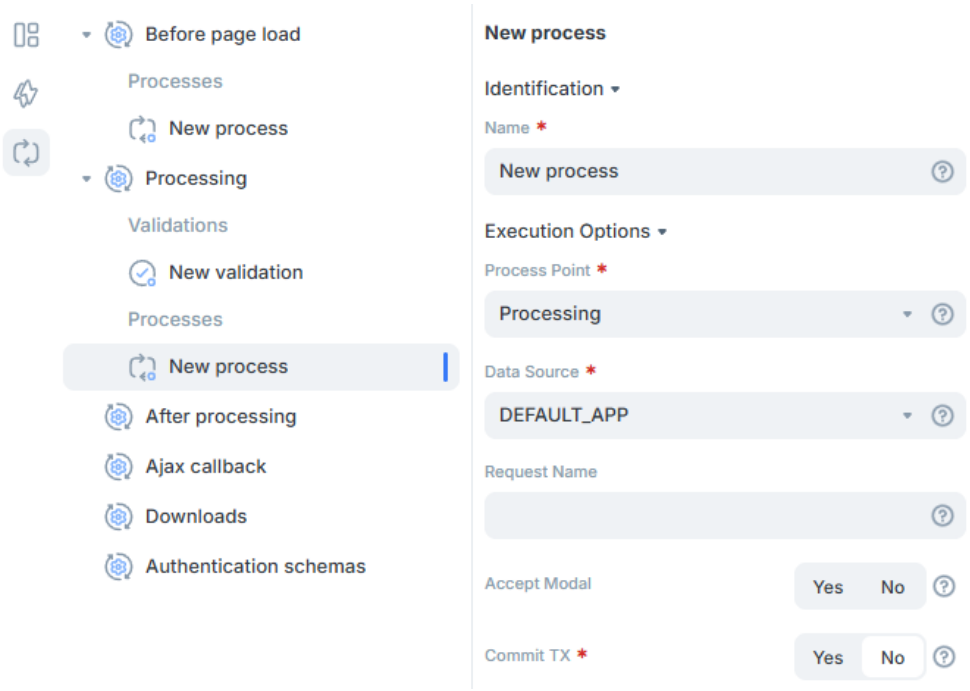
Processes are also grouped by process type:

- Before page loading - processes that are executed before the page is loaded. Their main purpose is to prepare components for interaction with the user (for example, to fill form fields with values from the database). The following types of handlers can be executed before the page is loaded: processes, REST calls and branches.
- Processing - processes of this type are performed when the form is submitted to the server (submit).
- After processing - processes of this type are executed after page processing. For example, transitions to other pages by pressing buttons.
- Ajax callback - processes of this type allow to interact with the database on a request from a JS script.

The response from Ajax processes must be handled by the developer in a JS script. The other types of processes can interact with the user independently, reporting whether the request was successful or whether an error occurred.

Processes

To create a process type handler, it is necessary to select by right click the corresponding item of the context menu and fill in the parameters of the process to be created.



Parameter	Type	Description
Name	Text	Process Name.

continues on the next page

Table 1 - continued from previous page

	Typ	
Process Point	List	The type of the process. The type affects the order in which code is called. XRAD provides the following types of processes: • Before the page loads • Processing • Ajax callback
Data source	List	The name of the data source for this process.
Request name	Text	Specifies the name of the request. If specified, the process will only be executed when the value of the global variable REQUEST corresponds to the entered value.
Accept modal	Switch	Determines whether the modal window should be closed after the process is executed.
Commit TX	Switch	Specifies whether a COMMIT transaction should be called after successful completion of a process. When COMMIT is called, all changes made to the database by processes that were executed before this process will be written to the database. After that, XRAD will open a new transaction to execute subsequent processes.
Source - SQL	Text area	Specifies the SQL query to be executed.
Source parameters	Text / Builder window	Specifies the elements to be passed to the executable request as variables.
Output parameters	Text / Builder window	Defines the output elements. If the process needs to set the values of session variables, it is necessary to specify them in this field.
Sequence	Number	Specifies the sequence of process execution in a list of processes of the same type.
Success message	Text area	Defines the message to be displayed to the user when a process is successfully executed. If a chain of processes is called, the last successful execution message will be displayed.
Error message	Text area	Defines the error message.
Display conditions - Type	List	Specifies the type of condition for displaying the region on the page. By default - Always. Depending on the type, additional condition parameters are required. For Always and Never, no additional conditions are required.
Display conditions - First condition	Text area	Query in SQL format. The parameter is applicable for the following types of conditions: • Exists (SQL query returns at least one row). If the query returns at least one row, the region will be displayed on the page. • NOT Exists (SQL query returns no rows). If the query returns no rows, the region will be displayed on the page.
Display Conditions - SQL Expression	Text area	Logical expression in SQL language. If the expression returns true, the region will be displayed on the page. The parameter applies to the SQL Expression condition type.

continues on the next page

Table 1 - continued from previous page

	Typ	
Display conditions – First input	Text / Builder window	Specifies a list of input parameters for the SQL query in the "First condition" or “SQL Expression” field. For each substitution variable in the request, an input parameter must be defined. Can accept values of global variables and page input and selection elements. Can be entered as text or selected in the builder. Applicable for condition types: • Exists (SQL query returns at least one row). • NOT Exists (SQL query returns no rows). • SQL Expression.
Display Conditions - Element	Text / Builder window	Allows to select the element, depending on the value of which the region is displayed or not. Applies to the following types of conditions: - Value of Item / Column in Expression 1 Is NOT NULL. Region will be displayed if the value of the element is not NULL. - Value of Item / Column in Expression 1 != Zero. The region will be displayed if the item value is not equal to 0. - Value of Item / Column in Expression 1 Is NULL. The region will be displayed if the item value is NULL. - Value of Item / Column in Expression 1 Is NULL or Zero. The region will be displayed if the item value is NULL or zero. - Value of Item / Column in Expression 1= Zero. The region will be displayed if the value of item is 0. - Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero. The region will be displayed if the item value is not NULL and is not equal to zero.

Note: When a request from an application is made, first all global processes that meet the conditions of the request are executed according to their sequence numbers, followed by the processes of the currently open page.

Process conditions

There are 2 types of conditions to execute the process:

1. Request,
2. Server-side condition.

Request execution

When running, the application is constantly sending various requests. Whether it is a page submit or a request from a JS script. Depending on the source of the request, processes of different types are executed, but they are all united by the type of execution - all processes of the same type are executed in the sequence order. The Request Name field is used to prevent unnecessary processes from being executed. If the Request Name field is empty, the process will be executed at any request from the application, if the process type matches the request type. However, if you specify a specific request name in the Request Name field, the process will be executed only if the name of the request sent by the application matches the name in the Request Name field.

The same effect can be achieved by using a Server-side Condition with the SQL Expression type and specifying the REQUEST field as the input field.

Server-side condition

There are 11 types of process execution conditions available to the developer:

1. Always - is executed always.
2. Exists (SQL query returns at least one row) - allows to execute a validation SQL query. The condition is considered to be met if the query returns at least one row.
3. Value of Item / Column in Expression 1 Is NOT NULL - built-in condition for filling a field or a table cell. The condition is considered to be met if the input field or table cell contains any value.
4. Value of Item / Column in Expression 1 != Zero - built-in condition for the value different from 0 (zero) of the input field or table column. The condition is considered to be met if the checked object will contain a value different from 0 (zero).
5. Value of Item / Column in Expression 1 Is NULL - built-in condition for NULL value (empty value) of input field or table column. The condition is considered to be met if the checked object contains NULL (empty value).
6. Value of Item / Column in Expression 1 Is NULL or Zero - built-in condition on the value of NULL (empty value) or 0 (zero) of the input field or column of the table. The condition is considered to be met if the check object contains NULL (empty value) or is equal to 0 (zero).
7. Value of Item / Column in Expression 1= Zero - built-in condition for the value 0 (zero) of the input field or column of the table. The condition is considered to be met if the checked object will contain the value 0 (zero).
8. Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero - a built-in condition for filling and value other than 0 (zero) of an input field or table column. The condition is considered to be met if the checked object does not contain NULL (empty value) and is not equal to 0 (zero).
9. NOT Exists (SQL query returns no rows) - allows to execute a verification SQL query. The condition is considered to be met if the query returns no rows.
10. SQL Expression - allows to execute an SQL query that returns TRUE or FALSE. The query input field contains only the body of the request, without SELECT and/or FROM keywords. If some subquery is to be executed, it must be wrapped in "(" ")". The condition is considered satisfied if the query returns TRUE.
11. Never - is never executed.

Transaction management

When executing a chain of database requests, sometimes a situation arises where an error in the current step should not cancel the results of the previous steps. This situation is solved by transaction completion when the process is successfully completed. To complete the transaction, the process must set the transaction commit flat (Commit TX).

If this flag is not set, then, in case of an error in any of the processes, all changes made by other processes that do not have the transaction completion flag set will be canceled. Thus, it is possible to split a large action into groups of smaller actions, errors in which will not affect the actions performed in the previous groups.

Closing a dialog

Applications often use modal windows to perform certain actions. For example: user creation. When pressing the "Create" button, a certain process bound to the button is executed. In such cases, the task of the modal window is completed, and the window can be closed. To make the task easier, the processes have

a special flag. If this flag is set, the modal window will be automatically closed with the result Accept if the process is successful.

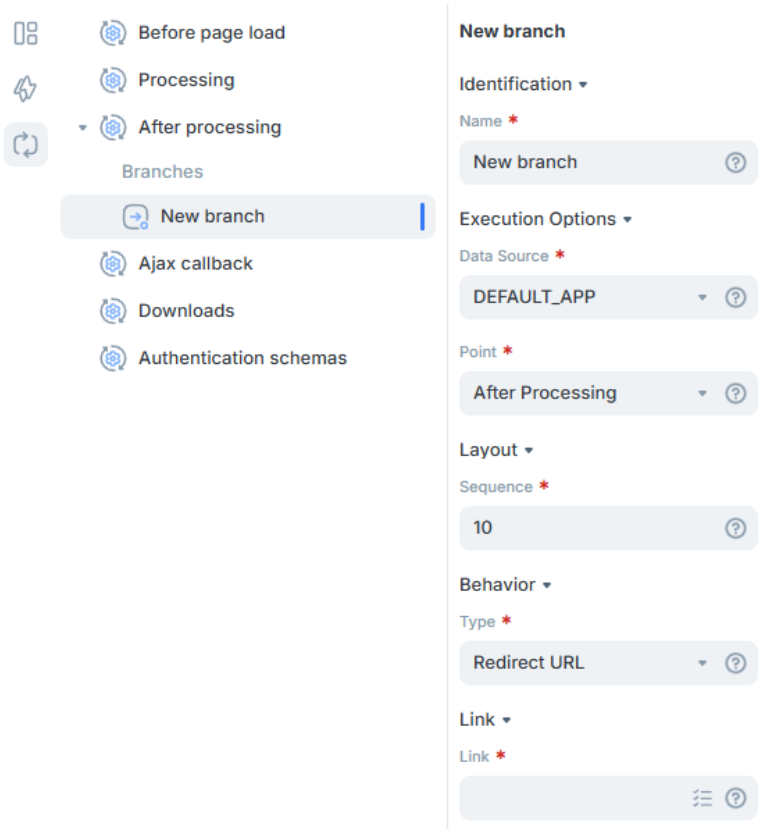
If several processes successfully respond to a request, the modal window will be closed automatically only if the last process in the process chain has the Accept Modal flag set.

Branches

When running an application, there are situations when it is necessary to automatically switch to some page when loading the selected page or after executing a certain process. For this purpose, branches - transitions between pages - are used.

If the branch is located in a block, the transition to the page specified in the Link field will be performed when the page on which the Branch is located is loaded if the condition in the Server-side Condition block is met. For unconditional redirection in the Server-side Condition block, it is necessary to use the value Always, for disabling redirection - Never.

If the Branch is located in the «After Processing» block, the transition to the page specified in the Link field will be performed when all processes are completed and the condition in the Server-side Condition block is met. For unconditional redirection in the Server-side Condition block, it is necessary to use the value Always, for disabling redirection - Never.



Please find below the unique parameters for processes of the branch type:

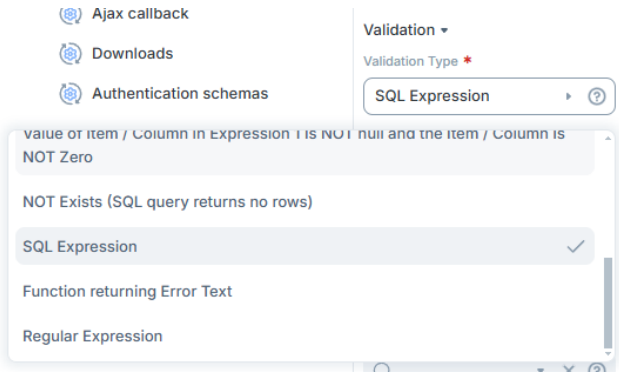
Parameter	Type	Description
Name	Text	Branch Name.
Data Source	List	Specifies the data source for this branch.
Transition location	List	Defines the location where the transition is to be performed. • Before page load - the transition will be executed before the page is loaded. This item is convenient to use in case it is necessary to implement redirection of the user depending on certain conditions. • After processing - redirection will be executed after the execution of the processes is complete.
Sequence	Number	Defines the sequence of process execution in the process list of the same type.
Behavior - Type	List	Specifies the type of page transition to execute: • Redirect URL • Function that returns the URL
Link	Text / Builder window	Specifies a link to a page if the transition type "Redirect URL" is set.
Source - SQL	Text area	Defines an SQL query that returns a URL
Request parameters	Text / Builder window	Defines the elements to be passed to the executable request as variables.
Output parameters	Text / Builder window	Defines the output elements. If the process needs to set the values of session variables, one must specify them in this field.
Display conditions Type	List	Specifies the type of condition for displaying the region on the page. By default - Always. Depending on the type, one needs to specify additional parameters No additional conditions are required for Always and Never.

Validation

User interaction is required in a fully-fledged application. All user actions and data must be stored in a database. However, the user may intentionally or unintentionally enter incorrect data. To avoid this, XRAD provides data validation. Validation is the verification that the data entered by the user into the form input fields is correct.

Types of validators

In XRAD, the validation process is a call to various functions and procedures (hereinafter referred to as validators) that are executed before the data is sent to the server for processing.



There are 11 types of validators available to the developer:

1. Exists (SQL query returns at least one row) - allows to execute a validation SQL query. The check is considered successful if the query returns at least one row.
2. Value of Item / Column in Expression 1 Is NOT NULL - built-in check if a field or a table cell is filled. The check is considered to be passed successfully if the input field or table cell contains any value.
3. Value of Item / Column in Expression 1 != Zero - built-in check for a value different from 0 (zero) of an input field or table column. The check is considered successful if the checked object contains a value other than 0 (zero).
4. Value of Item / Column in Expression 1 Is NULL - built-in check for NULL value (empty value) of input field or table column. The check is considered successful if the checked object contains NULL (empty value).
5. Value of Item / Column in Expression 1 Is NULL or Zero - built-in check for NULL (empty value) or 0 (zero) value of input field or table column. The check is considered successful if the checked object contains NULL (empty value) or is equal to 0 (zero).
6. Value of Item / Column in Expression 1= Zero - built-in check for value 0 (zero) of input field or table column. The check is considered to be passed successfully if the checked object contains the value 0 (zero).
7. Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero - built-in check for fullness and value other than 0 (zero) of an input field or table column. The check is considered successful if the checked object does not contain NULL (empty value) and is not equal to 0 (zero).
8. NOT Exists (SQL query returns no rows) - allows to execute a validation SQL query. The check is considered successful if the query returns no rows.
9. SQL Expression - allows to execute SQL query returning TRUE or FALSE. The query input field contains only the body of the request, without SELECT and/or FROM keywords. If necessary to execute a subquery, it must be wrapped in parentheses "(" ")". The check is considered successful if the query returns TRUE.
10. Function returning Error Text - allows to execute SQL query returning error text. In the query field it is necessary to specify a full query returning a string. The check is considered successful if the query returns NULL. Otherwise, the error text returned by the query will be displayed.
11. Regular Expression - allows to check the value of a form field by a regular expression. The Item field contains the field to be checked, and the Value field contains the regular expression. The check is considered successful if the value of the Item field matches the regular expression.

There is also a separate type of validation - a flag that determines whether a form field must be filled in. For any input field it is possible to set a flag that the field must be filled in. In this case, even if no other validators are generated, when trying to send data to the server, a check will be performed to see if the field is filled with data.

Validation error

If an exception occurs in any of the validators present on the form, sending them to the server is blocked and the user receives a developer-defined warning. If the validator is bound to an input field, the input field where the exception

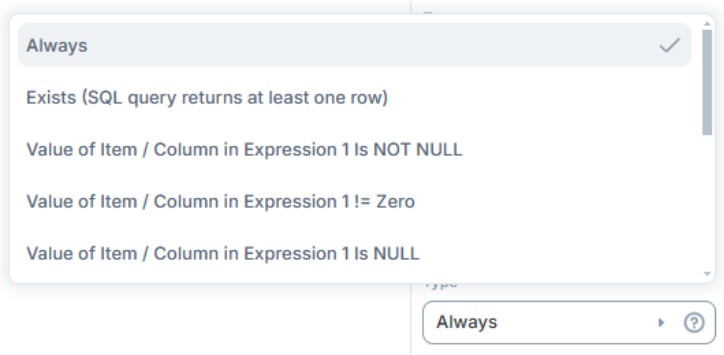
occurred will be highlighted in addition.

System validators

If a field is marked as mandatory but it is not filled in, the system validator will be triggered, and the user will be notified about the need to fill in mandatory fields with a pop-up tooltip. If additional validation is to be added to an input field, a validator must be created for this field. In this case, the system validators will be called first, and then the user validators will be called.

Conditions for performing validation

Since validations are performed automatically when a page is submitted, it is often necessary to limit their execution. For this purpose, XRAD provides a block of conditions for validations.



There are 11 types of validator fulfillment condition types available to the developer:

1. Always - is executed always.
2. Exists (SQL query returns at least one row) - allows to execute a validation SQL query. The condition is considered to be met if the query returns at least one row.
3. Value of Item / Column in Expression 1 Is NOT NULL - built-in condition for filling a field or a table cell. The condition is met if the input field or table cell contains any value.
4. Value of Item / Column in Expression 1 != Zero - built-in condition for the value different from 0 (zero) of the input field or table column. The condition is met if the checked object will contain a value different from 0 (zero).
5. Value of Item / Column in Expression 1 Is NULL - built-in condition for NULL value (empty value) of input field or table column. The condition is met if the checked object contains NULL (empty value).
6. Value of Item / Column in Expression 1 Is NULL or Zero - built-in condition on the value of NULL (empty value) or 0 (zero) of the input field or column of the table. The condition is met if the checked object contains NULL (empty value) or is equal to 0 (zero).
7. Value of Item / Column in Expression 1= Zero - built-in condition for the value 0 (zero) of the input field or column of the table. The condition is met if the checked object contains the value 0 (zero).
8. Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero - a built-in condition for filling and value other than 0 (zero) of an input field or table column. The condition is met if the checked object does not contain NULL (empty value) and is not equal to 0 (zero).
9. NOT Exists (SQL query returns no rows) - allows to execute a verification SQL query. The condition is met if the query returns no rows.
10. SQL Expression - allows you to execute an SQL query that returns TRUE or FALSE. The query input field contains only the body of the query, without SELECT and/or FROM keywords. If some subquery is to be executed, it must be wrapped in "(" ")". The condition is met if the query returns TRUE.
11. Never - is never executed.

REST calls

A REST call is a process that allows to call an external REST API.

Below we review the parameters unique to this type of process:

Parameter	Type	Description
Source - URL	Text	Specifies the URL to execute the request. The URL must be specified in full including the protocol.
HTTP-method	List	Specifies the HTTP method for executing the request: • GET • POST • PUT • DELETE • PATCH
HTTP headers	Builder window	Defines HTTP headers. Headers can include static values, as well as accept the values of the form elements.
URL query parameters	Builder window	Defines request parameters to be added to URL. The parameter values can be either static or taken from an input element.
Payload type	List	Specifies the body type of the request. Possible options: • None • Static value • Form-encoded • SQL
Response processing	List	Specifies the type of service response handler: • Absent • Assign variables • SQL handler
Variable assignment	Builder window	Determines the correspondence between form elements and response elements in JSON format. One can specify the JMES path as an argument of the response handler. If the path returns only JSON (string, number, Boolean value), the element will contain this value. If the JMES path returns a compound value (object or array), the returned value in JSON format will be passed to the element.
SQL handler	Builder window	Defines the SQL handler.
Inputs Parameters	Text / Builder window	Specifies the elements to be passed to the SQL query as variables.

Downloads

The process of downloading a file from a web application. When creating the download process, one needs to set the parameters:

Parameter	Type	Description
Name	Text	The name of the download process.
Data Source	List	Specifies the data source for this process.
Request Name	Text	Specifies the name of the request. If specified, the process will only be executed when the value of the global variable REQUEST corresponds to the to the value entered.
Sequence	Number	Defines the sequence of process execution in the process list of the same type.
Source - SQL	Text area	Specifies the SQL query that returns the URL.
File Name Column	List	Specifies the column containing the name of the file.
File Column	List	Specifies the column that contains the file data.
Mime Type Column	List	Specifies the column containing the mime type of the file.
Content Disposition	List	Specifies the content of the Content-Disposition header: • Attachment • Inline The header is responsible for how the content of the file is to be displayed in the browser. If attachment is specified, the file will be downloaded to the user's computer. If inline is specified, the content of the file will be displayed in the browser.

Authentication schemes

Authentication pages are used to restrict access to non-public application pages. Authentication methods are defined by authentication processes, which in their turn refer to global authentication schemes.

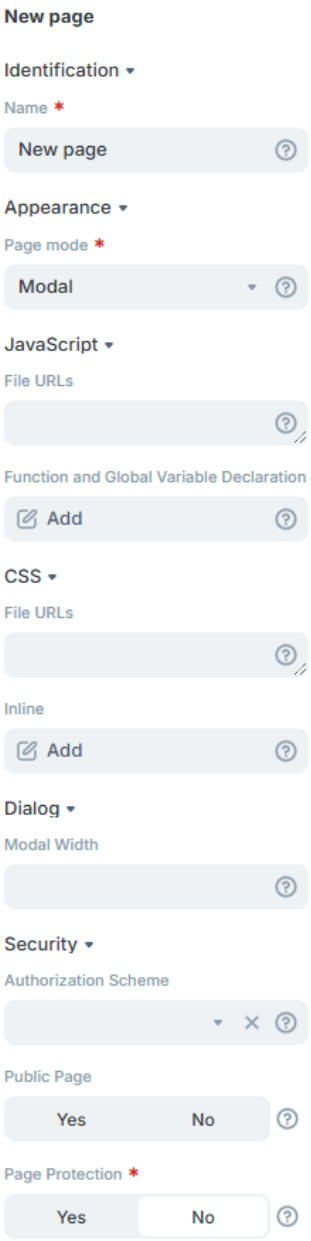
Below one can see the unique parameters of the authentication process:

Parameter	Type	Description
Authentication Type	List	Specifies the authentication type. Supported authentication types: • Custom • LDAP • Microsoft LDAP • Kerberos SSO • Microsoft Kerberos SSO • OIDC
Name	List	Specifies the name of the authentication process from the list of global authentication schemes in accordance with the selected type.
Source - Login	List	Specifies the form element to be used for login input. This attribute is available for LDAP and Microsoft LDAP schemes.
Post-Login Function	Text area	Specifies the SQL query that will be executed after successful authentication.

Page Options

A specific set of parameters is available for each page you create:

- Name is a mandatory attribute that defines the name of the page.
- Page mode is a mandatory attribute that defines the page view template. The following template options are available: standard, modal window, login, minimalist (description can be found in the “Page Creation” section).
- JS file URLs is a list of JavaScript file URLs with the code to be loaded for this page.
- JS functions and global variables declaration - used to define JS variables and functions at the page level.
- CSS file URLs -is a list of CSS file URLs that will be loaded for this page.
- Inline CSS - is used to define styles at the page level.
- Width - defines the width of the modal window. Available only for Modal Window page mode.
- Authorization scheme - determines whether the page is available to a specific user (See "Authorization schemes" section).
- Public page - determines whether a user needs to be authenticated (See “Authentication Schemes” section).
- Page protection - determines whether the parameters in the page request should be protected by a checksum.



7.3.5 Modal page

A modal page is a child window on top of the parent window, located in the same browser window. The modal page remains active until the user finishes working with it and closes it. The user cannot interact with the parent part of the page until the modal page is closed.

Calling a modal page

There are 2 ways to call the modal page:

- By link - this method does not provide for processing the result of the modal page execution.
- From the Javascript code

An example of calling a modal page using JavaScript:

```
jsAPI.modal.open({
  page: {
    src: '1010',
    query: {
      P1010_ID: jsAPI.getItem('P1010_ID'),
      clear: '1010'
    }
  },
  title: 'My modal page'
});
```

JavaScript call code can be used as a link value through the Redirect to page action, or through a Click type event on the selected button.

Closing a modal page (the "Cancel" event)

Closing the modal page is performed via Javascript code executed on the current modal page:

```
jsAPI.modal.close()
```

Updating data on the parent form

To handle the closing of a modal page by the required event, it is necessary:

Create an **Event of** type **Dialog Closed** (when closing via the Cancel event) or **Dialog Accept** (when closing via a process) on the element from which the modal page is called and specify the JS Code property there.

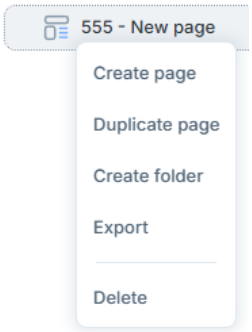
The processing of the result of a modal page execution can also be defined when the modal page is called in the onClose, onAccept, or onDecline blocks:

```
jsAPI.modal.open({
  page: {
    src: '1010',
    query: {
      P1010_ID: jsAPI.getItem('P1010_ID'),
      clear: '1010'
    }
  },
  title: 'My modal page'
}, {
  onAccept: function(e, i) {
    jsAPI.setItem('P1010_ID', i.P1010_ID);
  },
  onClose: function(e, i) {
    alert(3);
  }
});
```

Duplicate, delete, group and export a page

In addition to creating a page, the following actions are available in the page list from the context menu by right clicking the page list:

- Duplicate page - creates a copy of the selected page. For a new page, one should specify a new name, sequence number and view.
- Create folder - creates a folder with a specified name, which is used to group pages.
- Export - exports the page as a *.sql file with SQL code.
- Delete - deletes the selected page.

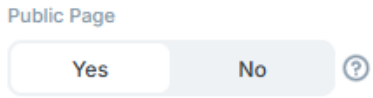


7.3.6 Page security

Find below the mechanisms for securing and protecting pages.

Public page

A public page is a page that does not require user authentication and authorization. You can set the public attribute in the page settings via the ‘Public page’ property.



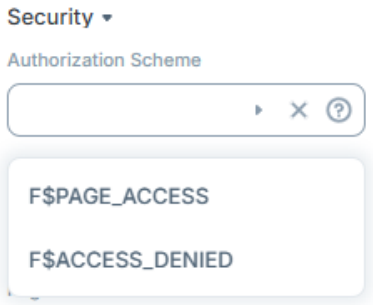
If the "Public page" property is not set or is set to "No", the user authentication page is displayed when the page is requested.

This property will be applied to the page after the changes are saved via the "Save" button.

Authorization schemes

You can determine whether a page is available to a specific user in the page editor via the “Security - authorization schemes” property.

In the drop-down list, one can see the available authorization schemes.



The selected authorization scheme will be applied to the page after saving changes via the "Save" button.

If the user does not have permissions to view the page, the error message specified in the authorization scheme will be displayed.

Checksums

A checksum is used when it is necessary to protect page parameters in a URL from modification. The checksum is generated for each link for a specific user. Changing the parameters in the URL will cause a checksum error.

If the page is checksum protected, an additional GET parameter “cs” appears in the URL:

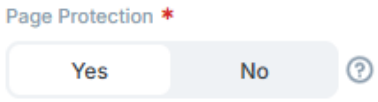
```
&cs=4825bc2e8de876595d5a36bccc89891b415e2575f45833cc5df786c000e24308
```

The checksum consists of:

- a set of page parameters,
- special character sets called salt. The salt is generated for each user session.

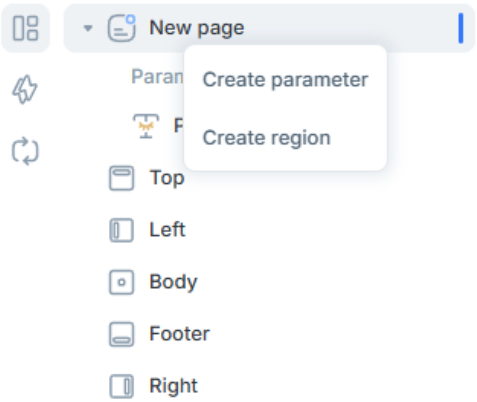
If the user manually provides a link to a page that is protected by a checksum, the system knows the link and adds the checksum to it.

Page protection via checksum is configured separately for each page.



7.4 Visual components

Visual components are the key elements of the PGHS user interface. Most web application pages consist of many components that the developer groups into containers. In XRAD, the universal container is the **region**. Regions can contain input fields, buttons, and other regions, or they can directly display data in the form of reports, charts, and other ways of presenting information. Regions default to the WRAPPER type and are the basic components of a page. You can convert a region to another visual component by changing its type. New regions are created via the context menu by right clicking.

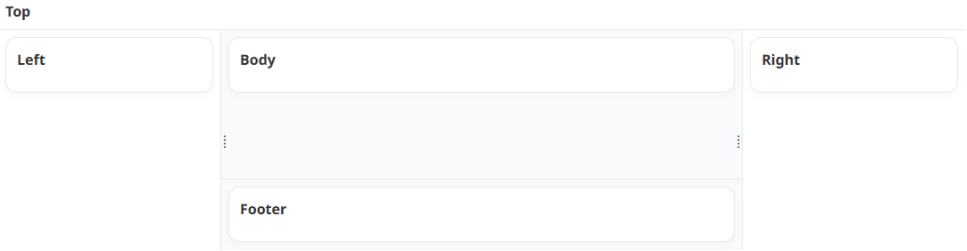


7.4.1 Location of components on the page

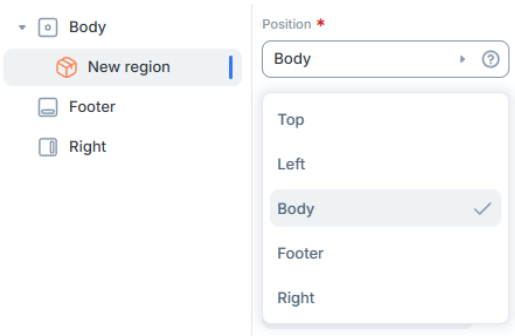
The position on the page is a mandatory parameter for a region.

XRAD provides a choice of five location options:

- Top. This is a convenient place to place "the Breadcrumbs" region or another variant of the header.
- Left. It is a good option to place the navigation panel, object tree, or filter form.
- Body (center). Here are the regions with the main contents of the page: reports, input forms, etc.
- Right. Convenient for placing an additional toolbar.
- Footer. The area for placing general information. The buttons are located here on the modal window view pages.

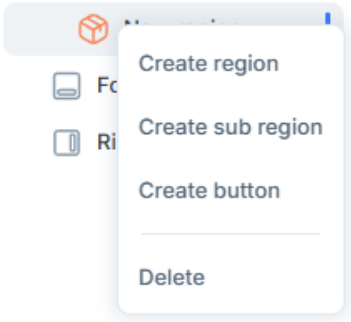


Once created, the region can be moved to another area of the page by changing its ‘Position’ property in the parameters.



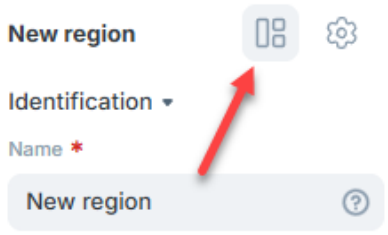
7.5 Regions

Regions can be parent, child or independent in relation to each other. Many other regions can be created within a region - child regions, regions with unlimited nesting, as well as buttons.



7.5.1 Parameters of regions

After creating a region, one needs to customize its settings.



The table in this section lists all the parameters of the regions. Depending on the type of the region, their set may differ.

Parameter	Type	Description
Name	Text	Region Name. Defines how the region is displayed in the object tree and other component settings, it is also displayed in the region header on the application page.

Table 2 - continued from previous page

Parameter	Type	
Type	List	Region Type. Defines a set of parameters and the presence of special attributes of the region, displaying the region on the application page.
Data source	List	The name of the data source for this component.
Sequence	Number	The sequence number of the region. Specifies the position of the region on the grid within the location/parent region.
Parent region	List	In the list you can select from a page and regions that are not children of this one. If a page is selected, the region will be displayed inside the position area, otherwise inside the selected region.
Position	List	Selects the location area of the region.
Column Span	Number	Select Automatic or number 1-12. Specifies the number of grid cells occupied by the region. The default is 12.
Start new line	Switch	Determines the vertical position on the grid.
Theme settings	Window Settings	Customizing the region theme.
Component classes	Text	Classes of child components within a region.
CSS classes	Text	Region css classes.
Show Header	Switch	Determines whether the region name is displayed in the header and the visual area of the header (important for button position - "in header").
Collapsible	Switch	Determines whether the region will be collapsible.
Expanded	Switch	Determines the state of the region when the page is opened. If selected, the region will be displayed in the expanded state. The parameter is only available if "Folded"="yes".
Store state	Switch	Determines whether the tab selected by the user is saved. When entering the page, the last tab selected by the user will be displayed. The parameter is available only if "Tabbed"= "yes".
Static identifier	Text	Specifies an identifier for the region for further reference from jsAPI.
Source - Type	List	Specifies the type of the data source. Applies to regions of type HTML. On two choices: • Static. You will need to enter HTML text in the source text field. • SQL. In the SQL field you will need to enter a query to the database.
Source – HTML code	Text area	HTML code. Applies to HTML region type with the selected source type - Static.
Source - SQL	Text area	Database query. The output format is determined by the region type and its attributes. The parameter is applicable for the following types of regions: HTML (with selected SQL source type) • REPORT • TREE • CHAT • CHART • CALENDAR • DATAGRID

Table 2 - continued from previous page

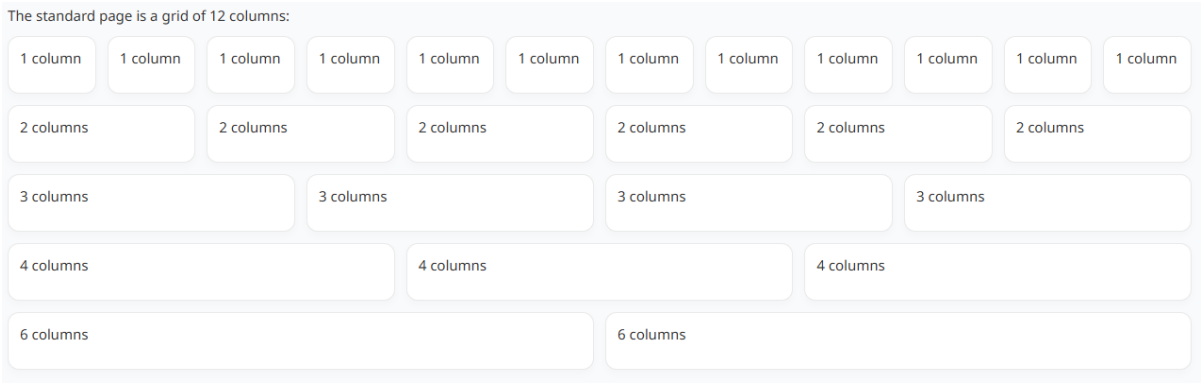
Parameter	Type	
Source - Source input	Text / Builder window	Specifies the list of input parameters for SQL query in the Source - SQL field. Applies to region types for which the Source - SQL parameter exists (see the line above). An input parameter must be defined for each substitution variable in the query. Can accept values of global variables and page input and selection elements. Can be entered as text or selected in the builder.
Source - List	List	Binds the list to a region. All lists created in the application will be presented for selection. The option applies to the following types of regions: • BREADCRUMB • PAGE NAVIGATION • CARDS • WIZARD • TILES
Display Conditions - Type.	List	Specifies the type of condition for displaying the region on the page. The default - Always. Depending on the type, you will need to specify additional condition parameters. No additional conditions will be required for Always and Never.
Display Conditions - The first condition	Text area	Query in SQL format. The parameter is applicable for the following types of conditions: • Exists (SQL query returns at least one row). If the query returns at least one row, the region will be displayed on the page. • NOT Exists (SQL query returns no rows). If the query did not return any rows, the region will be displayed on the page.
Display Conditions - SQL Expression	Text area	A logical expression in the SQL language. If the expression returns true, the region will be displayed on the page. The parameter applies to the condition type.
Display Conditions - The first input	Text / Builder window	Specifies a list of input parameters for the SQL query in the “First Condition” or “SQL Expression” field. An input parameter must be defined for each substitution variable in the query. Can accept values of global variables and page input and selection elements. Can be entered as text or selected in the builder. Applies to condition types: • Exists (SQL query returns at least one row). • NOT Exists (SQL query returns no rows). • SQL Expression.

continues on the next page

Display Conditions - Element	Text / Builder window	Allows you to select an item depending on the value of which the region will be displayed or not. Applies to the following types of conditions: • Value of Item / Column in Expression 1 Is NOT NULL. The region will be displayed if the item value is not NULL. • Value of Item / Column in Expression 1 != Zero. The region will be displayed if the item value is not equal to 0. • Value of Item / Column in Expression 1 Is NULL. The region will be if the item value is NULL. • Value of Item / Column in Expression 1 Is NULL or Zero. The region will be displayed if the item value is NULL or zero. • Value of Item / Column in Expression 1= Zero. The region will be displayed if the value of item is 0. • Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero. The region will be displayed if the item value is not NULL and is not equal to zero.

7.5.2 Arrangement of components on the grid

Visual components of the page - regions and form elements (input, selection and form buttons) are arranged in a grid of 12 columns. The grid for top-level regions (with the page as the parent) defines the position of the region within the location area; for nested regions, the grid defines the position within the parent region, as it does for form elements.



In XRAD, the position of the component on the grid is controlled by the following parameters:

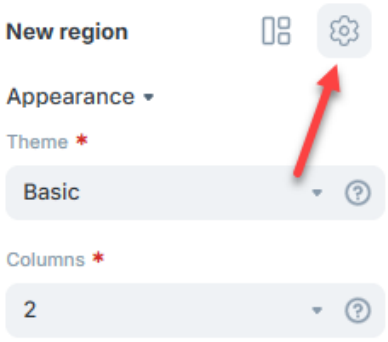
- Column Span. Accepts a value from 1 to 12, or Automatic. Specifies the number of grid cells occupied by the component. The default value is 12 for regions and Automatic for form elements.
- Start New Row. A switch that takes the value true / false. Indicates whether the component should occupy the specified number of cells from the left starting from a new row, or whether it should continue the row following the previous component in an order determined by the sequence property.

No more than 12 cells can be filled in one grid row. If the sum of column ranges for components without row transfer exceeds 12, a rendering error will be displayed when trying to open the page.

In case the sum of the components of one row is less than 12, the remaining cells will be filled if at least one region per row has Automatic in the number of columns, otherwise there will be empty space.

7.5.3 Attributes of the regions

For some types of regions, one must also specify component-specific attributes:

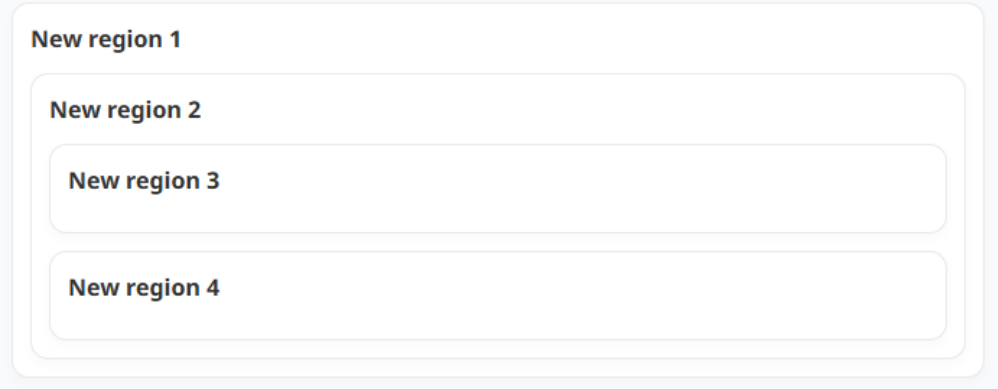


7.6 Components

7.6.1 WRAPPER

WRAPPER is the main container for the regions. It is used for grouping and zoning components or child regions.

The main parameters of WRAPPER type regions are position on the page and display of the region name. Regions of this type can be used to make blocks of any complexity and nesting.



Component customization

Component parameters are standard for the region.

7.6.2 BREADCRUMB

BREADCRUMB is a component to output a list-based navigation chain and page title.

To create the component, follow the steps below:

1. Select a page.
2. Create a new region in the "top" position.
3. Specify the region type as BREADCRUMB.
4. Create a list with type BREADCRUMB in the Lists section.
5. Specify the created list in the Source - List field.

Creating a list item

To display a navigation chain on a component, one needs to add the corresponding page element to a list of Breadcrumb type bound to a region

In the BREADCRUMB type list, create a new element.

In the element settings, enter:

- Parent item - the previous node of the navigation chain,
- Name - the displayed title on the page,
- Page - binding to a page to display a header on it,
- Link - link to a page from the navigation chain.

For more information about working with the lists, see ‘Managing lists’ section.

Component customization

Apart from the need to specify the source list, the other parameters of the BREADCRUMB component are standard for the region. For more details on the complete list of region parameters, see ‘Region parameters’ section.

7.6.3 BUTTONS

BUTTONS is a component for placing buttons.

Component customization

For this component all parameters are standard for the region.

Find below more details on the buttons.

The buttons can be placed in the body of any container, on a form, and in the headers and footers of any regions and are used to call a process, event, or jump to another page, as well as to display drop-down lists.

Button parameters

Parameter	Type	Description
Name	Text	Button name
Data source	List	Name of the data source for this component
Name	Text	Text on the button
Sequence	Number	Specifies the order of the button on the parent region within the location and position.

continues on the next page

Table 3 - continued from previous page

Parameter	Type	Description
Parental region	List	Specifies the region in which the button will be displayed
Location	List	Method for determining the position of the button. For all parent regions, except the form, has one value to choose from - REGION. There are three options in total: • REGION. Default option. The button is placed within the region at the selected position in the specified order relative to the other buttons of the position. • ITEM. Applicable with a parent region of type FORM. The position of the button is determined relative to the linked element in the specified order among other buttons of the same position of the same linked element. • FORM. Applicable with a parent region of type FORM. The button is placed on the form, the order is determined by the order among other form elements
Position	List	Specifies the relative position of the button. Applicable for options REGION and ITEM locations. The following options are available for the REGION location: • Bottom Fluid. The button will stretch to its full width along the bottom border of the region. • Bottom Left. The button will be placed under the main content of the region on the left side. • Bottom Right. The button will be placed under the main content of the region on the right side. • Left Body. The button will be positioned to the left of the main content in the region body. • Right Body. The button will be positioned to the right of the main content in the region body. • Top Left. The button is located on the left side of the region header in front of the region name. The button is only displayed if the "Show header" region setting is enabled. • Top Right. The button will be placed on the right side of the region header. It will only be displayed if the region setting is set to "Show Header". For the ITEM location, there are two options: • Left of Item. The button will be attached to the left border of the input item. • Right of Item. The button will be attached to the right border of the input item.
Action	List	Behavior when the button is pressed. There are four options to choose from: • Submit Page. Sends the form to the server, passes a static button identifier to the process equal to the parameter. • Redirects to Page. directs to the page specified in the link field. • Defined by Dynamic Action. Causes the Click event. • Open Dropdown list. Press the button to open a text menu, the contents of which are determined by the linked list.

continues on the next page

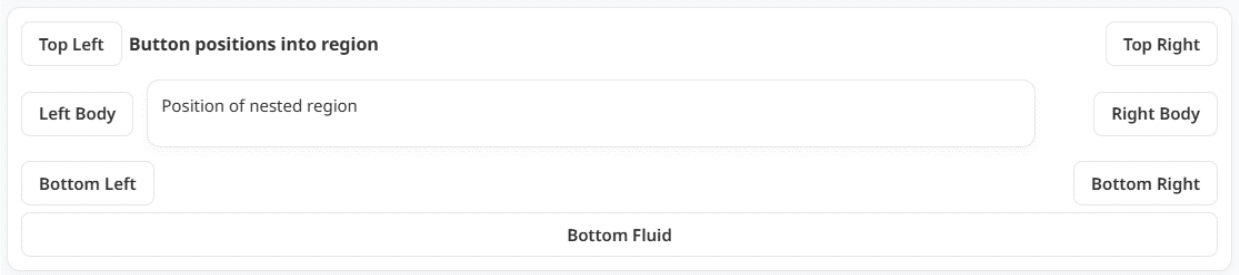
Table 3 - continued from previous page

Parameter	Type	Description
Link	Text / Builder window	Allows you to specify a link to another page, call a dynamic script, or customize in the builder a link to a page within the application. Applicable for “Redirect to Page” action.
Source - List-	List	Specifies the list displayed in the context menu. Applies to "Open drop-down list" action.
Classes	Text	A list of the CSS classes of the button. This defines the button's color, icon, etc.
Static identifier	Text	Button identifier passed to the REQUEST variable when the button is pressed. on it.
Display Conditions - Type	List	Specifies the type of button display condition. The default is Always. For depending on the type, you will need to specify additional condition parameters. No additional conditions will be required for Always and Never.
Display Conditions - The first condition	Text area	Query in SQL format. The parameter applies to the following types of conditions: - Exists (SQL query returns at least one row). If the query returns at least one row, the button will be displayed. • NOT Exists (SQL query returns no rows). If the query returns no rows, the button will be displayed.
Display Conditions - SQL-Expression	Area text	Logical expression in SQL language. If the expression returns true, the button will be displayed. The parameter applies to the SQL condition type Expression.
Display Conditions - The first	Text / Builder window	Specifies a list of input parameters for the SQL query in the “First Condition” or “SQL Expression” field. An input parameter must be defined for each substitution variable in the query. Can accept values of global variables, input elements, and page selections. Can be entered as text or can be selected in the builder. Applies to condition types: • Exists (SQL query returns at least one row). • NOT Exists (SQL query returns no rows). • SQL Expression
Display Conditions - Element	Text / Builder window	Allows you to select an item depending on whose value the column will will be displayed or not. Applies to the following types of conditions: • Value of Item / Column in Expression 1 Is NOT NULL. The button will be displayed if the item value is not NULL. • Value of Item / Column in Expression 1 != Zero. button will be displayed if the item value is not equal to 0. • Value of Item / Column in Expression 1 Is NULL. button will be displayed if the item value is NULL. • Value of Item / Column in Expression 1 Is NULL or Zero. The button will be displayed if the item value is NULL or zero. • Value of Item / Column in Expression 1= Zero. The button will be displayed if the item value is 0. • Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero. The button will be displayed if the item value is not NULL and is not equal to zero.

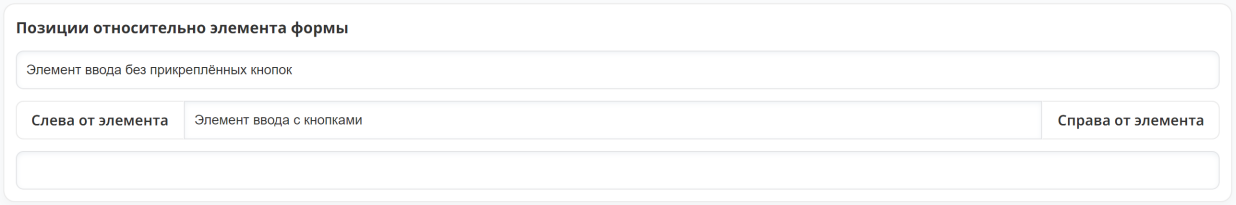
Button display

Depending on the type of location, the buttons may be displayed differently. For clarity, the possible variants are shown in the screenshots.

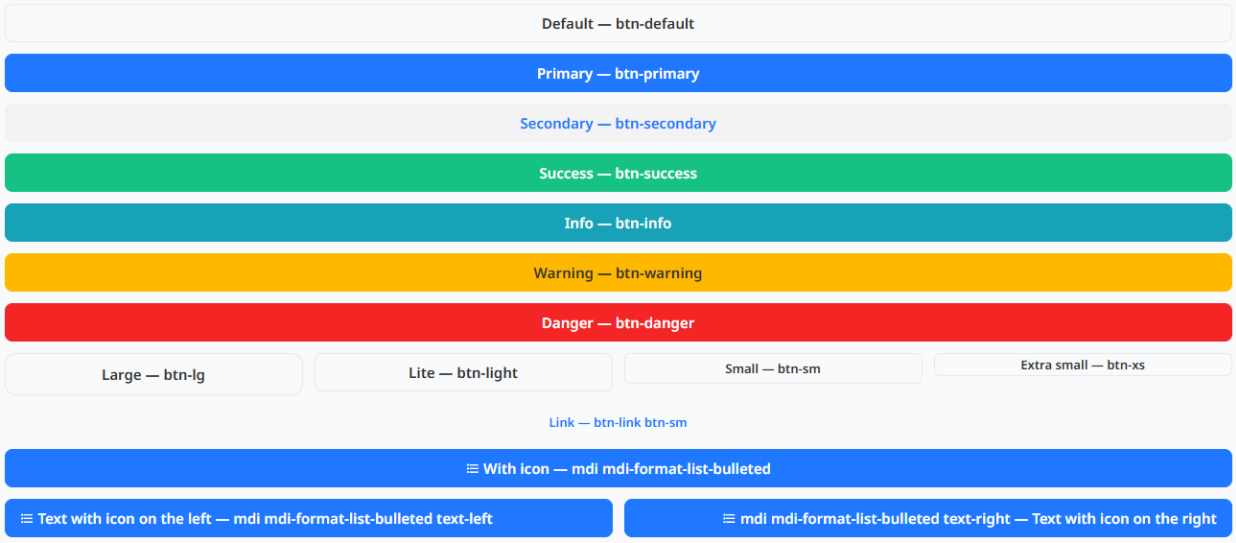
For REGION locations:



For the ITEM location:

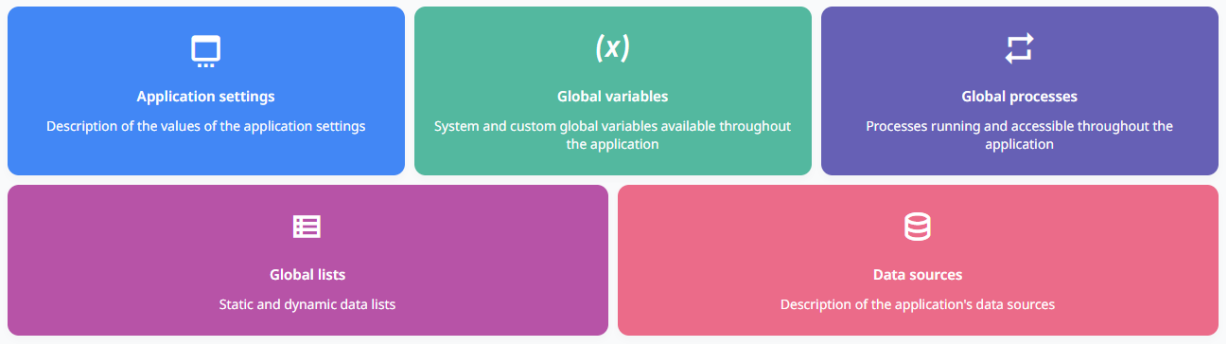


For the FORM location with a demonstration of the different classes of buttons:



7.6.4 CARDS

CARDS - region for graphical representation of the list in the form of cards. Cards have their own color, image, and a link to a page, or script.



The data source for the CARDS component is a list.

Creating a list item

To display the card on the component, one needs to add the element to the list bound to the region.

Create a new element in the list and configure the parameters:

- Name - the name of the item and the displayed title,
- Sequence - numerical order of an element that determines its position,
- CSS icon class - icon for the card,
- Link - link to a page or script when clicking on a card,
- Attribute 1 is an auxiliary inscription,
- Attribute 2 is the RGB color of the card in #fffffffff format.

Example SQL query to create a dynamic list:

```
SELECT
  'Regions' title,
  'dripicons dripicons-browser' css_icon,
  '/content/5020' target,
  'Demonstration of region types with use cases' attribute_01,
  '#FFC62D'      attribute_02
```

Note: When creating a dynamic list to use as a data source for a Card region, strict column naming as in the above example is required!

Component customization

Two levels of settings are available for the CARDS component: region parameters and attributes.

Region parameters

For this component, all parameters are standard for the region, except that the data source for the component is a list, which must be specified in the Data Source - List field.

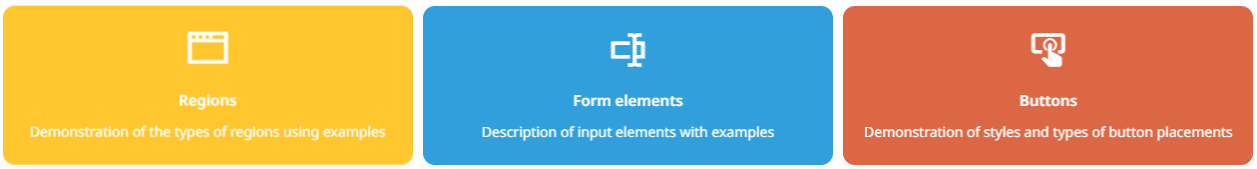
Attributes of the region

The CARDS component has two special attributes available in the component settings:

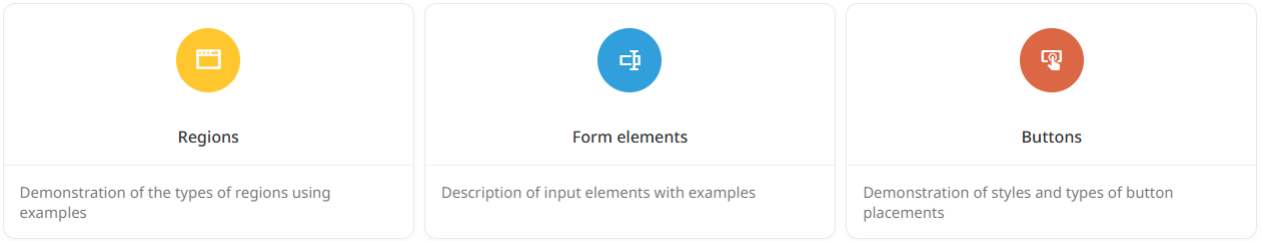
Attribute	Type	Description
Theme	List	A template that defines the display of cards. There are three options to choose from: • Block • Functional • Basic
Columns	List	The number of cards in a row. If there are more items in the list than the number indicated here, the cards will be arranged in several rows. It accepts a value from one to twelve.

For clarity, the types of templates are shown in the screenshots:

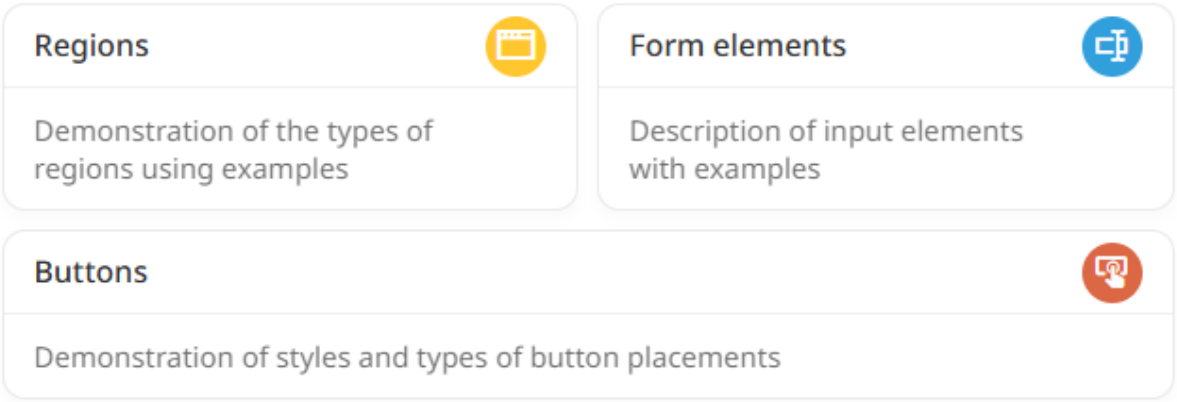
Block



Featured



Basic



X

7.6.5 CHART

CHART is used for graphical representation of numeric data in the form of diagrams. The component supports the following types of charts:

- Column - vertical

- Column - horizontal
- Ring diagram
- Line chart
- Diagram with areas

The source of data for the component is the results of SQL query.

Example of SQL query:

```
SELECT
  'Working days' day_kind,   -- Title
   247::numeric  days        -- Value
UNION ALL
SELECT
  'Non-working days',
  118::numeric
```

Component customization

Three levels of settings are available: region parameters, diagram attributes and series parameters. Find below the info on each group.

Region parameters

A distinctive feature of the region of the Chart type is the mandatory filling in the Data Source - SQL field.

Attributes of the region

A group of special attributes specific to a given type of region.

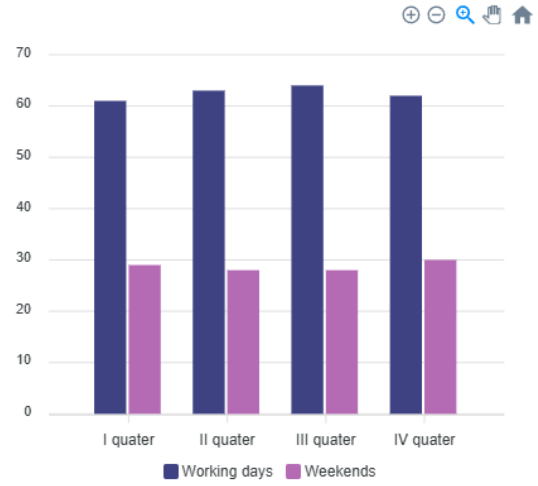
Attribute	Type	Description
Chart - Type	List	Defines the appearance of the diagram. Available options: • Bar - vertical • Bar - horizontal • Ring diagram (donut) • Line chart • Diagram with areas
Category Column	List	Determines which of the database request columns contains the value for the diagram category. The series values must be grouped by category.
Dynamic load	Switch	If set, the diagram will not be built without an explicit call from a dynamic event. Otherwise, it is loaded when the page is opened.
Width	Number	Specifies the width of the component.
Height	Number	Specifies the height of the component.

Series parameters

Parameter	Type	Description
Name	Text	Specifies the display name of the series. For the object tree and for display on the diagram in the application. Since in ring diagram charts there is only one series, for this type, the series name will only be displayed in the object tree.
Sequence	Number	Determines the order of the series when output to the interface. Since in ring diagram charts there is only one series, only the first series will be displayed in the application.
Value Column	List	From the drop-down list, select one of the request columns containing a value of numeric type to output the value to the series.
Conditions of selection - Type	List	Specifies the type of series display condition. The default is Always. Depending on the type, you will need to specify additional condition parameters. No additional conditions will be required for Always and Never.
Display Conditions - The first condition	Text area	Query in SQL format. The parameter applies to the following types of conditions: - Exists (SQL query returns at least one row). If the query returns at least one row, the series will be displayed on the chart. • NOT Exists (SQL query returns no rows). If the query returns no rows, the series will be displayed on the chart.
Display Conditions - SQL expression	Text area	A logical expression in the SQL language. If the expression returns true, the series will be displayed on the chart. The parameter is applicable for the SQL Expression condition type.
Display Conditions - The first input	Text / Builder window	Specifies a list of input parameters for the SQL query in the "First Condition" or "SQL Expression" field. An input parameter must be defined for each substitution variable in the query. It can accept values of global variables, input elements, and page selections. It can be entered as text or selected in the builder. Applies to condition types: • Exists (SQL query returns at least one row). • NOT Exists (SQL query returns no rows). • SQL Expression
Display Conditions - Element	Text / Builder window	Allows you to select an item depending on which value the column will be displayed or not. Applies to the following types of conditions: - Value of Item / Column in Expression 1 Is NOT NULL. The series will be displayed if the item value is not NULL. • Value of Item / Column in Expression 1 != Zero. The series will be displayed if the item value is not equal to 0. • Value of Item / Column in Expression 1 Is NULL. series will be displayed if the item value is NULL. • Value of Item / Column in Expression 1 Is NULL or Zero. The series will be displayed if the item value is NULL, or zero. • Value of Item / Column in Expression 1= Zero. The series will be displayed if the item value is 0. • Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero. The series will be displayed if the item value is not NULL and is not equal to zero.

For clarity, the types of diagrams are shown in the screenshots:

Columnar



Circular

Linear

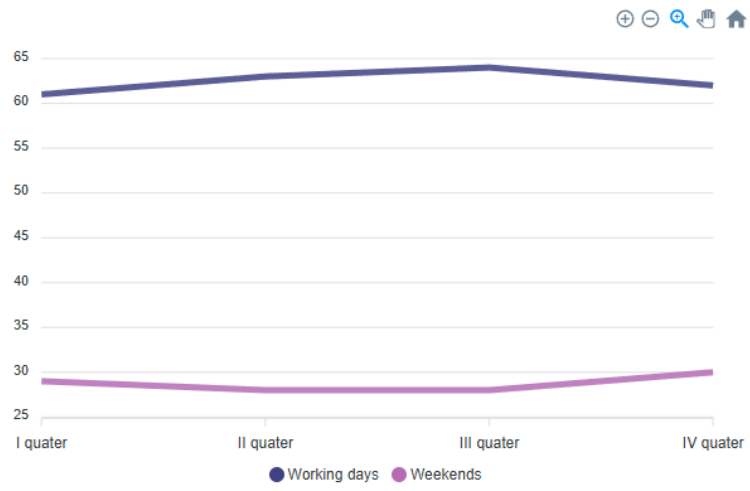
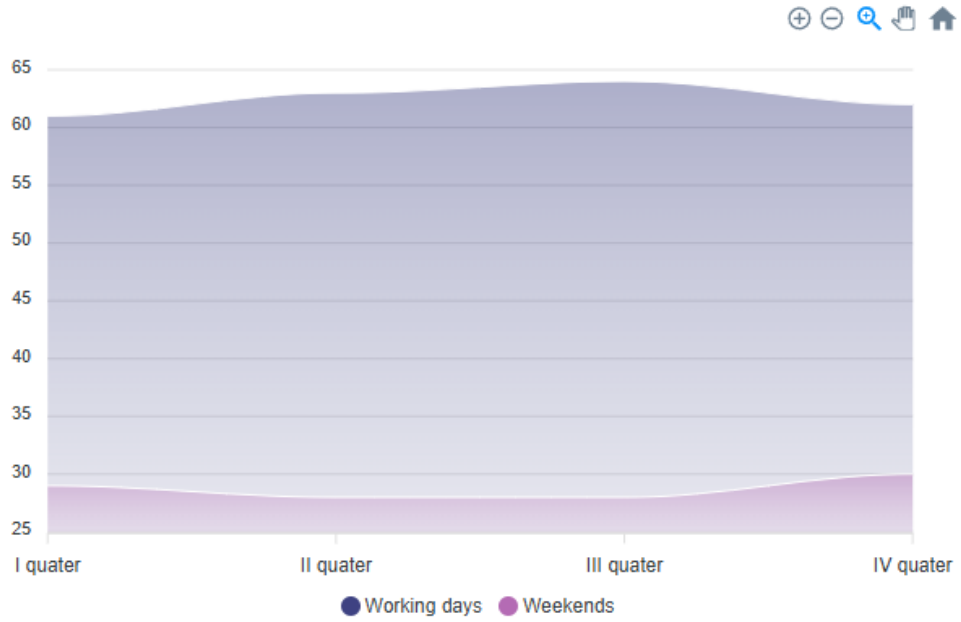


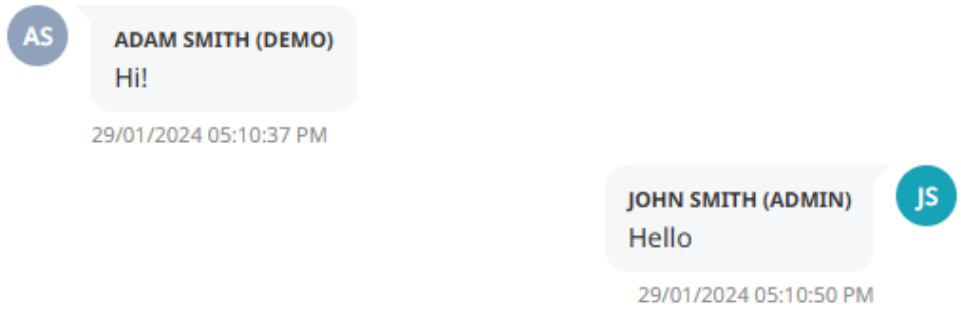
Diagram with areas



It is possible to set custom data filters by form elements to the chart. To do this, it is necessary to write the correct condition in the chart data sampling request and pass the necessary parameters that will limit the sampling.

7.6.6 CHAT

CHAT – the region for chat output. The data source for the component is the result of SQL query. Thus, the component displays the data set obtained as a result of SQL query in the form of a user chat.



In addition to the message text, date and user, the request also includes the position of the text and the design style of the icon.

Example SQL query:

```
SELECT
  cm.message_text    comment_text,   -- Message text
  TRIM(CONCAT(ul.last_name,' ',
              ul.first_name)||' ('||ul.username||')' user_name,     -- User Name
  CASE
    WHEN ul.username = UPPER($1) THEN 'right'
    ELSE 'left'
  END  align,   -- Message alignment (left, right)
  TO_CHAR(cm.message_date,'DD/MM/YYYY    HH12:MI:SS AM') comment_date,   -- Date of shipment
  SUBSTR(ul.last_name,1,1)||SUBSTR(ul.first_name,1,1)    user_icon,      -- Text in the icon
  CASE
    WHEN ul.username = UPPER($1) THEN 'bg-info text-white'
    ELSE NULL
  END icon_modifier    -- CSS icon classes
FROM
  chat.t_chat_messages cm
JOIN
  users.t_user_list ul ON cm.id_user = ul.id
WHERE
  cm.id_chat = 1
ORDER BY
  cm.message_date
```

Component customization

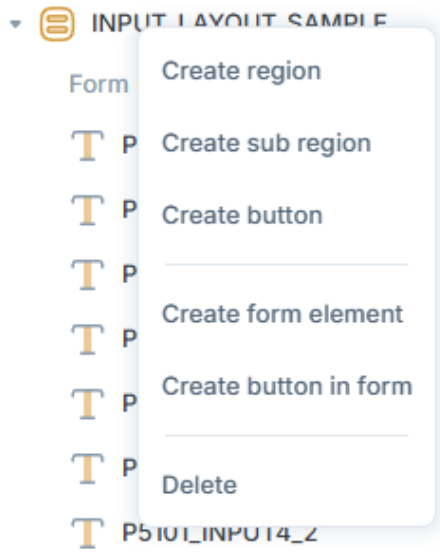
Apart from the need to select and configure the data source, the rest of the component parameters are standard for the region.

7.6.7 FORM

FORM is a component for creating a variety of user input elements.

Elements of the form

After creation on the region page with FORM type, one can add form elements. Adding elements is performed through the context menu by right click.



Basic attributes of the element

Find below the basic attributes that are present regardless of the type of input element.

Attribute	Description
Name	To identify an element, it is necessary to specify its name. The name must be unique in the context of a given page. For the correct operation of the application it is strongly recommended to specify the prefix in the format: PXXX. Where XXX is the number of the current page. When creating a new input element on the form, the name will be filled in automatically, with a correctly formed prefix. The name will also act as the element identifier in HTML.
Type	It is necessary to specify the element type. By default, it is set to element type “Text”. See below for a description of element types.
Data source	The data source for this component.
Sequence	The field defines the sequence of displaying the input element in this region. An integer value must be specified. When creating a new element, the field is filled automatically - increasing the value by 10 from the previous element.
Type	Specifies the region to which the input item belongs, you can only select a region with the FORM type.
Start a new line	Indicates the need to move the element to a new line within the region grid.
Number of columns	Specifies the number of occupied columns within the region grid.
Name	The caption of the item on the page. Displayed only if the item is displayed.
Not empty	Determines whether a value needs to be present within the element when processing the input result. If a value has not been specified, an error message will be sent to the user, requiring the user to fill in the value for the specified element.
Maximum length	Specifies the maximum possible value of the element length in characters. Valid for element types where keyboard input is expected.
Hint	The value for the tooltip that shows the expected values.

continues on the next page

Table 4 - continued from previous page

Attribute	Description
Input mask	Defines the format for entering information into the field. Use the following identifiers to format user input: • 9 - numeric symbol • a - letter symbol • *- alphanumeric character
JS Settings	Defines additional settings for the input element. See details below.
Default settings	Select the default value. The type determines where to take the value from. Available values: • Static - enter a default value that will be displayed if no session value is set for this item • SQL – enter the sql expression that will return a value for the item • Element - the value will be taken from the value of the application element recorded in the session.
Display conditions	Defines the conditions for displaying an element on a form.
Read only	Specifies the conditions for outputting an item as 'read only'.

Element types

The following element types are available for the FORM component:

Attribute	Description
Text	Element for text input of user information. Additional settings (JS Settings) include the ability to send the form to the server by pressing Enter.
Text area	Element for text input of a multiline value. It has no additional settings.
Select List	A selection item from a list of items.
Numeric	Element for numeric input. Additional settings include setting the following parameters: • Decimal part separator sign. • The digit separator sign of a number. • Minimum and maximum value of the number • Number of decimal places • Submit a form by pressing Enter When processing a value entered by the user in this field, it is necessary to convert the specified value to a numeric value using the to_number function.
Hidden	An element that will not be displayed on the form. Used to pass parameters to the form.

continues on the next page

Table 5 - continued from previous page

Attribute	Description
Date Picker	Item to select the date and time type value. Additional settings include: • Date selection option • Time selection option • Minimum and maximum date for selection • Submit a form by pressing Enter When processing a value entered by the user in this field, it is necessary to convert the specified value to the date type using the to_date/ to_timestamp function.
Radio Group	Element for displaying radio buttons. To specify a list of values, use an expression of type SQL, which will return two columns. The first column will contain the value of the radio button displayed on the form, the second column will contain the value passed to the form handler.
Check Boxes	Element for selecting values by checkboxes. As for radio buttons it is necessary to specify a list of values using an SQL expression. Additional settings: • Separator - defines the format of the delimiter of values that will be transferred to the database as a string. • Columns - the number of columns into which the checkbox values will be divided.
Switcher	An element to display a value switch. Works the same way as radio buttons, the difference is the visual design. As well as for radio buttons it is necessary to specify the list of values using SQL expression. There are no additional settings.
Autocomplete	List type element with autocomplete option. When entering a value will search through the available values in the list, filtering the result of possible values. To fill the element the end user needs to select a value from the filtered list. As well as for radio buttons it is necessary to set a list of values using SQL expression. There are no additional settings.
Password	Element for entering a password. The entered characters are automatically replaced by the * sign, at the user's request the information can be displayed by pressing the button next to the item.
WYSIWYG	The element is displayed as a WYSIWYG editor
File input	Element for uploading custom files. The element settings specify: • Block height - defines the height of the drag and drop area. • Mime types - sets a limit on the types of files that can be loaded. File types must be specified as mime-type separated by commas. If the value "any" is specified, the type restriction is removed. • Hint text - sets the hint text for the user • Max. file size - defines the maximum allowable file size • Max. number of files - defines the maximum number of files. To process uploaded files, XRAD creates a temporary table in the pg_temp schema using the following expression: CREATE TEMP TABLE IF NOT EXISTS pg_temp.xrad_files (name text, file_name text, file_mime text, file_content bytea). The name column will contain the value from the file input element passed to the server. If multiple files are uploaded, the value of the element will be passed as a string with the values separated by a colon (:).

continues on the next page

Table 5 - continued from previous page

Attribute	Description
Multiselect	A list of items with multiple values to choose from.

7.6.8 HTML code (HTML)

HTML – is a region for displaying HTML code. This type of region is used to display text, images, videos and any other HTML code elements.

A

The component supports two types of data source:

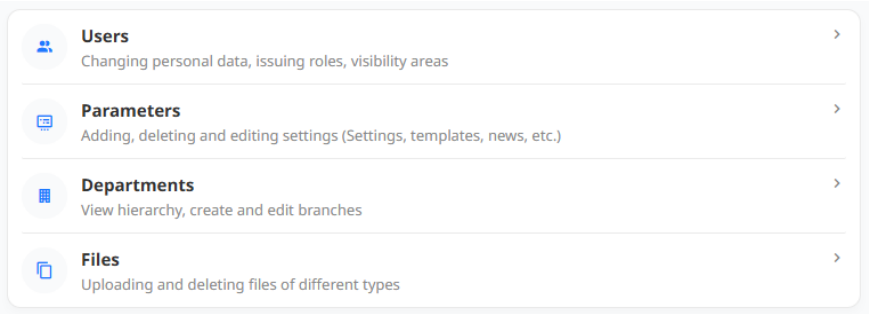
- Static - HTML code is specified directly in the HTML code parameter.
- SQL - HTML code will be obtained as a result of SQL query execution.

Component customization

Apart from the need to select a source, the rest of the component parameters are standard for the region.

7.6.9 PAGE NAVIGATION

PAGE NAVIGATION is a component for placing a navigation panel on a page. The data source is a list whose elements will be displayed with an icon, title, link and description.



Example of SQL code to create a dynamic list:

```
SELECT
    n.title name,
    'uil uil-envelope' icon,
    n.message_short attribute_01,
    '/content/9201?P9201_ID='||n.id target,
    ns.status_name attribute_02
FROM orgn.t_notifications n
ORDER BY n.date_created DESC;
```

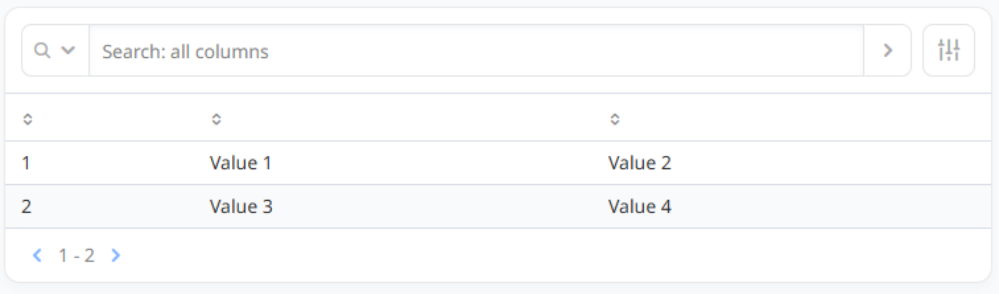
Note: When creating a dynamic list to use it as a data source for the Page Navigation region, strict column naming is required as in the above example!

Component customization

Apart from the need to specify the source list, the other component parameters are standard for the region.

7.6.10 REPORT

REPORT is a component for outputting data in the form of a table. The output data is generated as a result of SQL-query.



The parameters of the REPORT component allow to:

- set the number of lines to be displayed
- add pagination
- show/hide column headers
- add a data set search field
- set the message to be displayed when there are no rows in the sample.

The report column settings allow to:

- change the type from text to link, set an icon for the link
- add a sorting option
- include the column in the searchable list
- output mask
- set the width, alignment, and other display settings.

It is possible to set custom data filters for the report by form elements. To do this, one needs to write the correct condition in the report data selection request and pass the necessary parameters that will limit the selection.

Component customization

There are three levels of settings available: region settings, report attributes and report column settings. Find more info on each group below.

Region parameters

The distinctive feature of the component of the 'Report' type is the mandatory filling in the Data Source - SQL field. It is necessary to specify the SQL-query, based on which the list of report columns is formed and the data for output to the form is selected.

Attributes of the region

A group of special attributes specific to a given type of region.

Attribute	Type	Description
Enable search	Switch	Adds a report search element. A text input field and a drop-down list with the ability to select report columns to search by them.
Enable pagination	Switch	Adds pagination - the ability to view pages beyond the first page.
Enable headers	Switch	Adds headings to the report columns.
Template	List	Specifies the basic report template. There are currently two template options: • Standard • Column - value
String limitation	Number	Sets the number of report lines per pagination page.
Message about missing data.	Text area	Allows you to specify the text that will be displayed on the screen if the database query did not return any data.
Report templates	Window settings	Allows you to select a report template as the default template. By clicking opens a window with a list of report templates. A template is generated each time the user sets a new selection filter, sorting or highlighting. The developer can select one of these templates as the default template, after which this report view will be opened to all users.

Между Enable headers и Template добавить (англ не трогать – тех. наименования):

- Attribute: Enable actions
- Type: Switch
- Description: Добавляет кнопку "Действия" с дополнительным пользовательским функционалом в виде фильтрации, highlighting и управления колонками.

Column parameters

The generated list of columns will be displayed in the object tree immediately after successful validation of the query in the SQL Data Source field. Each column is an independent object with individual parameters. The table lists the full set of parameters of the report column component.

Parameter	Type	Description
Column name	Text (read only)	Column Name. Defined by the data source query column alias. The field is not editable in the parameter settings - it always corresponds to the column name in the query. The column value is referenced by this field.

continues on the next page

Table 6 - continued from previous page

Parameter	Type	Description
Type	List	Defines a set of column parameters and its behavior when displayed on the page. It can take the following values: • Text. Simple output of the value of information from the database. • Link. Allows clicking on an element within a cell of a column to redirect the user to another page or to call a dynamic script. • Hidden. The report column will be hidden in the user interface. Its value (e.g. to be passed to the reference parameters) will still be available • Checkbox. The column is a group of checkboxes for multiple selection of report values.
Associated Item	Text / Builder - window	Allows you to select an input element or enter a global variable, where values from cells of checkboxed rows will be transferred. The parameter is applicable for column type - Checkbox.
Heading	Text	The column header to display in the user interface on the application page.
Sequence	number	Defines the order of columns when output to the interface. By default it is equal to the order of columns in the query at the first generation. When adding a column to the query later, the new column will be assigned the last number, regardless of its position in the query.
Alignment	Switch	Alignment of the column content text. Left edge, centered, or on the right edge.
Link	Text / Builder window	Allows you to specify a link to another page, call a dynamic script, or customize in the builder a link to a page within the application. In the latter case, you can select values not only for input fields, but also for report columns. Instead of the link text, you can also pass the value of the report string - this approach allows you to customize the link inside a sql query. The parameter is applicable for Column Type - Reference.
Link icon	Text	Allows you to select an icon to display in a column cell. You can also pass a report string value instead of the link text - this approach allows you to select the picture inside a sql query. The parameter is applicable for Column Type - Reference.
Width	Text	Fixes the width of the column.
Format mask	Text	Formats the output by overlaying a mask according to the formatting rules of the to_char SQL function.
Enable sort	Switch	Allows rows to be sorted by column value.
Column search	Switch	Enables searching by column value if searching is enabled in the report attributes.
HTML Expression	Text area	Allows you to define the output format as HTML. You can refer to the value of the report column.
Display Conditions - Type	List	Specifies the type of the report column display condition. The default is Always. Depending on the type, you will need to specify additional condition parameters. For Always and Never, no additional conditions will be required.

continues on the next page

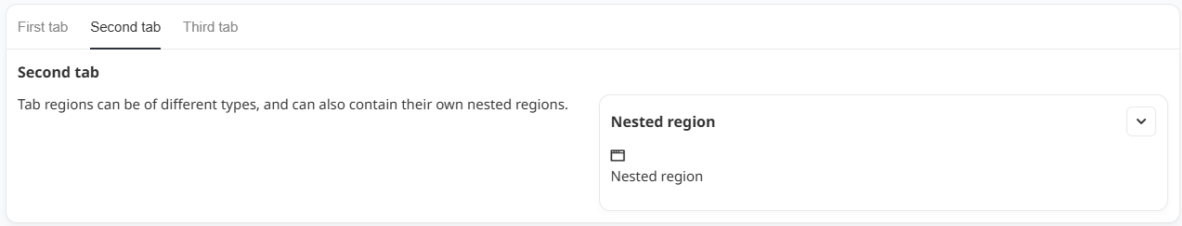
Table 6 - continued from previous page

Parameter	Type	Description
Display Conditions - First condition	Text area	Query in SQL format. The parameter is applicable for the following types of conditions: • Exists (SQL query returns at least one row). If the query returned at least one row, the column will be displayed in the report • NOT Exists (SQL query returns no rows). If the query returns no rows, the column will be displayed in the report.
Display Conditions - SQL Expression	Text area	Logical expression in SQL language. If the expression returns true, the column will be displayed in the report. The parameter applies to the condition type SQL Expression.
Display Conditions - The first input	Text / Builder window	Specifies a list of input parameters for the SQL query in the "First Condition" or "SQL Expression" field. An input parameter must be defined for each substitution variable in the query. Can accept values of global variables, input elements, and page selections. Can be entered as text or selected in the builder. Applies to condition types: • Exists (SQL query returns at least one row). • NOT Exists (SQL query returns no rows). • SQL Expression.
Display Conditions - Element	Text / Builder window	Allows you to select an item depending on whose value the column will be displayed or not. Applies to the following types of conditions: - Value of Item / Column in Expression 1 Is NOT NULL. The column will be displayed if the item value is not NULL. • Value of Item / Column in Expression 1 != Zero. The column will be if the item value is not 0. • Value of Item / Column in Expression 1 Is NULL. The column will be if the item value is NULL. • Value of Item / Column in Expression 1 Is NULL or Zero. The column will be displayed if the item value is NULL or zero. • Value of Item / Column in Expression 1= Zero. The column will be displayed if the value of item is 0. • Value of Item / Column in Expression 1 Is NOT null and the Item / Column Is NOT Zero. The column will be displayed if the item value is not NULL and is not equal to zero.

7.6.11 TABS

TABS is a container for grouping child regions on switchable tabs. After creating a component with the TABS type on the page, one needs to add child regions using the context menu. Each child region is displayed as a tab.

The tabs will display child region headers, even if the header is disabled for the regions themselves.



Component settings

Component parameters are standard for the region.

7.6.12 TREE

TREE -a region for displaying the tree structure of data. SQL query is used as a data source.

An example SQL query for the TREE component:

```
SELECT
  t.id,       -- Element ID
  t.id_fk,    -- ID of the parent element
  t.title,    -- Signature of the element
  t.link      -- The redirect link after clicking on the element
FROM
  dep.f_get_tree_with_scope(1) t;
```

Component settings

Two levels of settings are available: region parameters and tree attributes.

Region parameters

The component parameters are standard for the region.

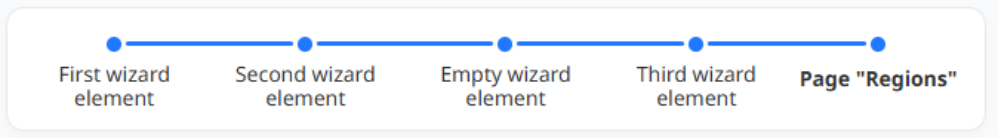
Attributes of the region

Attribute	Type	Description
Node ID column	List	Node identifier. Select one of the query columns from the drop-down list.
Parent Key column	List	Identifier of the parent node. Select one of the query columns from the drop-down list.
Node label column	List	The displayed title of the node. You must select one of the query columns from the drop-down list.
Link	Text / Builder window	The link where the redirection will be made after clicking on a tree node.
Link column-	List	Specifies the column containing the link to jump to.

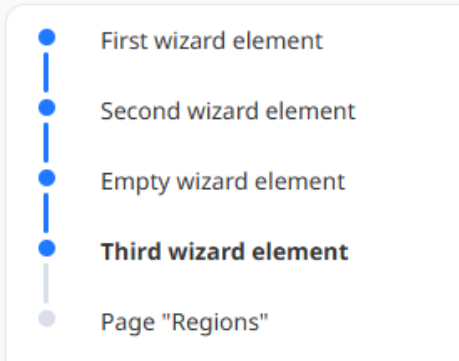
7.6.13 WIZARD

WIZARD is a component for graphical representation of a sequence of actions based on a list. The component supports two types of orientation:

Horizontal



Vertical



Component customization

Two levels of customization are available: region parameters and wizard attributes.

Region parameters

Component parameters are standard for the region.

Attributes of the region

Attribute	Type	Description
Clickable	Switch	Determines whether the linked page will be navigated to by a click on the wizard node.
Step labels	List	Several options are available for displaying wizard step headers: - All. The headers of each item in the list will be displayed on the wizard • Current. Only the header of the current step will be displayed on the wizard, the rest will be unnamed. • Never. No headers will be displayed on the wizard.
Orientation from:	List	The spatial orientation of the wizard. There are two options to choose • Horizontal • Vertical

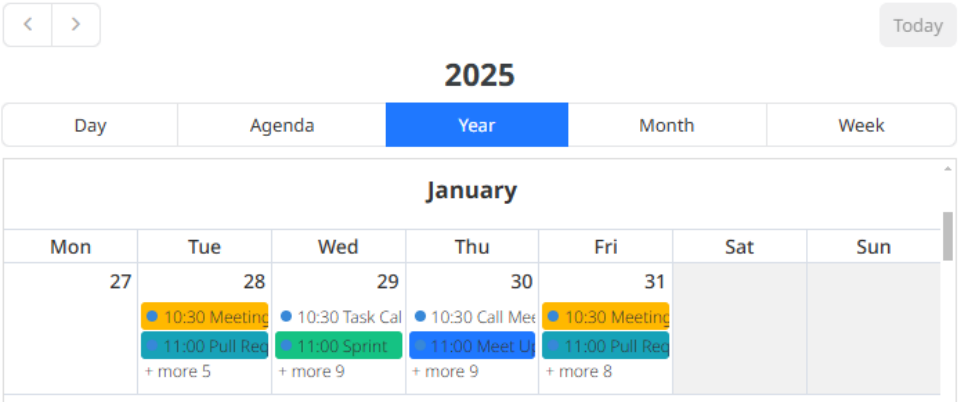
7.6.14 CALENDAR

CALENDAR is a component for displaying the data with start and end time stamp of an event in the form of an interactive calendar. The source of data for the component is the result of SQL query.

Example SQL query

```
SELECT
  c.id,           -- ID of the string to add to the link
  c.title,        -- Event title
  c.start_ev,     -- Start of event period
  c.end_ev,       -- End of event period
  c.color_class   -- HEX color code
FROM
  events.t_calendar c;
```

The calendar itself can be customized by changing the display of weekends, selecting the views to be displayed (Week, Month, Year, Agenda), as well as the calendar height and the display of a hint about the event when hovering the cursor. One can set a color to the calendar event, as well as add the redirect link (the id of the record can be added to the link).



Component customization

Two levels of customization are available: region settings and calendar attributes.

Region parameters

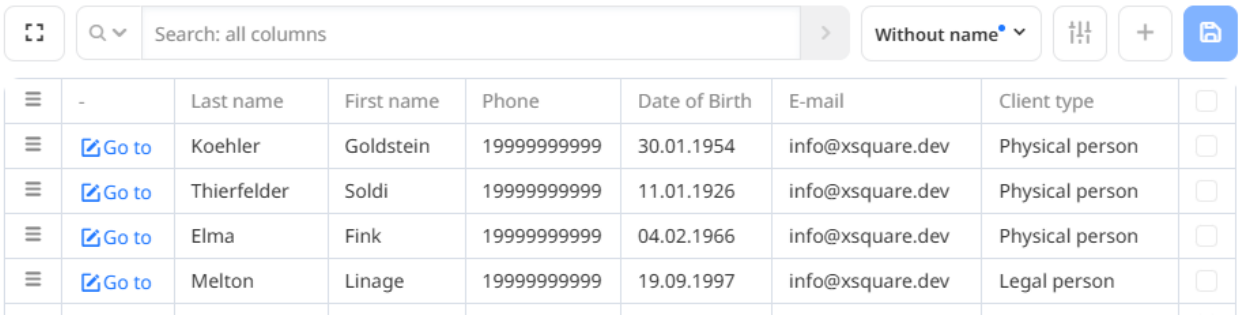
Component parameters are standard for the region.

Attributes of the region

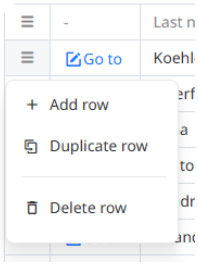
Attribute	Type	Description
Display Column	List	You can assign one of the SQL data source columns as the title of the event displayed on the calendar.
Start date Column	List	You can assign one of the SQL data source columns as the start time of the event displayed on the calendar
End date Column	List	You can assign one of the SQL data source columns as the end time of the event displayed on the calendar.
CSS column	List	One of the SQL data source columns can be assigned to define the style of event to be displayed on the calendar.
Height	number	The size of the component vertically.
Views	Multiple choice	List of available calendar view levels. The selected options will be offered to the user in the component interface. List of views: • Year • Month • Week • Day • Agenda
Show Time	Switch	If set, not only the date but also the time of the event will be displayed.
Show Tooltip	Switch	If set, the name of the event will be displayed on a tooltip. When you hover over a cell in the calendar.
Show Weekends	Switch	If set, all days of the week will be displayed.
View Link	Text / Builder window	Defines the link to which the user will be redirected when clicking on an event in the calendar.

7.6.15 DATAGRID

DATAGRID is a table with data similar to REPORT, but with the ability to edit data. Records in DATAGRID can be added, modified and deleted.



In addition, in DATAGRID you can change the sequence of columns by simply dragging the column header to the desired location. Also, existing rows can be duplicated.



Component customization

Two levels of settings are available: region parameters and table attributes.

Region parameters

Component parameters are standard for the region.

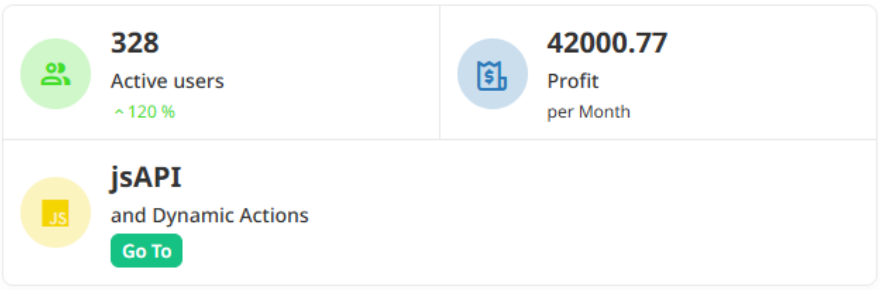
Attributes of the region

Attribute	Type	Description
Editing - enabled	Switch-	Specifies whether the contents of the datagrid can be edited.
Allowed operations	Multiple-choice	List of available operations in the table: • Insert • Update • Deletion The setting of allowed operations is available if editing is enabled.
Lost Update type	List	This attribute specifies the way in which data loss is prevented during the multiple users working together with the same data set. Available values: • String value. Before sending the changes, the system calculates the hash of the changed string and compares it with the hash obtained when loading the data. If the hashes are different, the system will not allow the update (someone else has already changed the string).
Authorization on editing	List	Specifies which user authorization scheme is used to allow to the selected operation in the component.
Show NULL values as	Text	Specifies what will be displayed in the table cell if there is no value (Null).
Show - Toolbar	Switch-	Specifies the display of the component's toolbar.
Toolbar - Controls	Multiple-choice	Defines a set of buttons on the toolbar, if its display is enabled. The following toolbar components are available: • Search columns • Search field • Actions • Save
Templates	Text / Builder window	Allows you to specify a default template.

7.6.16 TILES

TILES is used to display statistical information. The source of data for the TILES component is the result of SQL query. To customize the appearance of tiles, certain SQL query parameters are used, for example:

```
SELECT
  '42000.77'    title,      -- Main caption (title)
  'Profit'      text,       -- A caption describing the title
  '#2C7ABB'     color,      -- HEX code for tile design color
  'For the month'  descr,   -- Subsidiary caption below the main text
  'uil uil-bill'  icon      -- CSS icon class
```



Component customization

Two levels of settings are available: region parameters and table attributes.

Region parameters

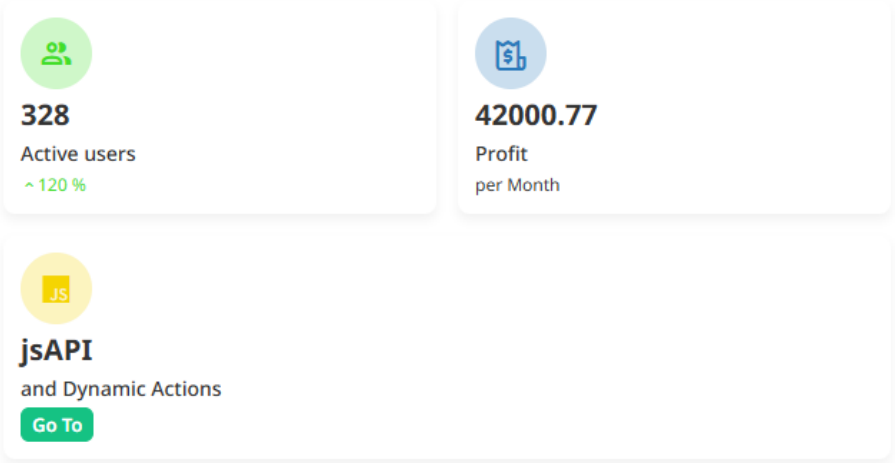
The distinctive feature of the component of the “Tiles” type is the mandatory filling in the Source - SQL field. It is necessary to specify the SQL query, based on which the list and appearance of tiles are formed.

Attributes of the region

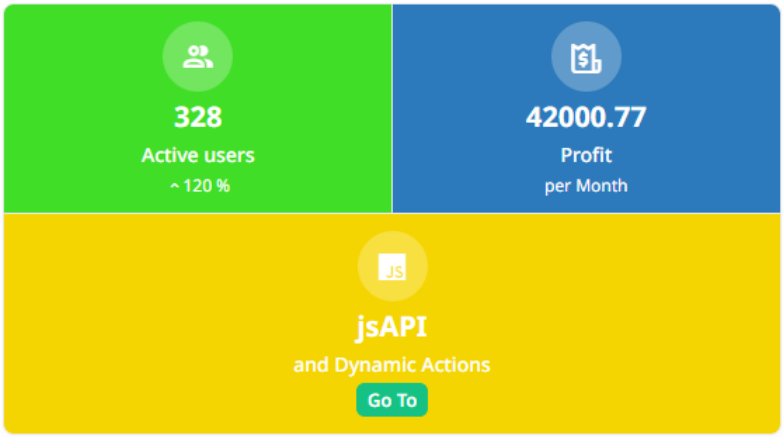
Attribute	Type	Description
Theme	List	Specifies a predefined theme for displaying tiles: • Classic - displays tiles with the icon on top. The color setting in this variant determines the color of the icon. • Contrast - similar to classic, but the color setting determines the background color of the tile. • Horizontal - compact display with the icon on the left. The color setting determines the color of the icon. • Horizontal-contrast - a compact display where the color setting determines the background of the tile
Columns	List	Specifies the grid settings for displaying the component. When specified parameter "auto" the component will distribute all elements into rows by itself. The width of each element in the row will be the same. Parameter "auto-float" works the same as "auto", but the width of each element will match the content of the component.
View	List	Defines the appearance of the component. • A grid is a flat display with no gaps between tiles. • Spaced - display tiles with gaps and shadows
Align	Switch	Specifies the type of alignment of the content in the tile: • Left align • Centered • Right align
Contrast Icons	Switch	Specifies the contrast mode of the display. In contrast mode, icons are displayed in white on a colored background.
Title column	List	Specifies the name of the parameter to display the tile header.
Text column	List	Defines the name of the parameter for displaying the main text of the tile.
Description column	List	Specifies the name of the parameter for displaying additional tile text.
Icon column	List	Specifies the name of the parameter to display the tile icon.
Color column	List	Specifies the name of the parameter to control the tile color. It is necessary to specify the color value in hex format.

Several different themes are implemented for the tiles, similar to the CARDS component. You can also select different alignment options for the text within a tile: left, right and center. In addition to tile display themes, there are also two types of tile layout: as a solid grid of elements and as individual elements. For clarity, the possible options are shown in the screenshots:

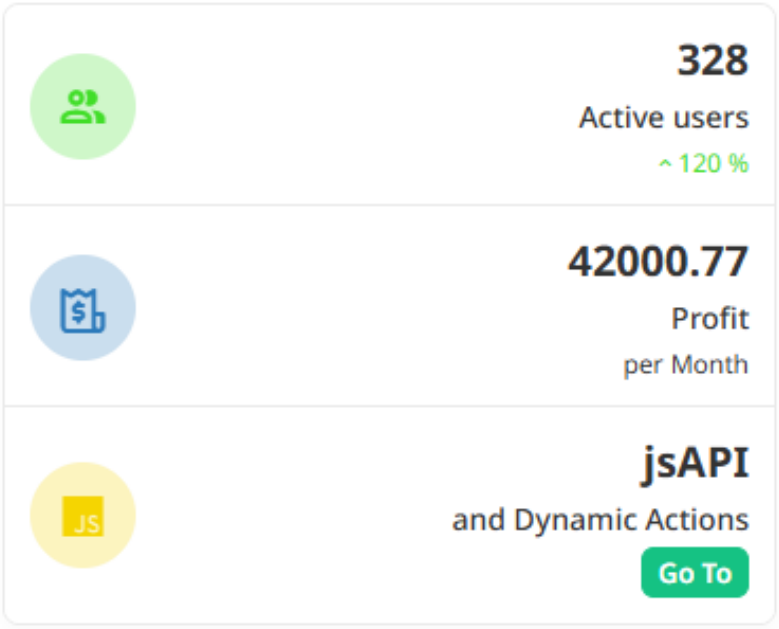
Classic 3 column theme with spaced tiles



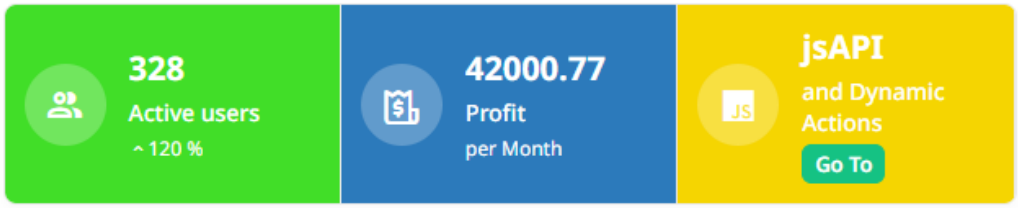
Contrast theme in 2 columns with center alignment



Horizontal 1-column theme with right alignment



Horizontal-contrast theme with contrasting icons



7.7 Working with lists

Such elements as menus, path to the current page (breadcrumb), navigation bars, etc. are often used in the applications being developed. To display information in them, you can use both a dynamically generated set of values (using SQL query) and a static - strictly defined list.

7.7.1 What is a list

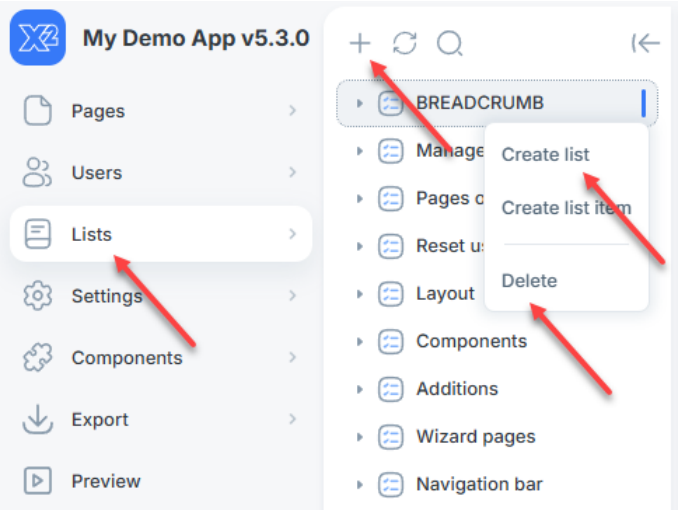
A list is a set of data of the same type.

There are 3 types of lists in XRAD:

1. Static - the list is created and edited by the developer in the XRAD environment.
2. Dynamic (Based on Query) - the list is created based on an SQL query. It can be changed by the user through the web application interface.
3. Breadcrumb is a special type designed to output the path to an application page.

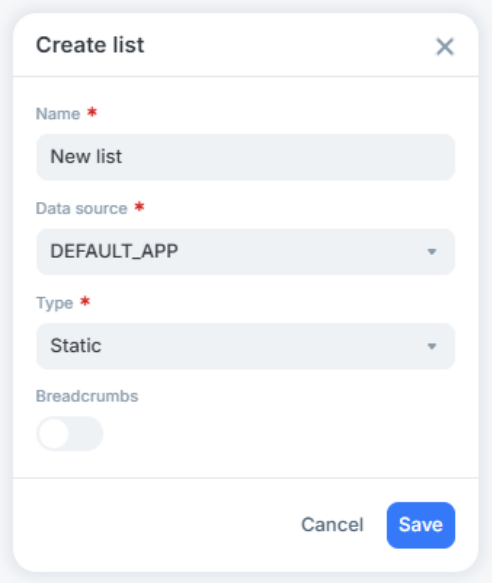
7.7.2 Creating, editing, deleting lists

To create a list, select the "Lists" section in the main menu and press the "+" button in the submenu. Also, to create or delete a list, you can use the context menu by right clicking on the area where the lists are shown and selecting the appropriate action.

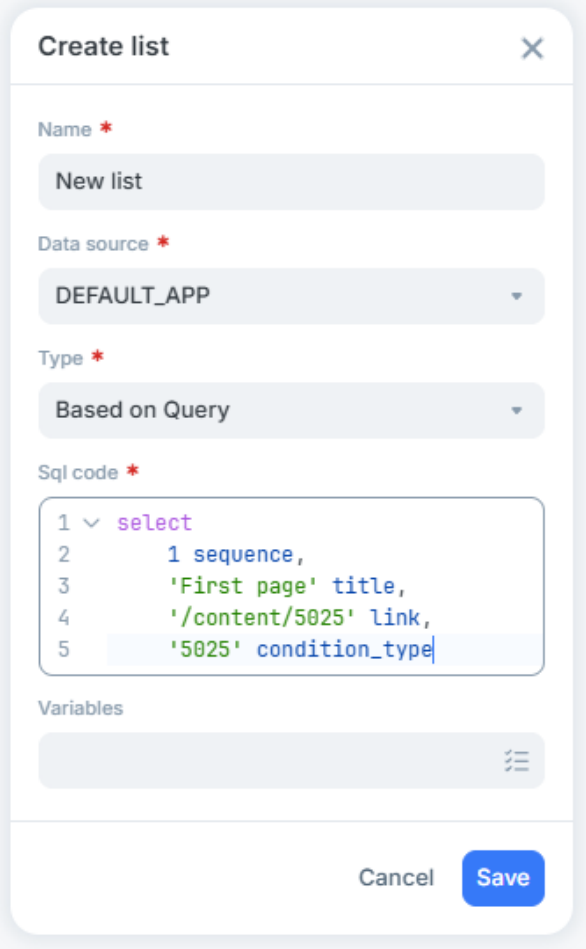


The creation of a new list takes place in a pop-up window where one needs to specify:

- Name is a mandatory attribute that defines the name of the list.
- Data source is a mandatory attribute that defines the database for storing the list.
- List Type - Static, Based on Query, Breadcrumb.



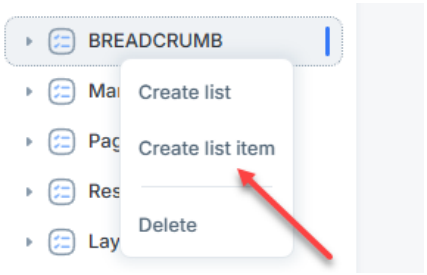
When creating a dynamic list (Based on Query), one must specify the SQL query on the basis of which the list will be built and, if necessary, the variables that are used in the query.



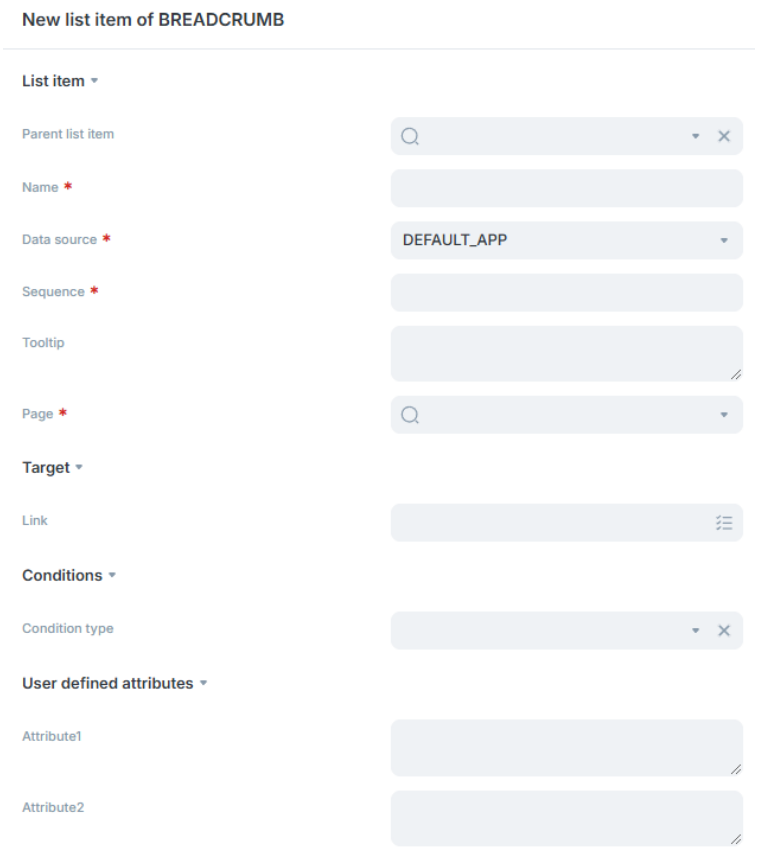
After clicking the "Save" button, a new list will be created to which you can add list items.

7.7.3 Creating list items

To create list items, one needs to right-click to call the context menu and select "Create list item".



In a new tab, fill in the attributes of the new list item.



Parent item - parent item of the list. It is intended for formation of multilevel menu.

- Name is a mandatory attribute that defines the name of the list item.
- Data Source is a mandatory attribute that defines the source of the data.
- Order number is a mandatory attribute that determines the order in which the element is displayed in the list.
- Css icon class - defines the icon to be displayed.
- Tooltip - defines the text tooltip of the element.
- Active for pages - list of page numbers on which this list item will be highlighted as active.
- Separated - Inserts a separator between items if the list is used to form a navigation menu at the top of the page.

- Link - actions that will be performed when selecting a list item
- Condition type - defines the type of condition that must be met to follow the link.

Attributes 1, 2 - for some output options one can also use additional list attributes. For example, for Page Navigation component in Attribute 1 one can specify what will be displayed under the item name, and in Attribute 2 - what will be displayed in the right part of the item.

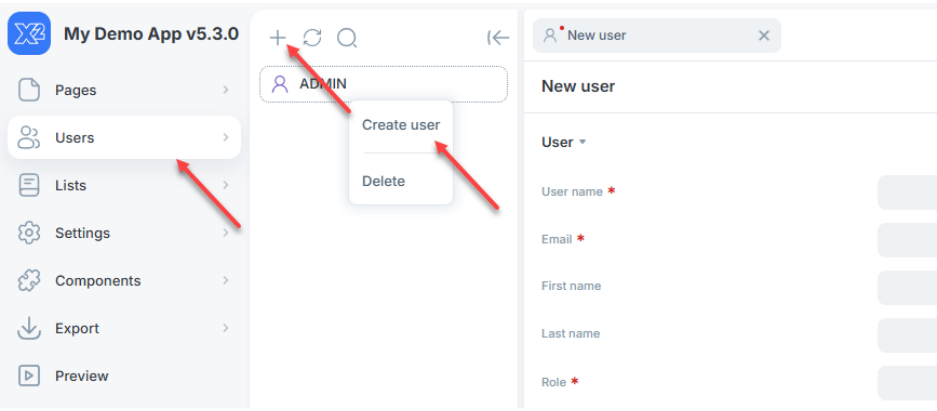
7.8 User management

XRAD provides the ability to manage users directly in the development environment. Internal users of the development environment are authenticated by login and password (CUSTOM scheme); or are matched by name with external users who are authenticated according to the chosen scheme. For example, a user named DEVELOPER can log in with a username and password, or can be matched with a DEVELOPER user from Active Directory or OIDC. Internal XRAD users have a role that defines access rights. The processes for creating and editing users are minimalistic but have all the necessary functionality. Three main roles are defined, which can be used to differentiate access zones for users:

- **ADMIN** is an administrator role. The ADMIN has full rights, so special care should be taken when granting this role. A user with this role can edit, create and delete other users. The ADMIN has the right to create another user with the ADMIN role. This role should be assigned strictly to an administrator, the DEVELOPER role is sufficient for developers.
- **DEVELOPER** is a developer role. The DEVELOPER can view and modify Pages, create and delete them. The DEVELOPER has the right to edit Lists and Settings, and has the right to view Users, but does not have the right to change them (including their own user). This role is suitable for developers, as the user with this role has access to all tools except User management.
- **VIEW** is a role with a minimum set of privileges. The VIEW users can view Pages and their contents, as well as Lists and Settings (but not their contents!). Users with this role cannot view Users and cannot create Pages, neither change their user settings. This role fits for the users who want to familiarize themselves with XRAD.

7.8.1 Creating, editing, deleting users

To create a user, select the "Users" section in the main menu and press the "+" button in the submenu. Also, to create or delete a user, you can call the context menu by right clicking on the area where the users are listed and select the appropriate action.



After filling in or editing the parameters of the selected account, confirm the changes by pressing the "Save" button.

Parameter	Description
Username	A mandatory attribute specifying the name of the user to be created, used as a login.
Email	A mandatory attribute that specifies the user's e-mail address.
First name, Last name	Attributes for entering the first and last names of the user.
Role	A mandatory attribute that defines one of the possible user roles.
Locked	A flag that specifies whether the user is locked out of the system.

It is mandatory to specify and confirm the user password, and one can also check the "Require password change" flag if it is necessary to change the password after the initial login. Note: when editing user parameters, changing the 'Username' field is not available.

7.9 Styles and themes

A developer can control the standard styles through custom CSS properties.

You can connect your CSS properties in two ways:

1. In the XRAD development environment, in the page settings section, add CSS properties to the "Embedded CSS" field (as local properties).
2. Add CSS between the <head></head> tags in the index.html file of the PGHS web controller (as global properties).

Global custom PGHS properties are applied to the pseudo-class: root, local (within a page) properties are applied to the class. page-{ID}, where {ID} is the page identifier.

Attention

Specify only those variables that need to be overridden. The default values are already defined. There is no need to copy the entire example to the page

If only the primary color is to be overridden, it is sufficient to specify the value -color-primary.

```
:root {
   --color-primary: #1f78ff;
 }
```

A list of PGHS custom CSS properties and an example of usage:

```
:root {
   --color-primary: #1f78ff; /* Primary color*/

   --color-danger: #f42525; /* "Error" status color (field highlighting, tooltips) */

   --color-warning: #ffb800;  /* Color of the "warning" status (tooltips) */
↪
   --color-success: #15c283;  /* Color of "success" status (tooltips) */
   --color-info: #17a2b8;     Status color "informing" (tooltips)*/

   --color-link: #1f78ff;  link color, inherited from --color-primary */

   --global-bg-color: #f9fafb; /* Page background color */
```

(continued on next page)

(continued from previous page)

```
  /* Buttons */
  --btn-radius: 8px; /* Button rounding */
  --btn-lg-radius: 12px; /* Button rounding with modifier btn-lg (large) */
  --btn-sm-radius: 6px; /* Button rounding with modifier btn-sm (small) */
  --btn-xs-radius: 6px; /* Button rounding with modifier btn-xs (extra small) */
  /* .btn */
  --btn-color: transparent; /* Button background color .btn (standard - without modifier)␣
↪ colors) */
  --btn-border-color: #eaeaea; /* Background color of the .btn button (standard - without
  modifier␣
↪ colors) */
  --btn-text-color: #333; /* Button text color .btn (standard - no modifier␣
↪ colors) */
  /* .btn-primary */
  --btn-primary-color: #1f78ff; /* Button background color .btn-primary, inherited from --
↪ color-primary */
  --btn-primary-text-color: #fff; /* Button text color .btn-primary */
  /* .btn-secondary */
  --btn-secondary-color: #f3f3f4; /* Background color of .btn-secondary button */
  --btn-secondary-text-color: #1f78ff; /* Button text color .btn-secondary,␣
↪ inherited from --color-primary */
  /* .btn-success */
  --btn-success-color: #15c283; /* Background color of button .btn-success, inherited from
  --
↪ color-success */
  --btn-success-text-color: #fff; /* .btn-success button text color */
  /* .btn-danger */
  --btn-danger-color: #f42525; /* Background color of .btn-danger button, inherited from -
  -
↪ color-danger */
  --btn-danger-text-color: #fff; /* .btn-danger button text color */
  /* .btn-warning */
  --btn-warning-color: #ffb800; /* Button background color .btn-warning, inherited from --
↪ color-warning */
  --btn-warning-text-color: #333; /* .btn-warning button text color */
  /* .btn-info */
  --btn-info-color: #17a2b8; /* Background color of button .btn-info, inherited from --
  color-info */
  --btn-info-text-color: #fff; /* .btn-info button text color */

  /*Load Indicator*/
  --loader-color: #1f78ff; /* Loading indicator color --color-info, inherited from -
↪ -color-primary */
  --loader-width: 80px;
  --loader-width-sm: 1rem; /* Loading indicator size .spinner-border-sm */

  /* Fonts */
  --font-family: "Noto Sans", sans-serif; /* Name, font family */

  /* Typography */
  --text-title-color: #333; /* Standard title color */
  --text-color: #333; /* Standard text color */
  --text-font-size: 15px; /* Standard text size */
  --text-font-weight: 400; /* Standard text boldness */
```

(continued on next page)

(continued from previous page)

```
    /* Page scrolling */
    --scroll-color: #bfbfbf; /* Scroll indicator color */
    --scroll-bg-color: transparent; /* Scroll indicator background color */

    /* Grid */
    --region-grid-col-gap: 12px; /* Indents between columns (col) */
    --regions-grid-row-gap: 12px; /* Indents between rows */

    /* Page */
    --page-position-border: 1px solid #eaeaea; /* border (separator line) between the␣
↪ page positions: sidebar navigation menu), top, left, right, body (center), footer */
    --resizer-hover-color: #1f78ff; /* Color of resizer (width control␣
↪ block) when hovering */

    /* sidebar                                  */
    --sidebar-width: 250px; /* Width of sidebar (navigation menu) */
    --sidebar-border: 1px solid #eaeaea; /* border (separator line) sidebar␣
↪ (navigation menu), inherited from --page-position-border */
    --sidebar-bg: #fff; /* Background color of sidebar (navigation menu) */
    --sidebar-text-color: #4b5667; /* Sidebar (navigation menu) text color */
    --sidebar-font-size: 12px; /* Sidebar (navigation menu) font size */
    --sidebar-active-bg: #f6f7f9; /* Background color of the active menu item in the
    sidebar(navigation menu) */
    --sidebar-active-border-color: #1f78ff; /* Background color of the border on the right of
    the active menu item in the sidebar (navigation menu), inherited from --color-
    primary*/
    --sidebar-active-bg-radius: 4px; /* Rounding the active menu item in the sidebar␣
↪ (navigation menu) */
    --sidebar-icon-color: #4b5667; /* Color of menu item icons in sidebar␣
↪ (navigation menu), inherited from --sidebar-text-color */
    --sidebar-drp-icon-color: #88888888; /* Child open/close icon color␣
↪ menu item elements in the sidebar (navigation menu) */

    /* left
    --page-left-position-width: 250px; /* Left position width, inherited from --
↪ sidebar-width */
    --page-left-position-padding: 8px; /* Internal left position offset */
    --page-left-position-border: 1px solid #eaeaea; /* border (separator line)␣
↪ position-left, inherited from --page-position-border */
    --page-left-position-bg-color: #fff; /* Background color of the left position */

    /* body (center) */
    --page-body-position-padding: 8px; /* Internal offset of body position (center) */

    /* right */
    --page-right-position-width: 250px; /* Right position width, inherited␣
↪ from --sidebar-width */
    --page-right-position-padding: 8px; /* Internal offset of position right */
    --page-right-position-border: 1px solid #eaeaea; /* border (separator line)␣
↪ position-right, inherited from --page-position-border */
    --page-right-position-bg-color: #fff; /* Background color of the right position */
```

(continued on next page)

(continued from previous page)

```
    /* top */
    --page-top-position-padding: 8px; /* Internal offset of top position */
    --page-top-position-border: 1px solid #eaeaea; /* border (separator line)␣
↪ top position, inherited from --page-position-border */
    --page-top-position-bg-color: #fff; /* Background color of the top position */

    /* footer */
    --page-footer-position-padding: 8px; /* Internal footer position offset */
    --page-footer-position-border: 1px solid #eaeaea; /* border (separator line)␣
↪ footer (bottom) position, inherited from --page-position-border */
    --page-footer-position-bg-color: transparent; /* Background color of footer position (bottom)␣
↪*/→

    /* header for page types: Standard, minimalist */
    --header-bg-color: #1f78ff; /* Header header) background color, inherited from --color-
↪ primary */
    --header-text-color: #fff; /* Header text color */
    --header-height: 48px; /* Header height */
    --header-menu-btn-border-color: transparent; /* Navigation button stroke color␣
↪ menu */
    --header-controls-border-width: 1px; /* Navigation menu button border size␣
↪*/→
    --header-menu-btn-bg-color: rgba(
      0,
      0,
      0,
      0.1
    ); /* Navigation menu button background color */
    --header-controls-radius: 6px; /* Navigation menu button rounding */
    --app-logo-text-font-size: 15px; /* Application name font size */
    --app-logo-text-font-weight: 600; /* Boldness of the application name font */
    --app-logo-text-line-height: 20px; /* App-logo-text-line-height */
    --app-logo-display: none; /* (none / block) Display logo (to the right of the button␣
↪ navigation menu) */
    --app-logo-url: none; /* (none / url("/files/images/my_logo.svg")) Picture␣
↪ logo (to the right of the navigation menu button), applied as background-image */
    --app-logo-size: 32px; /* Logo size (to the right of the navigation menu button),␣
↪ is applied as background-size */
    --app-logo-position-top: 0; /* Logo indentation (to the right of the navigation button␣
↪ menu) on top, applied as background-position (top) */
    --app-logo-position-left: 0; /* Indent the logo (to the right of the navigation button␣
↪ menu) left, applied as background-position (left) */
    --navbar-icon-radius: 6px; /* Navigation menu icon block rounding */
    --navbar-icon-bg-color: rgba(
      0,
      0,
      0,
      0.1
    ); /* Background color of the navigation menu icon */
    --navbar-icon-color: #fff; /* Navigation menu icon color, inherited from --
↪ header-text-color */
```

(continued on next page)

(continued from previous page)

```
    --navbar-icon-order: 3; /* Display order (flex order) for the icon␣
↪ navigation menu */
    --navbar-icon-margin: 0 0 0 0 0 8px; /* External indentation of navigation menu icon */
    --navbar-text-order: 2; /* Display order (flex order) for item text␣
↪ navigation menu */
    --navbar-arrow-order: 1; /* Display order (flex order) for the item icon␣
↪ drop-down list of the navigation menu (down arrow) */

    /* Regions */
    --region-bg-color: #fff; /* Region background color  */
    --region-border-color: #eaeaea; /* Region stroke color */
    --region-border-width: 1px; /* Region-border-width */
    --region-box-shadow: 0px 4px 7px 0px 0px rgba(0, 0, 0, 0, .02); /* Region shadow */
    --region-padding: 12px; /* Internal region indents */
    --region-radius: 12px; /* Region rounding */
    --region-head-separated-gap: 12px; /* Indent the bottom of the separated header␣
↪ region (in region settings), inherited from --region-padding */

    /* Modal windows */
    --modal-box-shadow: 0px 18px 30px 0px 0px rgba(51, 51, 51, 51, 0.64); /* Modal shadow␣
↪ windows */
    --modal-radius: 20px; /* Modal window rounding */
    --modal-controls-btn-icon-color: #888888; /* Icon color in the modal header, right side␣
↪ from the header (close button) */

    /* Form Elements (ITEMS) */
    --item-placeholder-color: #888; /* Label color */
    --item-text-color: #333; /* Text color, inherited from --text-color */
    --item-border-radius: 8px; /* Rounding */
    --item-box-shadow: 0px 2px 5px 0px 0px rgba(85, 114, 157, 0.11) inset; /* Shadow */
    --item-border-color: #e4e5e7; /* Color of stroke */
    --item-focus-color: #1f78ff; /* Color of the stroke in the :focus state, inherited from
    --
↪ color-primary */
    --item-error-color: #f42525; /* Error state stroke color and text color␣
↪ errors-under-field, inherited from --color-danger */
    --disabled-items-bg-color: #f9fafb; /* Background color in :disabled state,␣
↪ :readonly */
    --disabled-items-text-color: #aaa; /* Text color in :disabled, :readonly state␣
,*/→
    --item-control-inside-icon-color: #aaaaaa; /* Icon color inside the element (tooltip,␣
↪ selector arrow,etc.)*/.

    /* Checkboxes */
    --checkbox-border-color: #eaeaea; /* Checkbox stroke color */
    --checkbox-checked-bg-color: #1f78ff; /* Background color of the selected checkbox */
    --checkbox-icon-color: #fff; /* Color of the selected checkbox icon  */
    --checkbox-radius: 4px; /* Checkbox rounding */

    /* Radio buttons */
    --radio-border-color: #eaeaea; /* Radio button stroke color */
    --radio-checked-bg-color: #1f78ff; /* Background color of the selected radio button */
    --radio-icon-color: #fff; /* Color of the selected radio button icon */
```

(continued on next page)

(continued from previous page)

```
    --radio-radius: 100%; /* Radio button rounding */


    /* Switch (switcher) */
    --switcher-bg-color: #f3f3f4; /* Switch background color*/
    --switcher-separator-color: #d9dfe8; /* Separator background color (when no item is
    selected) */
    --switcher-radius: 6px; /* Switcher rounding */
    --switcher-btn-radius: 4px; /* Switch button rounding */
    --switcher-btn-active-bg-color: #fff; /* Background color */
    --switcher-btn-active-box-shadow: 0px 2px 8px 0px 0px rgba(0, 0, 0, , 0.08); /*    */
    --switcher-btn-text-color: #333; /* button text color, inherited␣
↪ from --text-color */
    --switcher-active-btn-text-color: #333; /* The text color of the toggle button in the␣
↪ active state, inherits from --text-color */
    --switcher-lg-radius: 8px; /* Switcher rounding (in the absence of a label) */
    --switcher-lg-btn-radius: 6px; /* Switch button rounding (if no label) */

    /* Dropdown (dropdown list, control) */
    --dropdown-bg-color: #fff; /* Background color */
    --dropdown-border-color: #eaeaea; /* Color of the stroke */
    --dropdown-radius: 8px; /* Rounding */
    --dropdown-box-shadow: 0px 3px 6px 0px rgba(0, 0, 0, 0.05), 0px 11px 11px 0px
        rgba(0, 0, 0, 0.04), 0px 25px 15px 0px rgba(0, 0, 0, 0.03);  /* Shadow */
    --dropdown-active-bg: #f6f7f9; /* Background color of the active menu item */
    --dropdown-active-border-color: #1f78ff; /* The background color of the border on the
    right side of the active menu item, inherited from --color-primary */
    --dropdown-active-bg-radius: 4px; /* Rounding of the active menu item */

    /* Calendar (ITEM type DATE) */
    --datepicker-days-text-color: #aaa; /* */
    --datepicker-days-text-color: #333; /* , inherited from --text-color */
    --datepicker-active-bg-color: #1f78ff; /* Background color of active date, inherited␣
↪ from --color-primary */
    --datepicker-active-text-color: #fff; /* Active date text color */
    --datepicker-hover-bg-color: rgba(
      45,
      52,
      62,
      0.06
    ); /* Background color of hover date */

    /* Field File */
    --item-file-radius: 10px; /* Rounding */
    --item-file-border-color: #888; /* Color of stroke */
    --item-file-placeholder-color: #888; /* Text color in the drag zone␣
↪ (dropzone) */
    --item-file-msg-color: #888; /* Explanatory message text color below the field */

    /* Tabs (TABS) */
    --tabs-padding: 0 12px 0 12px; /* Internal tab block indentation */
    --tab-padding: 12px 0 12px 0; /* Inner indentation of tab-switch button */
```

(continued on next page)

(continued from previous page)

```
    --tabs-gap: 24px; /* Spacing between radio buttons */
    --tabs-border-width: 1px; /* Tab block border width, inherited from --region-
↪ border-width */
    --tabs-border-color: #eaeaea; /* Tab block border color, inherited from --
↪ region-border-color */
    --tabs-color-border-active: #2d343e; /* Active tab border color */
    --tabs-color-text: #888; /* Switch buttons text color */
    --tabs-color-text-active: #333; /* Active toggle button text color */

    /* Cards (CARDS) */
    --cards-radius: 12px; /* Rounding, inherited from --region-radius */
    --cards-border-color: #eaeaea; /* Border color, inherited from --region-border-
↪ color */
    --cards-border-width: 1px; /* Border width, inherited from --region-border-
↪ width*/
    --cards-box-shadow: 0px 4px 7px 0px 0px rgba(0, 0, 0, 0, .02); /* Shadow, inherited from --
↪ region-box-shadow*/
    --cards-bg-color: #fff; /* Background color */

    /* Report (REPORT) */
    --report-th-text-color: #888; /* Text color of <th> cells in <thead> */
    --report-th-font-size: 14px; /* Font size of cells< th> to< thead> */
    --report-th-line-height: 16px; /* Cell line height< th> to< thead> */
    --report-th-font-weight: 400; /* Boldness of cell font< th> to< thead */>
    --report-td-text-color: #333; /* Text color of <td> cells in <tbody> , inherited from␣
↪ --text-color */
    --report-td-font-size: 14px; /* Font size of td<> cells in tbody<> */
    --report-td-line-height: 16px; /* Cell line-height td<> to tbody<> */
    --report-td-font-weight: 400; /* Boldness of cell font td<> to tbody */<>
    --report-cellpadding: 8px 12px; /* Internal cell indents */
    --report-border-width: 1px; /* Border-width */
    --report-border-color: #d9dfe8; /* Border color */
    --report-stripe-bg-color: #f9fafb; /* Background color of even-numbered lines */
    --report-acs-desc-active-color: #333; /* Active sort icon color,␣
↪ inherits from --report-td-text-color */
    --report-hover-bg: #d8e7ee; /* Background color of line on hover */
    --report-pagination-text-color: #888; /* Pagination text color */
    --report-pagination-button-color: #1f78ff; /* Pagination button color, inherited␣
↪ from --color-primary */
    --report-pagination-font-size: 14px; /* Pagination font size*/
    --report-pagination-line-height: 16px; /* Pagination-line-height */
    --report-footer-padding: 8px 12px; /* Internal pagination block indentation, ␣
↪ inherits from --report-cellpadding */
    --report-header-padding: 12px; /* Internal indentation of top block with search and␣
↪ filters, inherited from --region-padding */

    /* Step-by-step navigation (WIZARD) */
    --wizard-unactive-bg-color: #d9dfe8; /* Background color of inactive/unfinished␣
↪ states */
    --wizard-active-bg-color: #1f78ff; /* Background color of active/complete␣
↪ states, inherited from --color-primary */
    --wizard-step-label-color: #333; /* Label text color, inherited from --text-
```

(continued on next page)

(continued from previous page)

```
↪ color */
  --wizard-step-counter-color: #888888; /* Color of text inside the inactive point/
↪ incomplete state */
  --wizard-step-active-counter-color: #fff; /* Color of text inside the active point/
↪ completed state */
}
```

7.10 jsAPI reference guide

The jsAPI object serves as a tool for interacting the client side of the application with the server and the PGHS frontend, and aims to simplify the development of the visual interface of the application, provides ready-made DOM tools, wrappers for standard javascript functions, and much more. The jsAPI is installed by default and is already in the PGHS global scope (window). To view all methods and objects, one needs to call the jsAPI object from the debugger (jsAPI, console.log(jsAPI), console.dir(jsAPI)).

7.10.1 jsAPI: submit()

Performs page processing (submit) using the {API_URL}/processPage method.

Syntax

```
jsAPI.submit(items, callbacks);
```

Parameters

Items object - the values *items* are sent in the body of the request {APIURL}/processPage in addition to the main items of the page. It is allowed to use an empty {} object if no custom values need to be sent.

Callbacks object accepts onSuccess Function, onError Function.

Note: the custom onSuccess handler will not trigger a page reload (overriding the default behavior), if a page reload and additional logic is required in the custom handler, the jsAPI.reload() method within onSuccess is used.

Examples

Standard call:

```
jsAPI.submit({});
```

Call with additional items values and a successful send and error handler:

```
jsAPI.submit(
   {
     MY_CUSTOM_ITEM: "CUSTOM VALUE",
   },
   {
     onSuccess: function (res) {
        console.log("The custom handler will not reload the page");
     },
     onError: function (err) {
        console.error(err); // Service error handler {API_URL}/processPage
     }
   }
);
```

Call without additional items values and with a successful send handler:

```
jsAPI.submit(
   {},
   {
     onSuccess: function (res) {
       //Any code can be placed here
       jsAPI.reload(); //And after it the page reload will be executed
     }
   }
 );
```

7.10.2 jsAPI: process()

The jsAPI.process method executes the xhr request {API_URL}/callAction with the following parameters:

```
{
   action: "PROCESS",
   data: {
     page: pageID,
     request: requestName,
     items: items
   }
 }
```

pageID (data.page) is the ID of the current page. It is determined automatically. requestName (data.request) The name of the request. It is defined by the argument requestName items (data.items). Items values are defined by the items argument.

Syntax

```
jsAPI.process(requestName, items, callbacks);
```

Parameters

- requestName [String] - Request name
- items [Object] - the items values sent in the body of the {API_URL}/callAction request. It is allowed to use an empty {} object if no values are to be sent.
- callbacks [Object] - accepts functions: onSuccess Function, onError Function.
- onSuccess [Function] - callback, contains server response (res)
- onError [Function] - callback, contains server error (err)

Examples

Call without any additional parameters:

```
jsAPI.process("MY_PROCESS");
```

A call with items:

```
jsAPI.process("MY_PROCESS", {
   P1000_MY_ITEM_1: "Value 1",
   P1000_MY_ITEM_2: "Value 2".
 });
```

Call with items and a successful response handler:

```
jsAPI.process("
   MY_PROCESS",
   {
     P1000_MY_ITEM_1: "Value 1",
     P1000_MY_ITEM_2: "Value 2"
   },
   {
     onSuccess: function (res) {
       console.log(res);
     }
   }
 );
```

Call with items, success and error handlers:

```
jsAPI.process("
   MY_PROCESS",
   {
     P1000_MY_ITEM_1: "Value 1",
     P1000_MY_ITEM_2: "Value 2".
   },
   {
     onSuccess: function (res) {
       console.log(res); // Server response
     },
     onError: function (err)
       { console.error(err); // Server error
     }
   }
 );
```

7.10.3 jsAPI: reload()

Renders the page after a successful response to the `{API_URL}/showPage?page=id` method.

Syntax

```
jsAPI.reload();
```

Parameters

This function accepts no arguments.

Note is not similar to window.location.reload(). The method calls page re-rendering after successful response of {API_URL}/showPage method, as it is done when changing the root.

Examples

```
jsAPI.reload();
```

7.10.4 jsAPI: reloadWindow()

A wrapper for the native method window.location.reload() Reloads the application on the current page.

Syntax

```
jsAPI.reloadWindow();
```

Parameters

This function accepts no arguments.

Examples

```
jsAPI.reloadWindow();
```

7.10.5 jsAPI: redirect()

Navigates to the specified application page.

Syntax

```
jsAPI.redirect(path, callbacks);
```

Parameters

- path [String] - Relative path to the page
- callbacks - accepts the function: onSuccess [Function]

Note This method uses internal application routing and is not similar to window.location.href. Changing the routing does not involve reloading the page.

Examples

Standard call:

```
jsAPI.redirect("/content/5000?clear=5000");
```

Call with handler:

```
jsAPI.redirect("/content/5000?clear=5000", {
   onSuccess: function (data) {
     console.log(data);
   }
 });
```

7.10.6 jsAPI: getCurrentPage()

Returns information about the current page in the form:

```
{
    "pageId": "1000",
    "ELEMENT_STATES": {
         ...
     },
    "CLIENT_VALIDATION": {
         ...
     },
    "VALIDATION": {
         ...
     },
    "items": {
```

(continued on next page)

(continued from previous page)

```
        ...
    },
    "page": {
        ...
    }
}
```

Syntax

```
jsAPI.getCurrentPage();
```

Parameters

This function accepts no arguments.

Examples

Get the page id:

```
jsAPI.getCurrentPage().pageId;
```

Get item values:

```
jsAPI.getCurrentPage().items["P1000_MY_ITEM"];
```

7.10.7 jsAPI: notification

Includes success, error, and warning methods that output the application standard notification.

Notification of successful action

```
jsAPI.notification.success(message); //Displays notification of successful action
```

Error notification

```
jsAPI.notification.error(message);//Displays error notification
```

Warning (warning)

```
jsAPI.notification.warning(message); //Displays a warning (warning)
```

Parameters

- message String, Array - one message if the argument is specified as a string, or an array of messages if the argument is specified as an array of strings (output sequentially one after another)

Examples

Successful action message:

```
jsAPI.notification.success("Successful action");
```

Successful Action Messages:

```
jsAPI.notification.success(["Successful Action", "Successful Action 2"]);
```

Error message:

```
jsAPI.notification.error("Error");
```

Error Messages:

```
jsAPI.notification.error(["Error", "Error 2"]);
```

Warning (warning)

```
jsAPI.notification.warning("Warning!"); //Displays a warning
```

Warnings

```
jsAPI.notification.warning(["Warning", "Warning 2"]); //Displays warnings
```

7.10.8 jsAPI: modal

jsAPI.modal is a module for managing application modal windows.

Methods

```
jsAPI.modal.open(options, callbacks);
```

Parameters

- options Object - Options for displaying the dialog window. It contains the parameters:
- title String - Window title
- page String - URL of the page that will be displayed in the dialog box
- width Number - Width of the window
- text String - Text displayed in the content part of the dialog window
- buttons Array - Buttons displayed at the bottom of the dialog window
- selector String - Element selector for binding to Dynamic Action (DA)
- minHeight Number - Minimum window height, default 240 v4+
- centered Boolean - Vertical alignment to the center of the page, default false v4+
- callbacks Object Accepts functions:
- onClose (Function) - Executed when the window is closed using the jsAPI.modal.close();, jsAPI.modal.accept() methods
- onAccept Function - Executed when the window is closed using the jsAPI.modal.accept() method
- onDecline Function - Executed when the window is closed using the jsAPI.modal.close() method

jsAPI.modal.close();

Closes the modal window, calls onClose, onDecline in jsAPI.modal.open

jsAPI.modal.accept();

Closes the modal window, calls onAccept, in jsAPI.modal.open

Examples

Calling the modal

```
jsAPI.modal.open(
   {
     title: "Title",
     text: "Text",
     width:  420,
     buttons: [
        {
          text: "Yes",
          action: "accept",
          class: "btn-primary"
        },
        {
          text: "No",
          action: "decline",
          class: "btn-secondary"
        }
     ]
   },
   {
     onClose: function (e) {
       // Executed at closing
     },
     onDecline: function (e) {
       // Executed when the 'No' button is pressed and at standard closing of the modal window
     },
     onAccept: function (e) {
       // Executed when the button "Yes" is pressed
     }
   }
);
```

Standard page call in a modal window

```
jsAPI.modal.open({
   page:  `/1000/`,
   title:"Title"
});
```

Opening a page in a modal with GET parameters and a selector for binding to a dynamic action

```
jsAPI.modal.open({
   page:       `/1000/?P1000_ITEM=${jsAPI.getItem("P1000_ITEM")}&clear=1000`,
   title:"Title",
   selector: "#DA_ELEMENT_ID"
});
```

Opening a page in modal with GET parameters, selector for binding to dynamic action and callbacks

```
jsAPI.modal.open(
   {
     page:     `/1000/?P1000_ITEM=${jsAPI.getItem("P1000_ITEM")}&clear=1000`,
     title: "Title",
     selector: "#DA_ELEMENT_ID"
   },
   {
     onClose: function (e) {
       console.log(e);
       // Executed at jsAPI.modal.close(), jsAPI.modal.accept()
     },
     onAccept: function (e) {
       console.log(e);
       // Executed at jsAPI.modal.accept()
     },
     onDecline: function (e) {
       console.log(e);
       // Executed at      jsAPI.modal.close()
     }
   }
 );
```

7.10.9 jsAPI: confirmModal()

Displays a window to confirm an action.

Syntax

```
jsAPI.confirmModal(options, callbacks);
```

Parameters

- options [Object] - Contains parameters:
- title [String] - Title of the window,
- text [String] - Text to be displayed in the dialog window,
- callbacks accepts functions:
- onAccept [Function] - Executed when the button "Yes" is pressed
- onDecline [Function] - Executed when the button "No" is pressed and when standard closing is performed

Note This method is a special case of the jsAPI.modal.open call:

```
jsAPI.modal.open(
   {
     title: "Title",
     text: "Text",
     width:  420,
     buttons: [ {
         text: "Yes",
         action: "accept",
         class: "btn-primary"
```

(continued on next page)

(continued from previous page)

```
      },
      {
        text: "No",
         action: "decline",
        class: "btn-secondary"
      }
    ]
  },
  {
    onClose: function (e) {
      // Executed at closing
    },
    onDecline: function (e) {
      // Executed when the "No" button is pressed, and at standard closing of the modal window
    },
    onAccept: function (e) {
      // Executed when the "Yes" button is pressed
    }
  }
);
```

Example

```
jsAPI.confirmModal(
  {
    title: "Are you sure?"
    text: "The data will not be saved if closed"
  },
  {
    onDecline: function
      // Executed when pressing the "No" button and at standard closing of the modal window
    },
    onAccept: function (e) {
      // Executed when the "Yes" button is pressed
    }
  }
);
```

7.10.10 jsAPI: refresh()

Method jsAPI.refresh is designed to refresh data in the regions.

It executes xhr query {API_URL}/callAction with the following parameters:

```
{
  action: "REFRESH",
  data: {
    data: data
  }
}
```

Syntax

```
jsAPI.refresh(data, callbacks);
```

Parameters

- data [Object] - Data to be sent in the request body
- callbacks [Object] - Accepts functions: onSuccess Function, onError Function.

Note To update a region, one must specify its id in the data parameter:

```
data: {
   {
     id: "MY_REGION_ID";
   }
 }
```

Examples

Standard call:

```
jsAPI.refresh({
   id: "PARAMS",
   items: {
     P1000_ID: "1"
   }
 });
```

Standard call with handlers:

```
jsAPI.refresh(
   {
     id: "PARAMS",
     items: {
       P1000_ID: "1"
     }
   },
   {
     onSuccess: function (data) {
       console.log(data);
     },
     onError: function (err) {
       console.error(err);
     }
   }
 );
```

7.10.11 jsAPI: changeTab()

Switches the active tab in a region with the 'Tabs' type.

Syntax

```
jsAPI.changeTab(id, tab);
```

Parameters

- id [String] - id of the region with type "Tabs"

- tab [Integer] - Tab sequence number (starting from 1)

Example

```
jsAPI.changeTab("MY_TABS", 2);
```

7.10.12 jsAPI: component()

jsAPI.component - interface for working with region data within a given context

Syntax

```
jsAPI.component(id).method();
```

Parameters

- id [String] - region id

Refresh()

methods

```
jsAPI.component(id).refresh();
```

Example

```
jsAPI.component("REPORT").refresh();
```

Implemented for the following regions:

- Calendar
- Report

Updates the region with the specified parameters (items). For example, specifying the field name for the region type { "calendar":

```
{
   "items": [ "P50_FIO".]
}
```

value of the P50_FULL_NAME field will be added to the body of the /callAction request in addition to the main parameters.
Request body:

```
{
   "action": "CALENDAR",
   "data": {
     "page": 50,
     "id": "CLNDR",
     "items": {
       "G01": "20230428030000",
       "G02": "20230429030000",
       "P50_FULL_NAME": "John Smith".
     }
   }
}
```

7.10.13 jsAPI: setItem()

Changes the value of a form item

Syntax

```
jsAPI.setItem(id, value);
```

Parameters

- id [String] - id of the form element. It is NOT a selector (specify without '#').
- value [String] - Value

Examples

Assign a new value

```
jsAPI.setItem("P1000_ITEM", "New Value");
```

Clear

```
jsAPI.setItem("P1000_ITEM", "");
```

7.10.14 jsAPI: getItem()

Returns the value of a form element

Syntax

```
jsAPI.getItem(id);
```

Parameters

- id [String] - id of the form element. It is NOT a selector (specify without '#')

Example

```
jsAPI.getItem("P1000_ITEM");
```

7.10.15 jsAPI: updateItem()

Updates the form element parameters at the page level ({API_URL}/showPage)

Syntax

```
jsAPI.updateItem(id, parameters);
```

Parameters

- id [String] - id of the form element. It is NOT a selector (specify without '#')
- parameters [Object]

Contains parameters:

- label ([String]) - Placeholder (element label)
- required [Boolean] - Mandatory filling
- col [Integer] - Width of the column in the form (from 1 to 12)
- mask ([String]) - Field mask (works only with TEXT)

- maxlength [Integer] - Max. number of characters (works only with TEXT)

Example

```
jsAPI.updateItem("P1_LOGIN",  {
   label: "New label", required:
   false,
   col: 6,
   mask: "999",
   maxlength: 3
 });
```

7.10.16 jsAPI: hideItem()

Removes a form item from the DOM and the column bound to it

Syntax

```
jsAPI.hideItem(id);
```

Parameters

- id [String] - id of the form element. It is NOT a selector (specify without '#')

Example

```
jsAPI.hideItem("P1000_ITEM");
```

7.10.17 jsAPI: showItem()

Returns the form element (item) in the DOM that was removed using jsAPI.hideItem(id) and the column bound to it

Syntax

```
jsAPI.showItem(id);
```

Parameters

- id [String] - id of the form element. It is NOT a selector (specify without '#')

Example

```
jsAPI.showItem("P1000_ITEM");
```

7.10.18 jsAPI: refreshList()

The jsAPI.refreshList method is designed to refresh lists bound to form elements such as Select List, Multiselect, Autocomplete. Executes an xhr request {API_URL}/callAction with the following parameters:

```
{
   action: "REFRESH_LIST",
   data: {
     data: data
   }
 }
```

Syntax

```
jsAPI.refreshList(data, callback);
```

Parameters

- data Object Data to be sent in the request body
- callbacks Object

The following functions are accepted: onSuccess Function, onError Function.

Note To update a form element, one must specify its id in the data parameter:

```
data: {
   {
     id: "P1000_SELECT";
   }
 }
```

Examples

Standard call

```
jsAPI.refreshList({
   id: "P1000_SELECT",
   items: {
     P1000_PARAM: "New P1000_PARAM val".
   }
 });
```

Standard call with handlers:

```
jsAPI.refreshList(
   {
     id: "P1000_SELECT",
     items: {
       P1000_PARAM: "New P1000_PARAM val".
     }
   },
   {
     onSuccess: function (data) {
       console.log(data);
     },
     onError: function (err) {
       console.error(err);
     }
   }
 );
```

7.10.19 jsAPI: dom.hide()

Hides an element with the specified css selector by adding the css class js-api-hidden

Syntax

```
jsAPI.dom.hide(selector);
```

Parameters

- selector [String] - Element selector

Examples

Example 1:

```
jsAPI.hide("#P1000_MY_ITEM");
```

7.10.20 jsAPI: dom.show()

Displays an element with the specified css selector by removing the css of the js-api-hidden class

Syntax

```
jsAPI.dom.show(selector);
```

Parameters

- selector [String] - Element selector

Examples

Example 1:

```
jsAPI.show("#P1000_MY_ITEM");
```

7.10.21 jsAPI: dom.focus()

Sets focus on the specified element if it can be focused. Wrapper for native el.focus() method

Syntax

```
jsAPI.dom.focus(selector);
```

Parameters

- selector [String] - Element selector

Examples

```
jsAPI.dom.focus("#P1000_MY_ITEM");
```

7.10.22 jsAPI: dom.blur()

Removes keyboard focus from the current element. Wrapper for native method document.activeElement.blur()

Syntax

```
jsAPI.dom.blur();
```

Parameters

This function accepts no arguments.

Examples

```
```

7.10.23 jsAPI: dom.setAttribute()

Adds and modifies attributes of an element with the specified selector

Syntax

```
jsAPI.dom.setAttribute(selector, attrs);
```

Parameters

- selector [String] - Element selector
- attrs [Object] - Attributes

Examples

```
jsAPI.setAttribute("#P1000_MY_ITEM", {
   disabled: true,
   "data-custom-attr": "My custom value".
 });
```

7.10.24 jsAPI: debug

Enables and disables different levels of debugging information display. Includes *enable* and *disable*.

Activate debug

```
jsAPI.debug.enable(type);
```

Deactivate debug

```
jsAPI.debug.disable(type);
```

Parameters

- type [String] - The name of the debugging level.

Available debugging levels

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG

Examples

Activate debugging with CRITICAL type:

```
jsAPI.debug.enable("CRITICAL");
```

Deactivate debugging with CRITICAL type:

```
jsAPI.debug.disable("CRITICAL");
```

7.10.25 jsAPI: logout()

Exits the application (logout) by calling the {API_URL}/logout method

Syntax

```
jsAPI.logout();
```

Parameters

This function accepts no arguments.

Example

```
jsAPI.logout();
```

7.10.26 jsAPI: clearLocalStorage()

Deletes all records in the application's localStorage. It is a wrapper of native method window.localStorage.clear();

Syntax

```
jsAPI.clearLocalStorage();
```

Parameters

This function accepts no arguments.

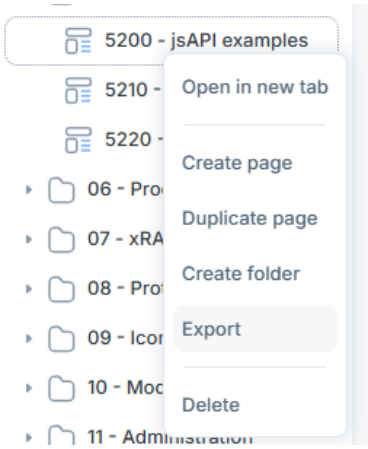
Example

```
jsAPI.clearLocalStorage();
```

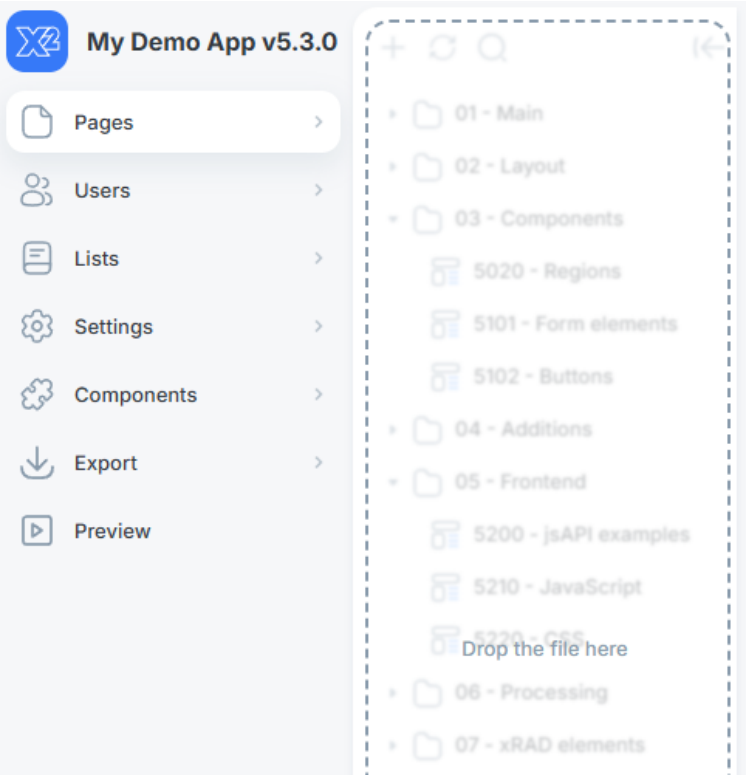
Export/import

XRAD allows to export and import a page or an entire application. This functionality is useful when you need to transfer a project to another stand or use the developments of one project in another.

Page export is available in the context menu of the page in the page list (right click - Export). The system will generate a script with .sql extension to create the page.

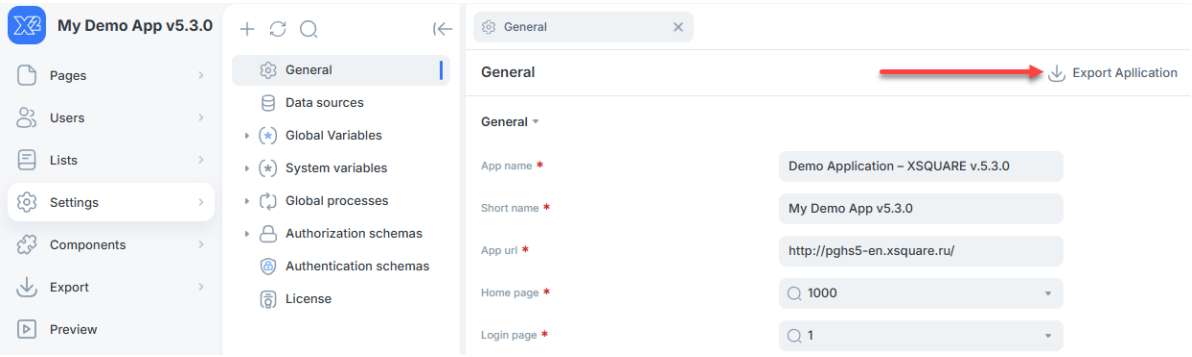


Page import is done by dragging the .sql script file into the page list area.

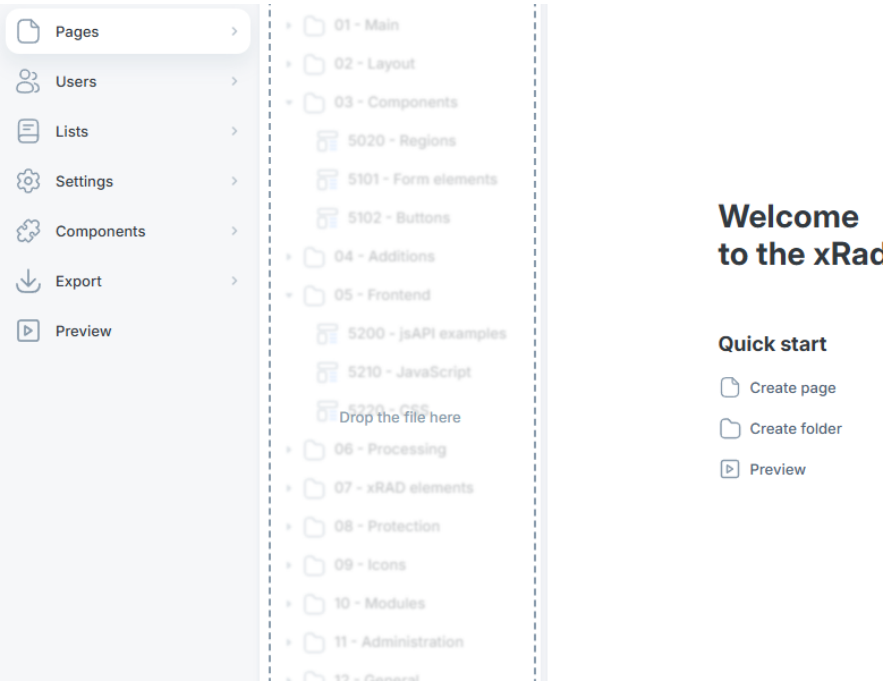


Warning. The script contains a procedure call that deletes the page before importing. The script also keeps all identifiers and links inside the components as they are in the database.

Exporting the application is available under Settings - Global. The system will generate a script with .sql extension to import the entire application.



The application is imported by dragging and dropping the .sql file into the application settings list area.



Warning. While importing, the script will completely overwrite the applications on the target stand. The script is executed in one transaction and in case of failure, the target application will remain in its original form.

7.10.27 jsAPI: setLoading()

Sets the loading state for the current page or modal window

Syntax

```
jsAPI.setLoading(value: boolean) : void
```

Parameters

value: boolean - **true** - show loading status. **false** - hide loading status.

Example

```
jsAPI.setLoading(true);
jsAPI.setLoading(false);
```

7.10.28 Loading property

Set the load status for the button/region. Supported regions:

- FORM;
- WRAPPER;
- HTML;
- TILES;
- REPORT;
- TABS;

- TREE;
- CARDS;
- DATAGRID;
- CHAT;
- CHART;
- CALENDAR.

Syntax

```
jsAPI.component("REGION_ID").properties({ loading: boolean }) : void
```

Parameters

loading: boolean - **true** - show loading status. **false** - hide loading status.

Example

```
jsAPI.component("FORM_SAMPLE").properties({ loading: true })
jsAPI.component("FORM_SAMPLE").properties({ loading: false })
```

7.10.29 jsAPI: download()

Download the file via JavaScript, without redirecting or refreshing the page.

This method allows to track the progress of the file download and to process errors that occur during generation.

One can lock the file generation button or display progress information to the user. This can be especially useful

if file generation takes a long time.

Syntax

```
jsAPI.download({
    process?: string
    url?: string
    method?: "GET" | "POST"
    data?: Record<string, any> | FormData
    headers?: Record<string, any>
    onProgress?: (progress: ProgressEvent) => void
    onSuccess?: (result: {
      response: any
    }) => void
    onError?:  (error:  { title: string; message: string, code: number }, raw: any) =>␣
↪void
}) : void
```

Parameters

process?: string - The name of the process to be called to download the file

url?: string - URL for downloading a file (file or API method)

method?: "GET| "POST" is the request method. The supported values are *GET`|`POST*.

data?: Record<string, any>| FormData - data to be sent. For POST method support of *FormData* data format is implemented

headers?: Record<string, any> - request headers

onProgress?: (progress: ProgressEvent)=> void - Callback called when the progress of a file download is updated. One can use it to display the status of the download. The callback accepts an argument of type [ProgressEvent (documentation: https://developer.mozilla.org/en-US/docs/Web/API/ProgressEvent)

```
onSuccess?: (result: {
  success: boolean
  response: any
})=> void
```

A callback called when the file download is complete. It accepts a result object that contains: - *response: any* - response from the server.

onError?: (error: { error: { title: string; message: string, code: number }, raw: any)=> void - Callback called when an error occurred while downloading a file: - *error.title* - short text; - *error.message* - *error* message; - *error.code* - *error* code.

Values:

- *-2* - failed to decode the response from the server
- *0* - Internet connection error
- *4xx,5xx* - the server returned an error.

Examples

Download file generated by *fooProcess* with data from *USERNAME* and *AGE* fields

```
jsAPI.download({
  process: `fooProcess`,
  data: {
    username: jsAPI.getItem('USERNAME'),
    age: jsAPI.getItem('AGE'),
  }
})
```

Download the /test.txt file and display progress

```
jsAPI.download({
  url: "/test.txt",
  method: "GET",
  onSuccess() {
    jsAPI.notification.success("Download completed!");
  },
  onError(error) {
    console.error(error)
    jsAPI.notification.error("Error #"+ error.code+ " <br>"+ error.message);
  },
  onProgress(progress) {
    let percent= progress.loaded / progress.total * 100;
    jsAPI.notification.warning("Downloaded "+ percent+ "%");
  }
});
```

Send data by POST method in *FormData* object to the address */api/download_file* with *token* header and lock the button for the download time:

```
let payload= new FormData();
payload.append('username', jsAPI.getItem('USERNAME'));
payload.append('age', jsAPI.getItem('AGE'));

jsAPI.component("BUTTON").properties({ loading: true });        // Block the button

jsAPI.download({
  url: "/api/download_file",
  method: "POST",
  data: payload,
  headers: { token: '@JKnfsdlk@E$@!BNKKOGDSbjO#$BKJO' },
  onSuccess() {
    jsAPI.component("BUTTON").properties({  loading: false  }); // Unlock the button
    jsAPI.notification.success("Downloading completed!");
  },
  onError(error) {
    jsAPI.component("BUTTON").properties(   loading: false }); // Unlock the button
    { console.error(error)
    jsAPI.notification.error("Error #"+ error.code+ " <br>"+ error.message);
  },
  onProgress(progress) {
      let percent= progress.loaded / progress.total * 100;
      jsAPI.notification.warning("Downloaded "+ percent+ "%");
  }
});
```