[Introduction 3](#_Toc196750424)

[Final report 3](#_Toc196750425)

[Audit scope 4](#_Toc196750426)

[Business processes 4](#_Toc196750427)

[UI forms 4](#_Toc196750428)

[Business process audit 5](#_Toc196750429)

[BP-01. X-Access-Token in Service Tasks 5](#_Toc196750430)

[BP-03. Transactions within loops 6](#_Toc196750431)

[BP-05. Search condition limits 6](#_Toc196750432)

[BP-08. Multiple data factory calls within a single transaction 8](#_Toc196750433)

[BP-09. Initialization and usage of variables 8](#_Toc196750434)

[BP-11. Modeling readable BPMN diagrams 10](#_Toc196750435)

[BP-15. Timers on User Tasks 11](#_Toc196750436)

[BP-18. Historical events for high-load business processes 11](#_Toc196750437)

[BP-19. Other 12](#_Toc196750438)

[BP-19-01. Developer business processes 12](#_Toc196750439)

[Data model audit 12](#_Toc196750440)

[DM-01. Indexes for search conditions 12](#_Toc196750441)

[DM-04. List of columns in search conditions 13](#_Toc196750442)

[DM-05. Search condition limits 14](#_Toc196750443)

[DM-06. Database schema normalization 14](#_Toc196750444)

[DM-06-1. Absence of unique constraints for fields that should contain unique values 14](#_Toc196750445)

[DM-06-2. Complete absence of foreign keys (FK) in the database 14](#_Toc196750446)

[DM-06-03. Use of JSON fields for storing structured data 15](#_Toc196750447)

[DM-09. Pagination in search conditions 16](#_Toc196750448)

[Audit of business process UI forms 17](#_Toc196750449)

[FR-01. Large forms 17](#_Toc196750450)

[FR-02. Lazy load the “Select” component 17](#_Toc196750451)

[FR-03. Javascript logic in form components 18](#_Toc196750452)

[FR-06. Use of experimental form components 19](#_Toc196750453)

[FR-07. Limits on search conditions for “Select” components 20](#_Toc196750454)

[Security audit 21](#_Toc196750455)

[SC-01. Principle of least privilege 21](#_Toc196750456)

[SC-02. Minimization of roles with elevated permissions 21](#_Toc196750457)

[SC-03. Principle of separation of duties 21](#_Toc196750458)

[SC-04. RLS (Row Level Security) in the data model 21](#_Toc196750459)

[SC-05. RBAC (Role Based Access Control) in the data model 22](#_Toc196750460)

[SC-06. Confidential data 23](#_Toc196750461)

# Introduction

During the audit of the X registry regulation, the document with general recommendations for auditing and development was used as a baseline — see [link](https://diia-engine.github.io/diia-engine-documentation/en/platform/1.9.8/registry-develop/audit/registry-audit/registry-audit-instruction.html).

Due to limitations in human and time resources, the audit scope for business processes and forms was restricted to the list available to the end user and evaluated as the most heavily used. The list was formed based on the business process authorization model. Administrative processes, developer processes, and those available only to ministry representatives were excluded from the audit scope.

The audit was conducted in two stages:

- Registry regulation version as of 11.11.2023

- Registry regulation version as of 12.12.2024

Some findings and recommendations were continuously shared with the development team during the audit process and were addressed in the updated second version of the regulation, which was also audited.

The list of identified issues is not exhaustive but represents a general pattern of usage that must be completely eliminated.

## Final report

Number of findings by category and criticality

|  |  |  |  |
| --- | --- | --- | --- |
|  | **High** | **Medium** | **Low** |
| Performance | 8 | 0 | 1 |
| Reliability | 1 | 3 | 0 |
| Security | 7 | 1 | 0 |
| Maintainability | 1 | 3 | 0 |

The following are the most critical recurring issues observed during the audit:

* Full table unloading to prefill data on forms.
* Absence of limits, filters, and pagination in search criteria.
* Use of heavy persistent variables in business processes.
* Lack of a proper procedure for preparing the registry regulation for production, including removal of all demo elements (demo roles, demo code fragments, processes under development).
* Missing database indexes.
* Lack of data access separation.
* Lack of proper rules for handling confidential data.
* Highly customized form development, which may cause problems during platform version upgrades and require additional rework.

Audit scope

#### Business processes

1. medic-log
2. team-lead-control
3. registry-doc-arrange
4. entity-summary
5. protocol-records
6. hr-supervise-all
7. group-access-queue
8. account-verify
9. client-info
10. profile-details

#### UI forms

1. adjust-schedule
2. adjust-schedule-auth
3. new-order-summary-final
4. new-order-summary-final-edit
5. new-order-summary-final-edit-auth
6. new-order-summary-final-auth
7. new-order-summary-step-i
8. new-order-summary-step-ii
9. new-order-summary-step-iii
10. new-order-summary-step-iv-opt
11. new-order-summary-step-v
12. new-order-summary-step-vi
13. new-order-summary-v2-final
14. new-order-summary-v2-final-edit
15. new-order-summary-v2-final-edit-auth
16. new-order-summary-v2-final-auth
17. new-order-summary-v2-step-ii
18. new-order-summary-v2-step-iii
19. new-order-summary-v2-step-iv
20. new-order-summary-v2-step-vi
21. new-order-summary-v2-step-vi-edit
22. new-order-summary-v2-step-vi-edit-auth
23. new-order-log
24. new-order-v2-log
25. patient-issue
26. snippet-alert

# Business process audit

## BP-01. X-Access-Token in Service Tasks

**Criticality:** Medium  
**Category:** Reliability

**Impact:**  
The user's authorization token may expire before the business process reaches the point where it is needed. This issue may not be detected during development or testing due to the absence of relevant conditions (e.g., a task left unclaimed for over 5 minutes, a long-running transaction, temporary service unavailability, etc.).

**Description:**  
When using JUEL functions to retrieve a user's authorization token, it is essential to ensure that the function call and the corresponding activity (user task or start event) occur within the same transaction of the business process.

Saving the token in a persistent (non-transient) variable is unsafe, as every persistent variable can be viewed via the business process management web interface.  
Moreover, storing the token in a persistent variable may lead to using an outdated token in service tasks (for more details, see section BP-09).

**Findings:**

* **protocol-records / Sign and save data (activitySign):** After this user task, the initAccToken variable is not updated, resulting in several subsequent service tasks at the end of the process using the token from a previous transaction.  
  **Recommendation:** Add a script that updates the initAccToken variable after the Sign and save data task (activitySign).
* **hr-supervise-all / Initialize accessToken (Activity\_0y1ra7k):** In this script task, the token used is not from the most recent task (Sign task activitySign) but from the previous task (View all reference items activityCreateTimeslots). As a result, service tasks at the end of the process use the token from a previous transaction.  
  **Recommendation:** Modify the initialization of initAccToken to use the token from completer('activitySign').accessToken.
* **group-schedule / View/edit data (activityTerminated):** After this user task, the initAccToken variable is not updated. Therefore, when the process returns to activityQuestionnairePage2, directory queries related to the task View/edit data (activityQuestionnairePage2) may be executed using an outdated token.  
  **Recommendation:** Add a script that updates the initAccToken variable after the View/edit data task (activityTerminated).
* **User registration in the system (user-confirm-registration) / Welcome information (activityWelcome):** After this user task, the initAccToken variable is not updated. As a result, the next service task, Fetch company list from DB (Activity\_1pvg0nd), uses an outdated user token.  
  **Recommendation:** Add a script that updates the initAccToken variable after the Welcome information task (activityWelcome).

## BP-03. Transactions within loops

**Criticality:** Medium  
**Category:** Reliability

**Impact:**  
When loops with many iterations are present, the Camunda Engine holds a database transaction open throughout the entire execution time. This can lead to the negative effects associated with long-running transactions. Additionally, if an error occurs during, for example, the 100th iteration, the engine will attempt to re-execute all iterations starting from the first one.

**Description:**  
A long transaction can be caused by a loop that processes many iterations. Therefore, when modeling loops in business processes, it is necessary to apply asynchronous execution for each iteration to prevent long transaction holding.

**Findings:**

* **group-access-queue / Assign supervisor (Group\_00mue9k):** The update-teams-boss resource is updated inside a loop, which may cause a long-running transaction.

**Recommendations:**

* Consider applying asynchronous execution to the loop:  
  Set the camunda:asyncBefore attribute on the task *Prepare save cycle* (Activity\_12kr5gc) and use the system\_user().accessToken in all service tasks between this script task and the next user task. Alternatively, consider using an Expanded Sub-Process with Multi-Instance configuration for updating the update-teams-boss resource.

## BP-05. Search condition limits

**Criticality:** High  
**Category:** Performance

**Description:**  
When using a service task to search for entities in the Data Factory or calling the Keycloak API, you must explicitly set a maximum limit (limit) for the number of records that can be retrieved.  
Failing to set a limit causes several issues:

* It increases the load on the database (both Keycloak and registry databases). Connections are held open longer, queries queue up, and overall response times increase across services.
* It raises heap memory usage across multiple services (keycloak, bpms, registry-rest-api). Long connection holding ties up threads in the thread pool, impacting response time across the platform.

**Examples:**

* **group-access-queue / Fetch directory data from the database (Activity\_05irp75):**  
  Calls the get-organizations resource without any search criteria, limits, or pagination.
* **profile-details / Fetch all users from Keycloak (Activity\_198yrv5):**  
  Loads all users from Keycloak into the business process for further processing.
* **client-info (initial version) / Fetch directory data from the database (Activity\_07isovq):**  
  Filters only by two boolean parameters (teamActive and isAssigned), which potentially retrieves ¼ of all records from the table.
* **medic-log (initial version) / Fetch diagnosis data (Activity\_1y9fo2j):**  
  Filters by a single boolean parameter (directoryMkxActive), which could retrieve up to 50% of table records into the business process.
* **entity-summary / Fetch Organizations from the database (Activity\_1owk9w8):**  
  Fetches all records from the get-organizations resource without applying search criteria, limits, or pagination.

**Recommendations:**

* Avoid preloading all table data into the memory of a business process just to populate a form. Instead, use select fields that pull data from the Data Factory using filtering criteria and limits.
* If it is necessary to process all users within a business process:
  + Implement batch retrieval with pagination and phased processing.
  + Reevaluate whether it is necessary to retrieve all table records; apply filtering based on specific criteria whenever possible.

## BP-08. Multiple data factory calls within a single transaction

**Criticality:** Medium  
**Category:** Reliability

**Impact:**  
The risk of creating inconsistent data in the database if an error occurs between separate Data Factory calls. Depending on the business logic and the registry data model, this can result in blocking the ability to work with specific records. If an error occurs, the retry policy of the business process will restart execution from the beginning, which may cause repeated inserts into certain database tables.

**Description:**  
When modeling a business process, there may be a need to update multiple database tables as part of a single database transaction (not to be confused with a business process transaction). The platform does not provide a way to group multiple Data Factory calls into a single transaction. Therefore, several sequential Data Factory calls in a business process can lead to inconsistent data if an error occurs midway.

**Where Found:**

* **account-verify / View/Edit Data (activityBlankFinalSign):**

After this user task, multiple non-idempotent Data Factory save operations are performed.

**Recommendations:**

* Since the use of nested entities is not feasible due to the presence of update operations within the business process transaction, it is recommended to design compensatory actions for each database operation. Model database update operations as closely and atomically as possible to reduce the risk of inconsistency.

## BP-09. Initialization and usage of variables

**Criticality:** High  
**Category:** Performance, security

**Impact:**

* Storing a user's token in a persistent (non-transient) variable can create security risks because all persistent variables are saved in the Camunda database and can be viewed or even modified through the Web-based process execution management interface.
* A large number of massive persistent variables increases the load on the Camunda database, which in turn extends the transaction time of the business process.

**Description:**  
When it is necessary to create additional variables in a business process, they should be initialized as close as possible to the point of their actual use. Additionally, both the number and size of persistent variables should be minimized.

**Example: Storing a token in a persistent variable:**

* **hr-supervise-all** – User token is saved as a persistent (non-transient) variable:
  + Initialization of accessToken, bpTimeOut (Activity\_0q7yhor)
  + Token update (Activity\_1ddy2pg)
  + Initialization of accessToken (Activity\_0ektqic)
  + Initialization of accessToken (Activity\_1ssn51o)
  + Initialization of accessToken (Activity\_1494y0d)
  + Initialization of accessToken (Activity\_1esrxmp)
  + Initialization of accessToken (Activity\_029b6tq)
  + Initialization of accessToken (Activity\_0jayguo)
  + Initialization of accessToken (Activity\_1cppcvk)
  + Initialization of accessToken (Activity\_1dfgyuq)
  + Initialization of accessToken (Activity\_0ura22x)
  + Initialization of accessToken (Activity\_0vxzu5t)
  + Initialization of accessToken (Activity\_0dv53k2)
  + Initialization of accessToken (Activity\_1jhg5zu)
* **team-lead-control** – User token is saved as a persistent (non-transient) variable:
  + Initialization of accessToken and process timeOut (Activity\_0q7yhor)
  + Initialization of accessToken (Activity\_0ektqic)
  + Initialization of accessToken (Activity\_1ssn51o)
  + Initialization of accessToken (Activity\_1k45sut)

**Recommendations:**

* Always use transient variables to store user tokens.

**Example: Early initialization of a variable:**

* **client-info / Preparing data for display (Activity\_1biwds4):**

The script is named "Preparing data for display," but the data prepared there is not used on the next form. It is only used later, after the "Case Journal" (activityDictMain), even though it is initialized before reaching it.

**Recommendations:**

* Always initialize variables immediately before their use. Initialization can be either an explicit creation of the variable or the use of tasks whose output results are automatically saved as variables.

## BP-11. Modeling readable BPMN diagrams

**Criticality:** Low  
**Category:** Maintainability

**BPMN modeling practices**

**Description:**  
When modeling BPMN diagrams, it is important to follow commonly accepted practices. For detailed guidance, please refer to the [link](https://diia-engine.github.io/diia-engine-documentation/en/platform/1.9.8/registry-develop/audit/registry-audit/modules/bp-audit.html#_bp_11).

**Where found:**

* **account-verify:** The bpmn:collaboration tag contains an invalid attribute camunda:delegateExpression="${keycloakGetUserRoleConnectorDelegate}", which was likely mistakenly copied when duplicating a part of the business process.

**Recommendations:**

* Remove any unnecessary attributes from the business process description tags (the XML code of the business process).

***Creating readable sequence flows***

**Where found:**

* **account-verify:** The gateway after init (Activity\_0ganl0c) has no name, making it unclear what user action should lead to the lower branch.  
  Additionally, two gateways are named “тест page 2?” but they have different conditions, which is confusing.
* **hr-supervise-all:** The gateway after init (Activity\_0ganl0c) has no name.
* **hr-supervise-all:** The script task Initialize accessToken (Activity\_08x5y9z) has a misleading name, as the operation performed inside is unrelated to tokens.
* **account-verify:** There are unused elements or chains of elements in the business process.

**Recommendations:**

* Assign clear and meaningful names to all elements in the business process diagram, ensuring that the names reflect the specific action or decision being performed.
* Remove unused elements that are not actively part of the business process. If they are needed for future use or testing purposes, add comments explaining their purpose.

## BP-15. Timers on User Tasks

**Criticality:** High  
**Category:** Performance

**Impact:**

Accumulation of open business processes due to user tasks that are no longer intended to be completed leads to unnecessary load on the system, excessive resource consumption, and the need for additional operations to remove abandoned business processes.

**Description:**

In registry-related business processes, responsible users work with user tasks assigned to them. However, for various reasons, these tasks may be forgotten and never completed.

One of the possible solutions is to implement timers that automatically terminate the business process after a certain period of inactivity.

**Where found:**

* account-verify / Notify Subject (activityMessage)
* account-verify / Notify Subject (Activity\_0owgxfe)
* account-verify / Notify Subject (Activity\_1dj9zzs)
* account-verify / Notify Subject (Activity\_0p6bmdi)
* registry-doc-arrange / Info on Completion (Activity\_17zat1v)

## BP-18. Historical events for high-load business processes

**Criticality:** Low  
**Category:** Performance

**Impact:**  
Additional load on the Asynchronous Messaging Subsystem and the Process History Persistence Service due to the large number of historical events generated during business process execution. This results in increased processing delays and potential failures of certain system components. There is also increased load on the Relational Database Management Subsystem, which may cause critical services, such as the Business Process Execution Service, to fail — potentially blocking the entire registry operation.

**Description:**  
During the execution of business processes, historical events are recorded, such as task completions and the storage of variable updates. The process of saving these events can significantly strain the system and even lead to failures if the volume of running processes is large. It is recommended to adapt business process designs accordingly when a high load is expected.

**Where found:**

* protocol-records
* hr-supervise-all
* group-access-queue
* account-verify

**Recommendations:**

* Avoid using full initiator names as business keys in processes that do not involve interaction between multiple users.
* Minimize the number of tasks executed within a business process. For example, if two script tasks are executed consecutively, consider combining them into one.
* Consider replacing script tasks with Execution Listeners where appropriate.
* Minimize the use of persistent variables (e.g., initAccToken, userOrg).

## BP-19. Other

### BP-19-01. Developer business processes

**Criticality:** High  
**Category:** Security

Developer business processes (such as developer-tools and developer-restore-users) must be excluded from the production environment to avoid security risks and unauthorized actions.

# Data model audit

## DM-01. Indexes for search conditions

**Criticality:** High  
**Category:** Performance

**Impact:**  
The absence of required indexes in the database can lead to:

* Significant degradation of query performance, especially on large datasets.
* Increased system response time.
* Excessive load on the database server.
* Inefficient use of system resources.

**Recommendations:**

* Create indexes on all foreign keys (FK).
* Create indexes on other fields (in addition to unique and foreign keys) that are used in table JOINs, if any.
* Use the indexing=true option for search criteria to automatically create indexes according to the search type.

If a decision is made not to use automatic index creation via the indexing=true option, use the following templates for manual index creation depending on the search type:

* **For text and varchar fields with searchType="equal":**

CREATE INDEX IF NOT EXISTS user\_drfo ON users(lower(cast(user\_drfo as varchar)));

* **For searchType="contains":**

CREATE INDEX ix\_user\_work\_data ON users USING gin (user\_work\_data gin\_trgm\_ops);

* **For searchType="startsWith" on text fields:**

CREATE INDEX tbl\_col\_idx ON tbl(col text\_pattern\_ops);

* **For searchType="startsWith" on varchar fields:**

CREATE INDEX tbl\_col\_idx ON tbl(col varchar\_pattern\_ops);

* **For searchType="startsWith" on char fields:**

CREATE INDEX tbl\_col\_idx ON tbl(col bpchar\_pattern\_ops);

In all other cases, create regular indexes.

## DM-04. List of columns in search conditions

**Criticality:** High  
**Category:** Performance, Security

**Key findings:**

* For most fields returned in Search Criteria, a search type is assigned.
* However, real business logic does not require search functionality for the majority of these fields.

**Impact:**

* Potential creation of unnecessary indexes if indexing=true is enabled.
* Alternatively, the need to disable automatic indexing and manage indexes manually.

**Recommendations:**

* Conduct a detailed analysis of all Search Criteria and the corresponding business logic.
* Remove the searchType attribute for fields that are not used in actual search operations.
* Alternatively, disable the automatic creation of indexes as mentioned in DM-01 and create only necessary indexes manually, taking into account the field type and the search type.

## DM-05. Search condition limits

**Criticality:** High  
**Category:** Performance, Security

Described in section DM-09.

## DM-06. Database schema normalization

**Criticality:** High  
**Category:** Reliability

### DM-06-1. Absence of unique constraints for fields that should contain unique values

**Impact:**

* Data integrity issues
* Decreased query performance

**Examples:**

* team\_case\_statuses.team\_case\_statuses\_numb
* account.teams\_code
* Possibly other natural keys in different directories

**Recommendations:**

* Identify fields that require unique constraints.
* Create the appropriate uniqueness constraints in the database.
* Verify existing data for duplicates.

### DM-06-2. Complete absence of foreign keys (FK) in the database

**Impact:**

* Violation of data integrity between tables
* Risk of "orphaned" records
* Difficulty in maintaining relationships between tables
* Increased complexity in analyzing and understanding the database structure

**Examples:**

* doc.case\_status
* time.timeslots\_teams\_code

**Recommendations:**

* Analyze relationships between tables.
* Identify all necessary foreign keys.
* Create appropriate FK constraints in the database.
* Validate existing data against the new constraints.

### DM-06-03. Use of JSON fields for storing structured data

**Impact:**

* Difficulty in querying and indexing data stored in JSON fields
* Potential performance issues when working with large volumes of data
* Complicated data validation at the database level
* Possible data integrity issues
* Increased complexity when analyzing and reporting based on these data

**Examples:**

* In the users table, the user\_work\_data field stores additional information (such as edrpou, job, spec, roles) in JSON format.
* In the time table, the timeslots\_data field stores a complex data structure for timeslots in JSON format.

**Recommendations:**

* Analyze the data structures currently stored in JSON fields.
* Consider normalizing the data:
  + Create a separate table with dedicated fields for user\_work\_data.
  + Create a separate table for timeslots\_data to store timeslots with appropriate relationships.
* If full normalization is not feasible:
  + Create proper indexes (see DM-01 for searchType="contains" cases).
  + Implement data validation mechanisms at the application level.

While using JSON fields can be justified in certain cases, it is recommended to review the current data structure to improve performance, ensure data integrity, and simplify future work with the data.

## DM-09. Pagination in search conditions

**Criticality:** High  
**Category:** Performance

**Impact:**

* High load on the database server
* Increased query execution time
* Network overload
* A large amount of returned data can overwhelm the memory of the REST API application
* As the number of users or data grows, the load will increase non-linearly, potentially leading to system failures or significant performance degradation

**Findings:**  
Some queries return on average more than 100 rows, and some queries return up to 11,000 rows.

**Examples:**

* get\_countries\_v
* get\_directory\_account\_spec\_v where account\_spec\_active = $1
* get\_org\_v
* get\_org\_v where active = $1
* get\_org\_v where is\_assigned = $1
* get\_org\_v where (is\_assigned = $1 and active = $2)
* get\_dir\_mk\_v where active = $1
* get\_users\_v where user\_work\_data ilike ($2)
* get\_users\_v where user\_roles ilike ($2)
* get\_dir\_dr\_v where active = $1
* get\_dir\_dr\_dt\_v where active = $1
* get\_org\_choice\_v

**Recommendations:**

* Review the business logic to verify if retrieving all data is truly necessary. Where possible, add additional filters to reduce the number of returned rows or implement pagination.
* Apply a forced limit at the data model level for search criteria.

## Audit of business process UI forms

## FR-01. Large forms

**Criticality:** High  
**Category:** Maintainability

**Impact:**

* Degraded user experience when filling out or viewing the form.
* Increased likelihood of system errors for the user while working with the form (e.g., loss of entered data due to session expiration).
* Complicated form testing due to a large number of test cases that need to be covered.

**Examples:**

|  |  |
| --- | --- |
| UI form name | Number of components |
| new-order-summary-step-vi | 240 |
| snippet-alert | 236 |
| new-order-summary-v2-final-edit | 219 |
| questionnaire-view | 134 |

**Recommendations:**

* For forms requiring the input of a large amount of data, it is advisable to split the form into groups based on the entities being filled out.

For example, for the **details-page-final** form, personal information, contact information (including authorized representatives), and healthcare institution data could be separated into individual forms.

* For the **team-control-main** form, even though the number of components is not large, duplicate fields such as “Current Date” and “New Date Selection” could be merged by pre-populating the editing form.
* An exception for large forms may apply to signing tasks, where the user must view and sign information in one step.

## FR-02. Lazy load the “Select” component

**Criticality:** High  
**Category:** Performance

**Impact:**

* Additional load on servers and network
* Increased form rendering time
* In some cases, with a significant number of Select components, users may encounter a system error related to authorization token refresh due to parallel execution of multiple requests

**Examples:**

* **team-control-main** – 11 non-lazy Select components, resulting in at least 11 simultaneous requests when the form page is opened
* **new-order-summary-v2-final-auth** – 6 non-lazy Select components

**Recommendations:**

In general, select components with lazy loading are actively used across the registry. However, in the forms mentioned above, lazy loading is not applied.

This can be critical for registries experiencing high load and for frequently used forms.

As a self-check practice, it is advisable to monitor the Network tab in the browser’s DevTools for a large number of requests upon opening the form.

Additionally, smaller and more focused forms make it easier to identify and troubleshoot such cases.

## FR-03. Javascript logic in form components

**Criticality:** Medium  
**Category:** Maintainability

**Impact:**

* This code is not covered by static analysis tools, vulnerability scanners, or unit tests. Whenever possible, it is recommended to avoid its usage.
* A large amount of JavaScript logic embedded in form components often leads to errors. The code in these fields undergoes only basic syntax checking, making runtime errors difficult to detect and troubleshoot.
* This code is executed on behalf of the user initiating the business process, potentially leading to security vulnerabilities.
* Executing code unrelated to the context of the field where it is invoked (e.g., navigation, styling in customValidation or other fields) may cause critical errors in the corresponding portal, especially since platform developers may change HTML elements or page routes.

**Examples:**

* In almost all forms, additional navigation buttons are added in the header with the following JavaScript logic:

var pathSegments = urlObject.pathname.split('/');

var newPathSegments = pathSegments.slice(0, -2);

newPathSegments.push('process-list');

var newURL = urlObject.origin + newPathSegments.join('/');

window.location.href = newURL;

The "process-list" URL may be changed in future platform versions or hotfix releases. Additionally, this type of navigation can break the browser's native "back" and "forward" functionality.  
It is recommended to reconsider using additional custom navigation on forms or replace it with a built-in **Button** with the **Navigation** action.

* In almost all forms, inside the columns component (used to display additional form headers), there is excessive JavaScript logic in the **Advanced Conditions** field, including navigation, styling, and conditional text rendering.  
  This field should be used only for simple conditional rendering of the columns component. The custom logic there is fragile and may break after platform updates.
* In Email components, a custom validation script like if (input.includes('.ru'))... is used (e.g., in the form new-order-summary-v2-step-iv).  
  The platform already supports a built-in flexible **email blacklist** configuration via the portal’s environment-js ConfigMap in the emailBlacklist property.
* In the adjust-schedule-auth form, a "Print" button is implemented with excessive custom JavaScript logic to print the form.  
  Printing should be handled via **extracts** (registry excerpts) instead. Direct DOM manipulation is discouraged, especially in React and low-code environments.

**Recommendations:**

* Whenever possible, avoid custom navigation and rely on the built-in portal navigation and **Navigation Button** component.
* Replace the "Print" button with an extract-based solution.
* Migrate email validation to the built-in emailBlacklist functionality.
* Avoid custom styling through JavaScript logic where possible.

## FR-06. Use of experimental form components

**Criticality:** Medium to High (depending on the case)  
**Category:** Maintainability

**Impact:**

* In some cases, the criticality can be high because certain experimental components that encapsulate other components (for example, well) may break the logic of the components they contain.
* Complex components such as EditGrid, File, or DataImport may behave incorrectly when placed inside experimental wrapper components that are not part of the "Updated" component set. This can disrupt pre-population, scripting, or component-specific logic.
* Experimental components have not been fully tested by platform developers, so their compatibility with other components is not guaranteed and requires additional validation effort from form and registry developers.
* Their visual rendering may not comply with the platform’s design system and might not adapt correctly when UI themes are switched.

**Examples:**

* The well component is used in almost every analyzed form.
* Across all reviewed forms, the following experimental components were identified: well, datagrid, content, fieldset.

**Recommendations:**

* The fieldset component has an equivalent among the "Updated" components. It is recommended to replace it accordingly.
* The content component also has an alternative among the updated and supported components, with slightly reduced functionality but better visual and functional stability. Consider migrating to the updated component.
* For the datagrid component, it is recommended to consider replacing it with EditGrid or Columns, which are better integrated into the platform and better maintained.

## FR-07. Limits on search conditions for “Select” components

**Criticality:** High  
**Category:** Performance

**Impact:**

* Additional unnecessary load on the relational database.
* Additional unnecessary load on the synchronous registry data management service.
* Increased form rendering time.

**Examples:**

* snippet-alert – addCommunity, addrSettlement, formAddrCommunity, formAddrSettlement.
* new-order-summary-v2-final-edit – addCommunity, addrSettlement, addrCommunity, formAddrSettlement.
* new-order-summary-v2-step-iii – addCommunity, caddrSettlement, formAddrCommunity, formAddrSettlement.

**Recommendations:**

* Always set limits. Even when the entire list of values is needed, it's better to use a reasonable limit value.
* This will improve the performance of the Select component and significantly reduce unnecessary load on the database.

# Security audit

## SC-01. Principle of least privilege

**Criticality:** High  
**Category:** Security

Detailed description available in sections SC-04, SC-05.

## SC-02. Minimization of roles with elevated permissions

**Criticality:** Medium  
**Category:** Security

**Impact:**  
The demo role is typically used for demonstration or testing purposes and may lack proper security configurations. This creates a risk of unauthorized access to critical business processes.

**Recommendations:**

The demo role, which currently has access to all business processes, must be removed from the production environment.

## SC-03. Principle of separation of duties

**Criticality:** High  
**Category:** Security

Detailed description available in sections SC-04, SC-05.

## SC-04. RLS (Row Level Security) in the data model

**Criticality:** High  
**Category:** Security

**Impact:**  
After authentication, a system user can call a search criterion and retrieve information (table rows) that was not intended to be accessible.

**Description:**  
Row-Level Security (RLS) allows restricting read access to specific rows (entities) based on user attributes.  
All search criteria should be considered publicly accessible by default, and appropriate filtering rules must be added at the data model level.

For example, in the get-users search criterion, which uses DRFO (taxpayer ID) as a filtering parameter, any authenticated user can either omit the parameter or substitute someone else's DRFO to gain unauthorized access to sensitive information.  
To mitigate this, forced filtering based on user token data must be applied.

**Example of usage:**

registry-doc-arrange / Retrieve subject organization data from the DB: uses the get-users search criterion with the DRFO of the process initiator.

**Recommendations:**

Add RLS rules for automatic filtering of search results based on the authenticated user's token data.

## SC-05. RBAC (Role Based Access Control) in the data model

**Criticality:** High  
**Category:** Security

**Impact:**  
After authentication, a system user can call a search criterion and retrieve information (search criteria or table) that was not intended to be accessible.

**Description:**  
To ensure data security and granular access control, it is necessary to use the RBAC (Role-Based Access Control) mechanism. RBAC allows restricting access to an entity or specific attributes (table columns) based on the user's role.

**Where found:**

No RBAC access control rules were configured in the reviewed registry regulation.

**Recommendations:**

* Create an access control matrix for all roles in the registry, specifying access rights to tables and their columns.
* Implement the corresponding RBAC rules at the data model level based on this matrix.

*Table example*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **katottg** | **users** | **goods** | **accounts** | **system\_pool** |
| **user** | read | deny | read | read | deny |
| **leader** | read | read | read | read | read |
| **manager** | read | read | create | create | read |
| **secretary** | read | read | read | read | read |

## SC-06. Confidential data

**Criticality:** High  
**Category:** Security

**Impact:**

* **Risk of data leaks:** The probability of a data leak increases if the database is compromised.
* **Increased attack surface:** A higher volume of confidential data stored in the database makes it a more valuable target for attackers.

**Where found:**

* Storage of the user access token (described in detail in section BP-07: Handling transient variables)
* registry-doc-arrange / invocation of the search criterion get-users: the query response contains confidential information (such as name, phone number, email address) which is logged in bpms and is not further used in the business process.

**Recommendations:**

* Identify what information in the registry is considered confidential (e.g., DRFO (tax ID), name, phone number, address, passport series and number), including temporary data used in business processes.
* Reassess the necessity of storing such information at all.
* Remove any persistent storage of confidential information within business process variables, both in plain text and in the form of user tokens.
* Minimize the amount of confidential information stored in user attributes in Keycloak.