Terms of Reference

Reusable Component: Application & Case Management Framework

Mendix Reusable Components Initiative - ISAA

Version	Status
0.0.1	Draft

1. Background

This Terms of Reference document is issued as part of ISAA's (Information Systems Agency of Armenia) initiative to build a standard, reusable foundation of Mendix components for public digital services. Full context, goals, and target users for the overall initiative are described in the accompanying **Concept Note**.

Almost every public digital service follows the same shape: something is applied for, an authority examines it, and a decision is issued and recorded. Licences, permits, registrations, notifications and certifications differ in their rules, their evidence and who decides - but not in that shape. Today each project re-implements it independently - its own status list, its own transition rules, its own work queue, its own reassignment logic and its own hard-coded decision buttons - so a change of process requires a change of code, and two services in the same ministry behave differently for no reason a citizen could explain. This component addresses that gap by concentrating the shared shape into one configurable framework, so that defining a new service's process becomes an administrative act rather than a development project.

2. Objective

Develop a configurable, reusable **Application & Case Management Framework** for Mendix applications that standardizes how case processes are modelled, executed, allocated and controlled across public digital services, with the process itself defined as configuration rather than code.

3. Scope of Work

3.1 Functional Requirements

Standardized Case Model

Provide a common representation of a case - its subject, its current state, its history and its outstanding work - across different application domain models and service types.

Reusable Case Management Capabilities

The framework should provide reusable capabilities for:

- Process Flow definition and versioning;
- State and transition control;
- Work allocation, queues and reassignment;
- Task checklists and decision gating;
- Deadlines, reminders and escalation;
- Applicant interaction, withdrawal, cancellation and reopening;
- Case search and operational reporting.

Configuration-Based Process Flow Definition

An administrator should be able to define a service's process flow as an ordered set of steps - each with its status, its responsible actor and its outcomes.

Declarative State Model

Which state may follow which, and who may make that move, should be held as configuration and enforced in a single place, so that no code path can move a case in a way the configuration does not permit. A configured process should be validated before it is put into use - unreachable states, states with no way out and steps with no responsible actor should be detected before a citizen meets them, not after.

Process Flow Versioning

Processes change while cases are in flight. A case must be judged by the process flow version in force when it started, and publishing a new version must not retroactively alter cases already under way.

Work Allocation & Queues

Provide reusable work queue capabilities: presenting outstanding work to the right officers, claiming and releasing a case, and reassigning work between officers or teams. Allocation should be configurable - to an individual, to a role or to an organisational unit - without application-specific logic.

Task Checklists & Decision Gating

An administrator should be able to define, per step, the verification questions an officer must

answer, and to require that mandatory answers are recorded before any decision outcome becomes available. The questions asked of a case should be fixed at the moment the case reaches the step, so that later edits to the checklist do not change what a past decision was based on.

Automatic and System Steps

Not every step involves a person. The framework should support steps performed by the system - including services that are registered on submission with no human review at all - so that a notification-type service and a licence-type service can be expressed in the same model without either being a special case in code.

Deadlines, Reminders & Escalation

Support configurable processing deadlines per step or per process, with reminders and escalation when a deadline approaches or passes, and make elapsed and remaining time visible to both the officer and the applicant.

Applicant-Facing Interaction

Where a process requires something further from the applicant - additional documents, a correction, a clarification - the framework should support returning the case to the applicant and receiving it back into the same case, without the applicant starting again.

Withdrawal, Cancellation & Reopening

Provide standard handling for cases that end without a decision, and for reopening a closed case where the law permits it, including how the reopened case relates to the original record.

Authorization/Access

Provide a reusable authorization model that integrates with the application's security model, controlling which cases an officer may see, which steps they may act on and which outcomes they may choose, without requiring application-specific authorization logic.

Reporting & Search

Support searching and filtering cases by state, service type, assigned officer, date range and applicant, and provide operational reporting on volumes, outcomes and processing times.

Integration with Other Reusable Components

The framework should be usable by, and integrate with, other components in this initiative - in particular the **Audit & Activity History Framework** (every state change and decision appearing in the case history), the **Document Upload & Attachment Framework** (documents belonging to a case, and completeness as a condition of submission), the **API Integration Framework** (where a step reads from or writes to an external registry through X-Road or a direct API), an application/form component (the answers a case is decided on being reachable from the case), a notification component, and the register that records the resulting authorisation. Integration

points and mechanisms should be reviewed and defined together with each relevant component's vendor.

Configuration-Based Adoption

New service processes should be enabled through configuration where possible, minimizing custom development effort. The framework must be adoptable by an application without that application's domain model and this component becoming structurally dependent on one another.

Vendors should describe:

- How a new service process is added (configuration steps vs. required development), how a case refers to its business subject, and what an adopting application must and must not change in its own model;
- How allowed transitions are declared, validated and enforced in a single place, and whether an administrator can inspect the resulting state model visually and simulate a case through it;
- How process flow versions are published, superseded and pinned to a case;
- How allocation rules, checklists, deadlines and escalation are configured per step without custom code;
- How the framework relates to Mendix's native Workflow capabilities (built on, extended or replaced, with justification);
- How operational visibility (queue sizes, ageing cases, processing times) is exposed to administrators.

3.2 Non-Functional Requirements

- **Security**: Transition and access rules enforced consistently regardless of access path (UI, API, scheduled process); every decision and reassignment attributable to an identified user or system step.
- **Performance**: Transition evaluation and queue rendering should not materially degrade transactional performance; heavy operations (e.g., bulk reassignment, reporting) should run asynchronously.
- **Scalability**: Queues and case history should remain performant (e.g., via pagination and indexing) for high-volume services and long-running cases.
- **Observability**: Structured logging of state changes, decisions and allocation events sufficient to support the Audit & Activity History Framework and operational troubleshooting.
- **Compatibility**: Must run on *Mendix version > 11.10.0* and be upgradable across supported Mendix versions.

- **Localization**: All administrator-configured labels and applicant-facing text maintainable in Armenian and English.

4. Deliverables

Vendors are expected to provide the following, in addition to the specific requirements above:

Standard Implementation Deliverables

#	Deliverable	Description
1	Mendix Source	Mendix application source (Team Server/Git) and deployment package.
2	Technical Documentation	Module purpose, architecture, dependencies, exposed entities, microflows, Java actions, and configuration.
3	Architecture Documentation	Architecture overview and key design decisions.
4	Security Configuration	Module roles, user role mappings, authentication and authorization configuration.
5	API Documentation	Interface specifications, request/response schemas, sample payloads, and error handling.
6	Configuration Guide	Environment-specific constants and configuration settings.
7	Third-Party Component Inventory	Marketplace modules and external dependencies used.
8	Code Quality Report	Static code analysis and Mendix quality metrics.
9	Test Plan	Unit, integration, system, UAT, regression, and non-functional testing approach.
10	Test Results	Test cases and execution evidence.
11	Security Test Report	Security testing and vulnerability assessment results (where applicable).
12	Performance Test Report	Performance and load testing results (where applicable).
13	User Documentation	User and administrator guides.
14	Training Materials	Training guides and supporting materials (where applicable).

Additional Deliverables for Reusable Components

#	Deliverable	Description

1	Module README	Purpose, prerequisites, installation and upgrade instructions.
2	Interface Contract	Public microflows, entities, Java actions, parameters, and expected outputs.
3	Configuration Guide	Post-import configuration instructions.
4	Sample Application	Reference application demonstrating component implementation, including at least two contrasting processes configured without code (one requiring human review with a checklist, one registered automatically on submission) and a worked example of publishing a new process flow version while a case is in flight.

5. Vendor Response Requirements

Vendor proposals should address:

1. **Technical approach** - proposed architecture, how it maps to the functional requirements in Section 3, and how it relates to Mendix's native Workflow capabilities.
2. **Configuration model** - concretely how an administrator defines a new service process, and what is/isn't possible without custom development.
3. **State model & versioning** - how allowed transitions are declared, validated and enforced, and how in-flight cases are protected from process changes.
4. **Security approach** - authorization model, allocation rules, and controls over who may decide what.
5. **Reuse & extensibility** - how the framework avoids tight coupling to any single adopting application, and how it will be consumed by other components in this initiative.
6. **Team & experience** - Experience with Mendix (certificates, if available) and completed relevant projects, in particular case management or workflow-oriented systems.
7. **Timeline** - proposed delivery schedule against the deliverables in Section 4.
8. **Indicative Cost** - cost breakdown by phase/deliverable.
9. **Assumptions & risks** - any assumptions made and risks identified.