**Clinical Decision Support (CDS)**

Technical proposal document

**Thought**Works

**Table Of Contents**

[**1. Introduction**](#_k6ylb9o7bvw8) **3**

[**2. Scope**](#_f9ritruqrhff) **3**

[**3. Types of notifications**](#_w6djscr9tl58) **3**

[**4. Rules Configuration Approaches**](#_qpl1vdk8nqmo) **7**

[Rule Definition](#_14j3mcc0kk05) 7

[UI based configuration](#_dkzvykwy8k2u) 7

[Advantages](#_jcvhb4k0rcdz) 10

[Disadvantages](#_2iagyzs20uol) 10

[Configuration based](#_voy5zzkiafdi) 11

[Advantages](#_koj7vxwbuw4x) 12

[Disadvantages](#_pvxob5o831kc) 12

[**5. Rule Store**](#_hbbixwijmbv5) **12**

[**6. Alert Store**](#_dr9dzpvmy1e3) **13**

[**7. The CDS engine**](#_a4xk3inavobm) **14**

[Architecture Diagram](#_9odtzewymul0) 14

[Approach 1:](#_wqadq77mwmid) 15

[Front-End layer](#_ewmej2t0n6ll) 15

[Back-End module](#_urgfsyt5bsww) 15

[Approach 2:](#_nxvtqawytii3) 16

[Front-End layer](#_bdrldt9z0b8s) 16

[Back-End module](#_td5v7vuentp0) 16

[**8. Enhancements**](#_jxa0si4958jo) **17**

## 1. Introduction

In Bahmni there are a lot of places where data validation is required. And these validations usually result in some form of notification (like an error pop-up message).

But the need and execution of these validations are spread across the system with various configurations and implementations.

Clinical Decision Support (CDS) is aimed towards centralizing this in Bahmni(OpenMRS). It is a rule-based engine which would assess the data flowing from the Bahmni-Web, Bahmni-Connect, OpenELIS, OpenERP etc.

## 2. Scope

This document is written with the following versions of the products under Bahmni ecosystem:

* Bahmni: 0.91
* Forms: 1.0

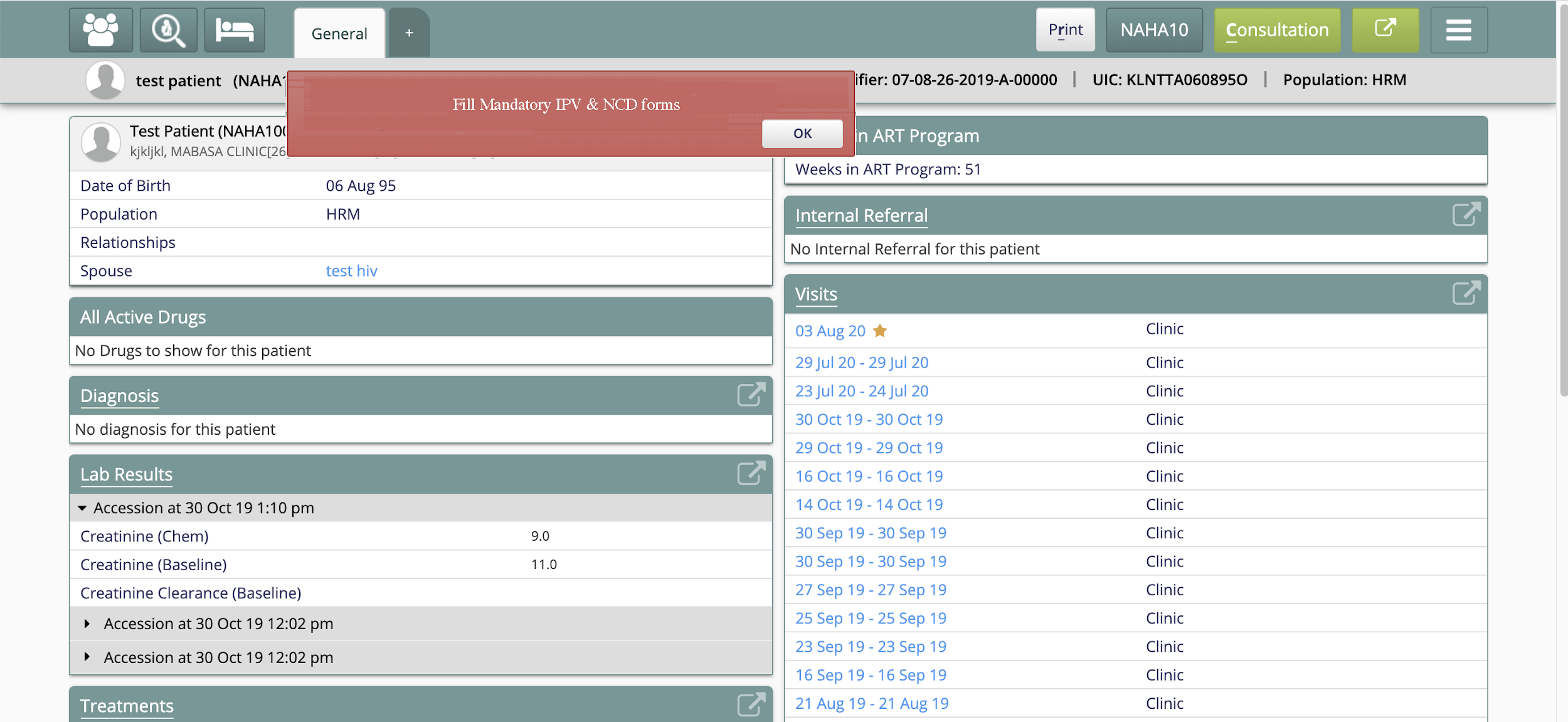
## 3. Types of notifications

The notifications could be INFO/WARN/ERROR type with appropriate color code.

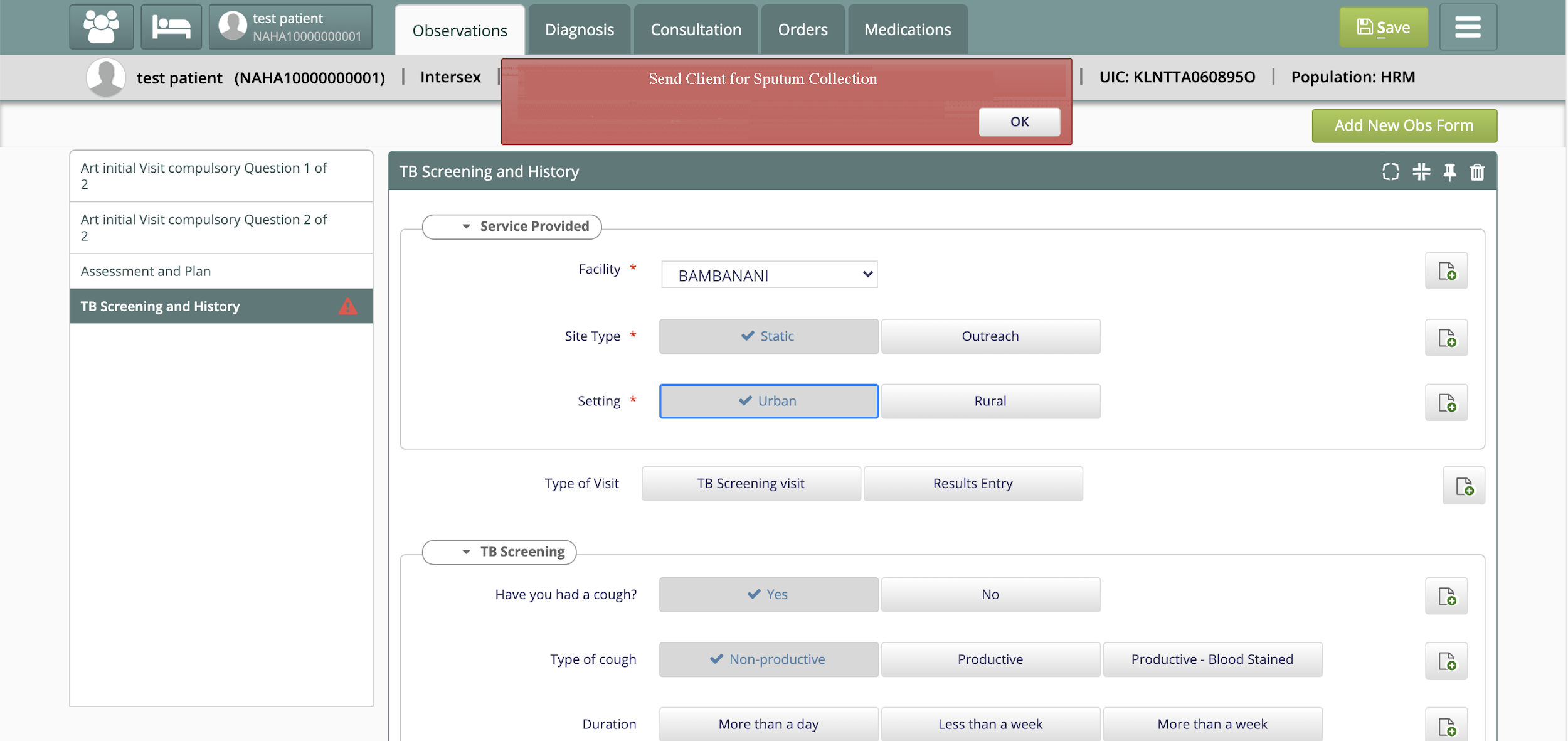
The following 3 types of notifications are supported:

1. Forms notifications. These would be of type ALERT(pop-up dialogue).
   1. Skipped mandatory form alert
   2. Concept based alerts

Mandatory form Alert:

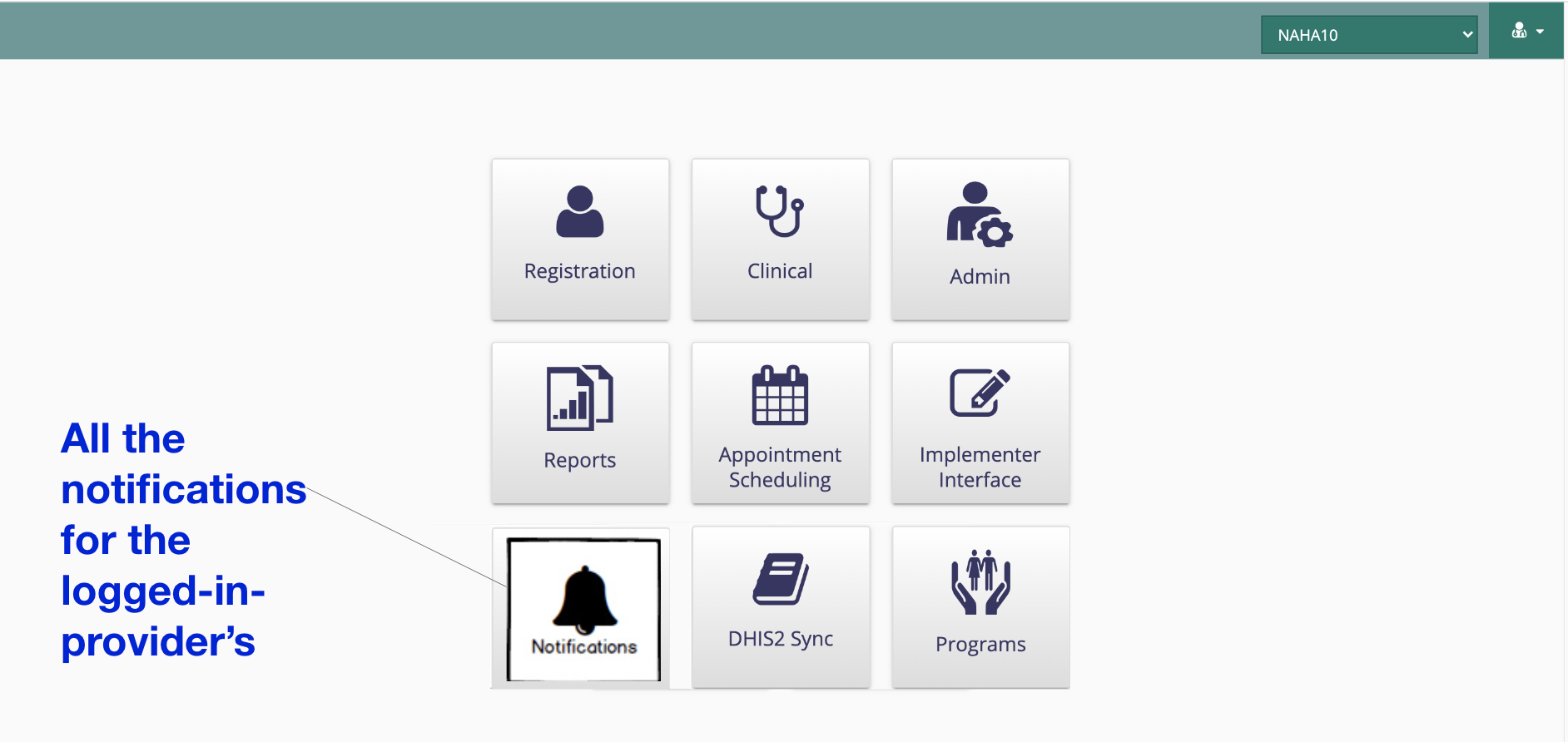


Sample Mock up for Concept based alerts:



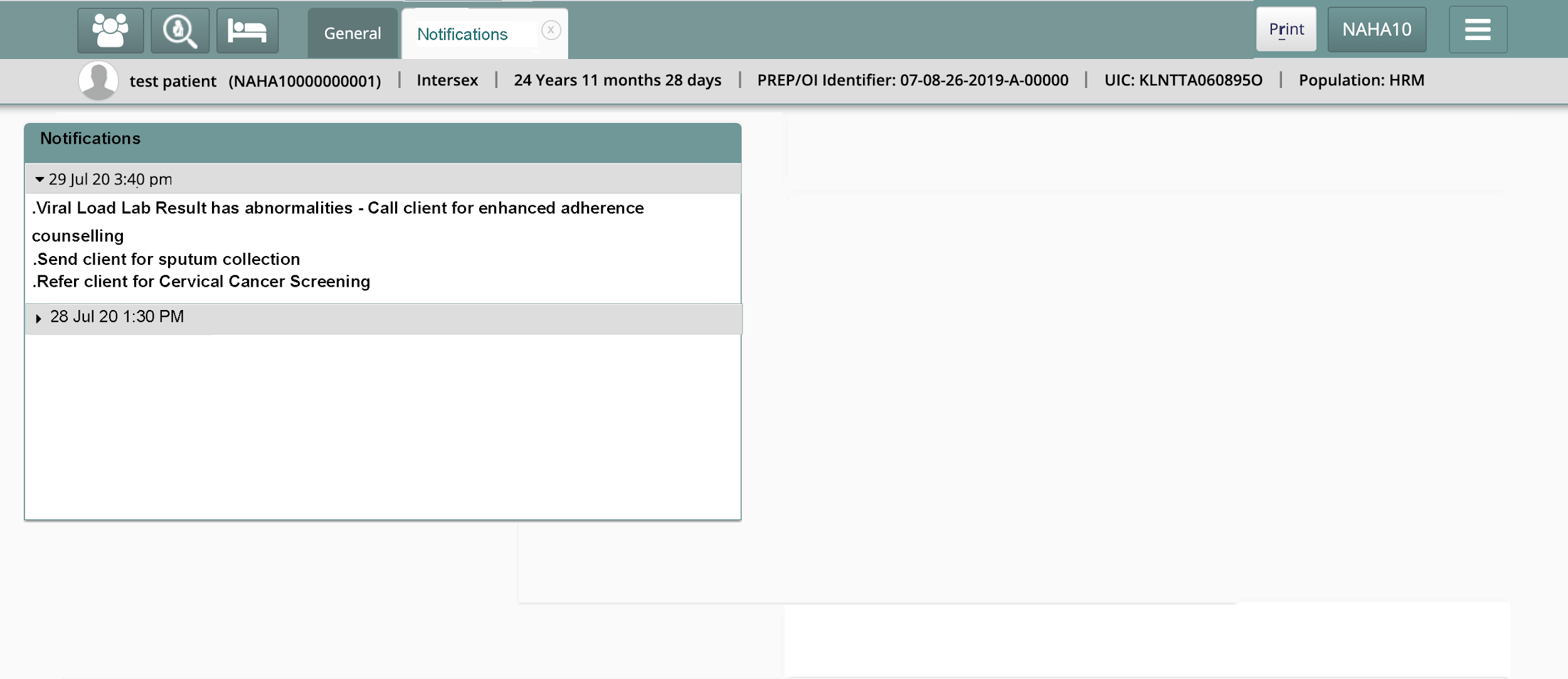
1. Lab result notifications. These would be of type NOTIFICATION.
   1. Abnormality Test
   2. Lab Result Availability
   3. Field Specific Lab Test

**Notifications for Provider for all the clients :**



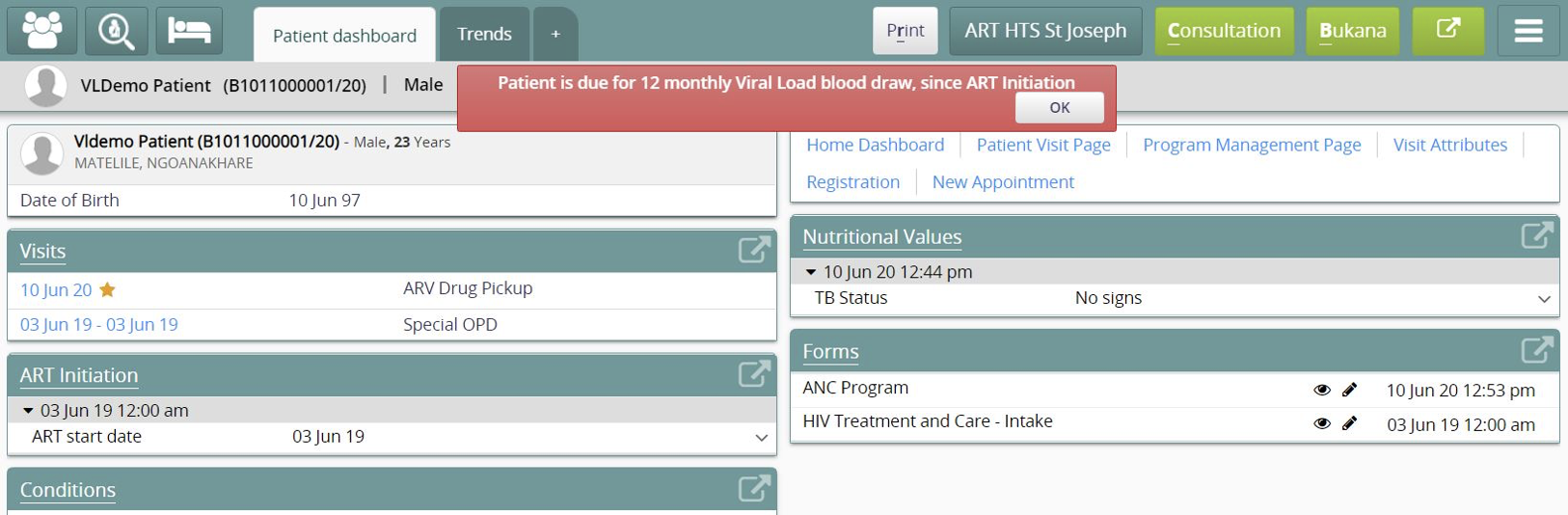


**Notification to provider for a Specific Client ( In Patient Dashboard)**



1. Adherence Monitoring (Scheduled Alerts)

**Sample mockup for Adherence Monitoring:**



## 

## 4. Rules Configuration Approaches

The CDS would need to know the validations/rules for the above mentioned various types of alerts/notifications. These set of rules must be configured (one-time activity) so that the CDS could validate the data against it.

### Rule Definition

Each decision-rule should have the following mandatory fields:

1. Rule-Name : User-defined rule-name
2. Concept/Form unique ID/name : The concept (sets) which require validation.
3. Type of notification : Value could be either Alert or Notification.
4. Message to display : Custom message for this rule to display.
5. Display page : The page where the alert/notification needs to be displayed.

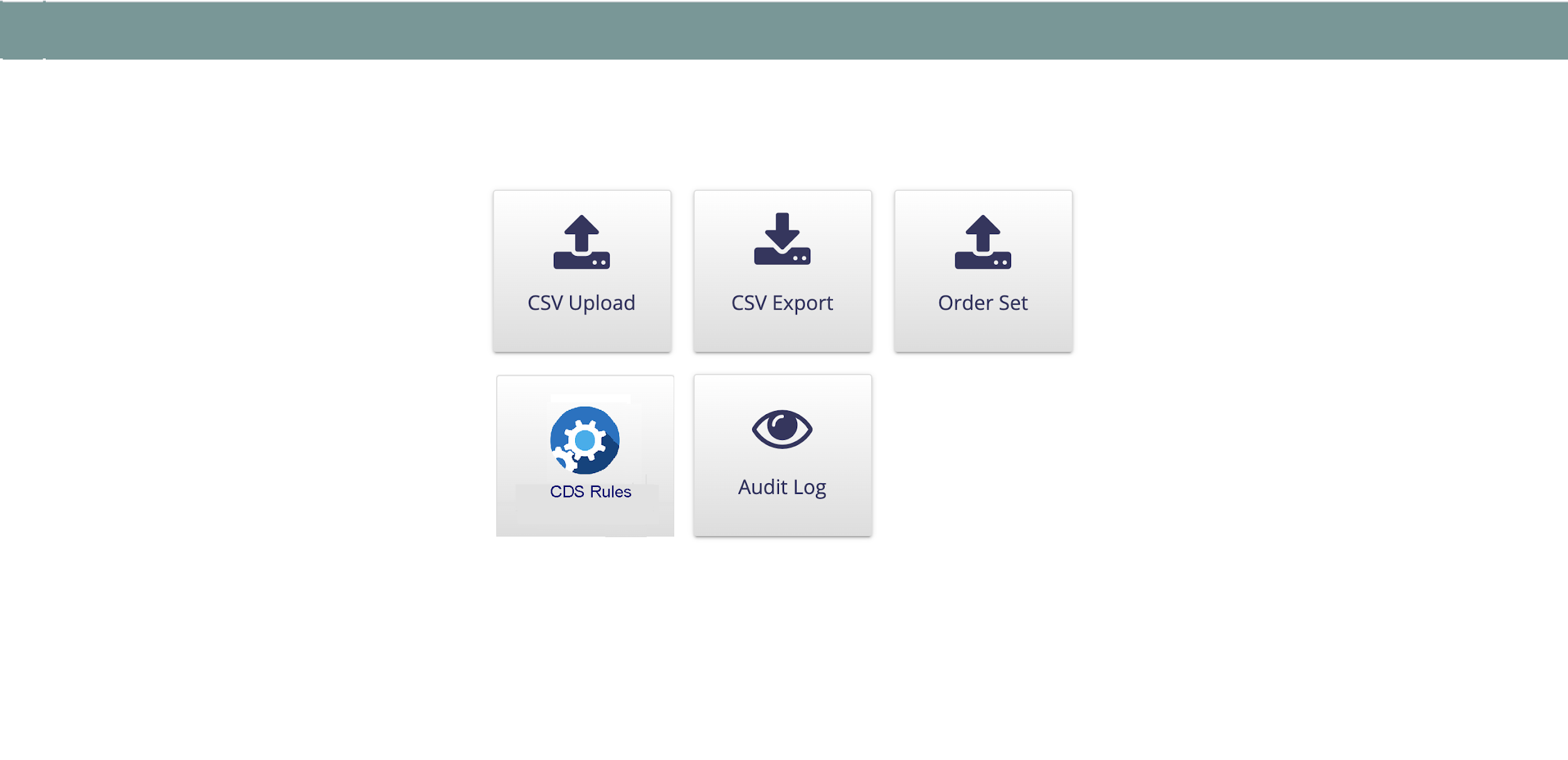
### UI based configuration

In the UI based approach, the end-user can configure CDS rules through Clinical decision rules User Interface.

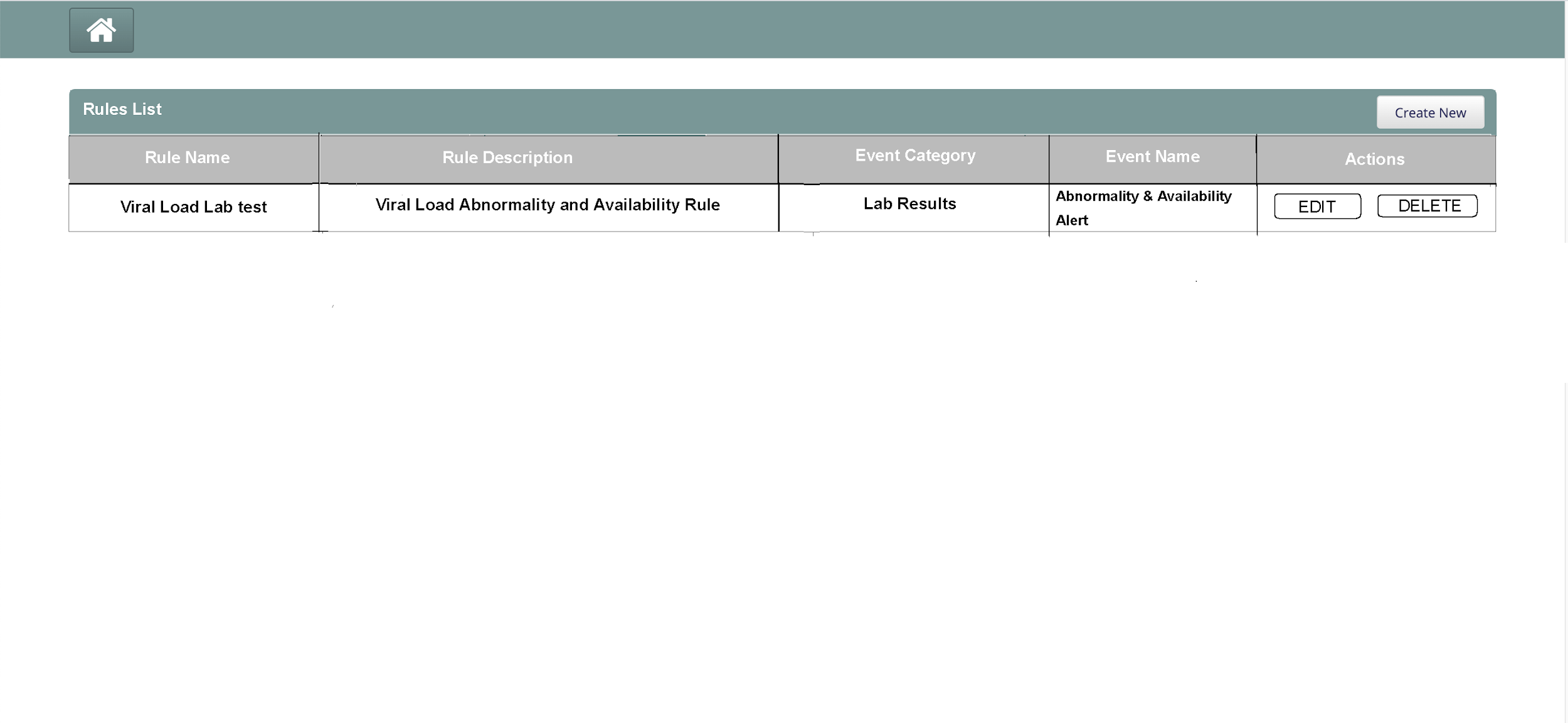
**Navigation:**

Clinical Service → Admin → CDS Rules

**Admin Screen with new Icon for CDS Rules:**



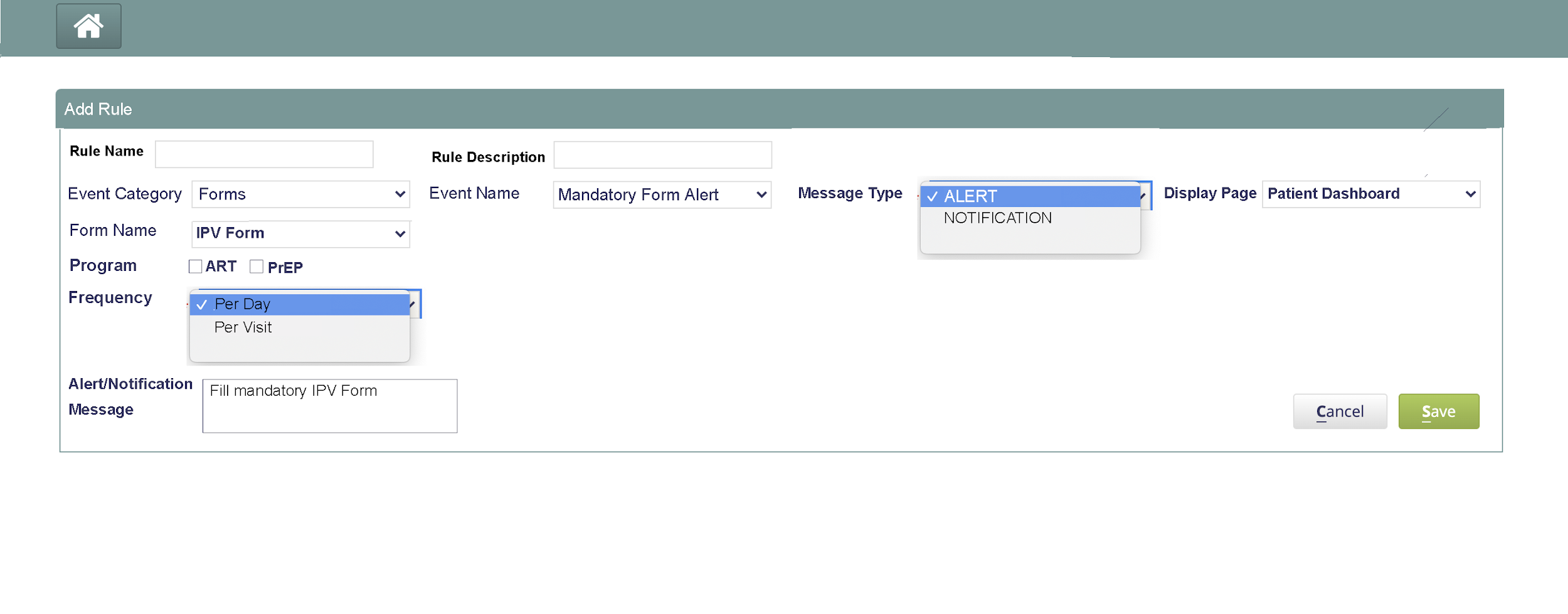
Once user click on CDS rules, User will be navigated to CDS rules list page , where user can view/edit existing rules defined and can add new rules



Create New Button enables users to define various type Events like Form related event, lab results related event , Adherence related events etc.

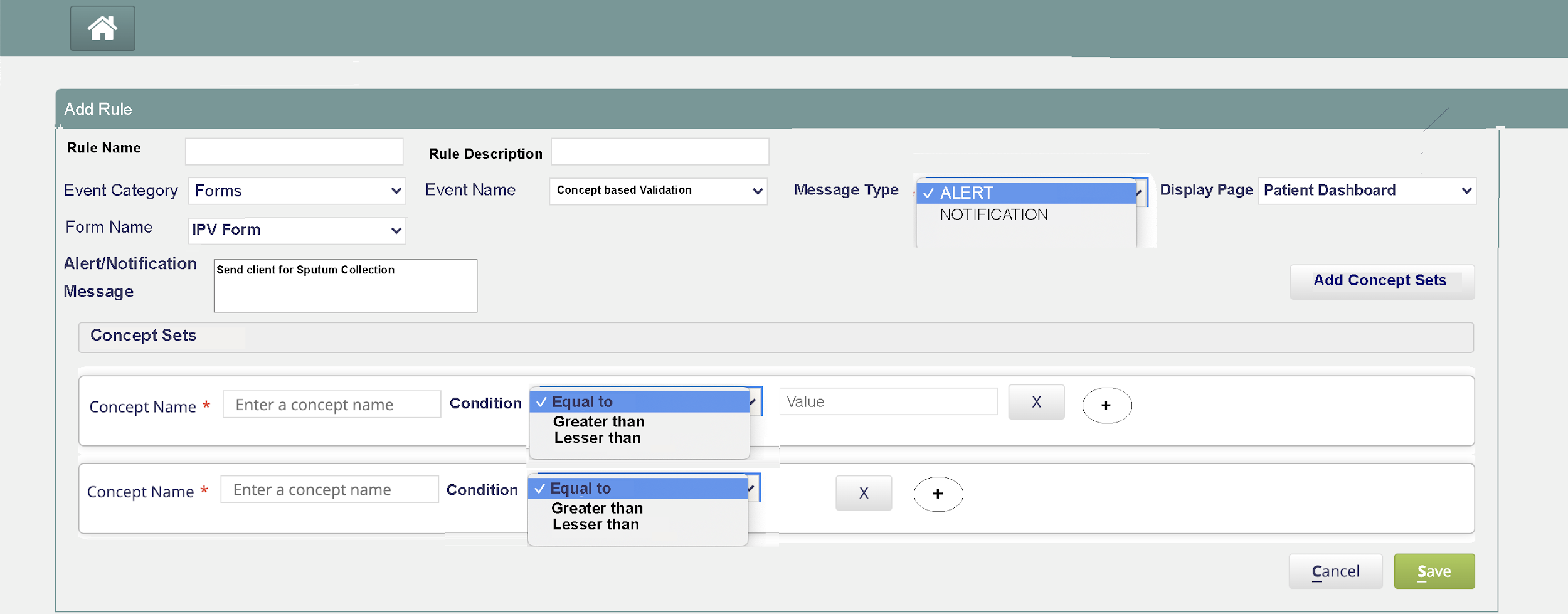
**Forms related Notifications:**

Mandatory Form Rules configuration Screen:



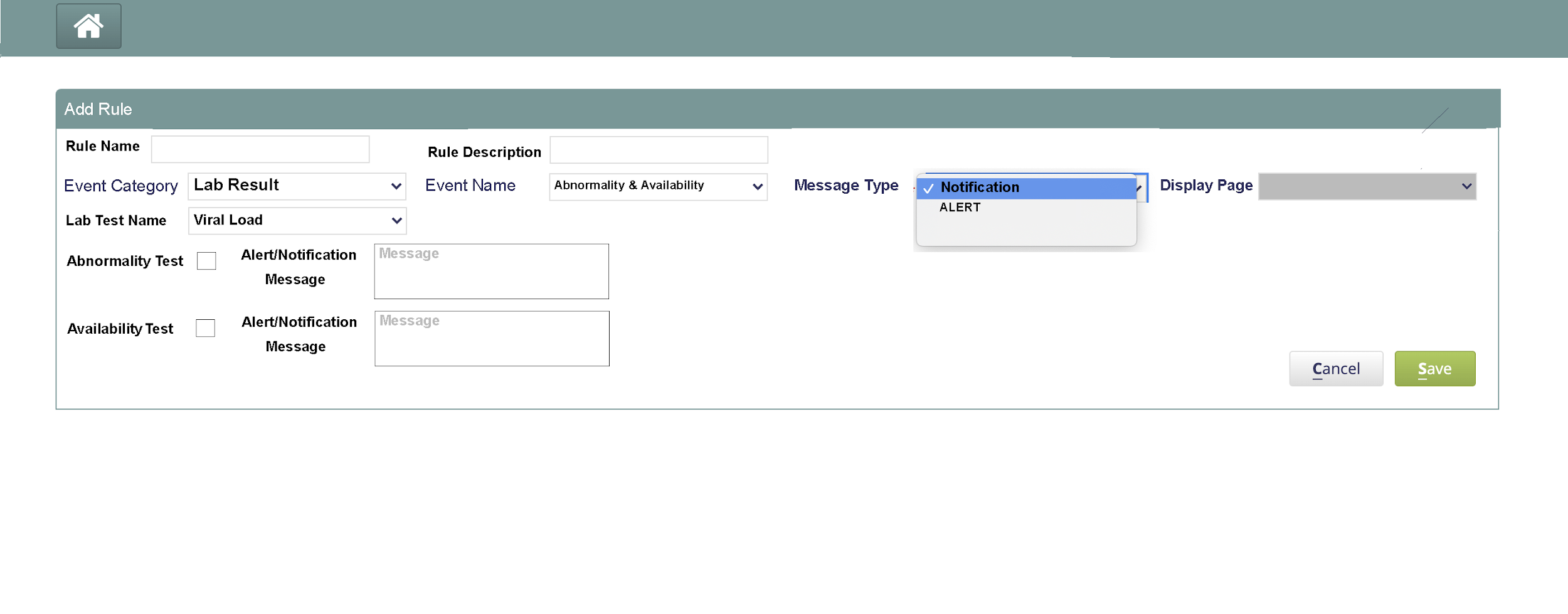
Concept based notifications configuration Screen:

:

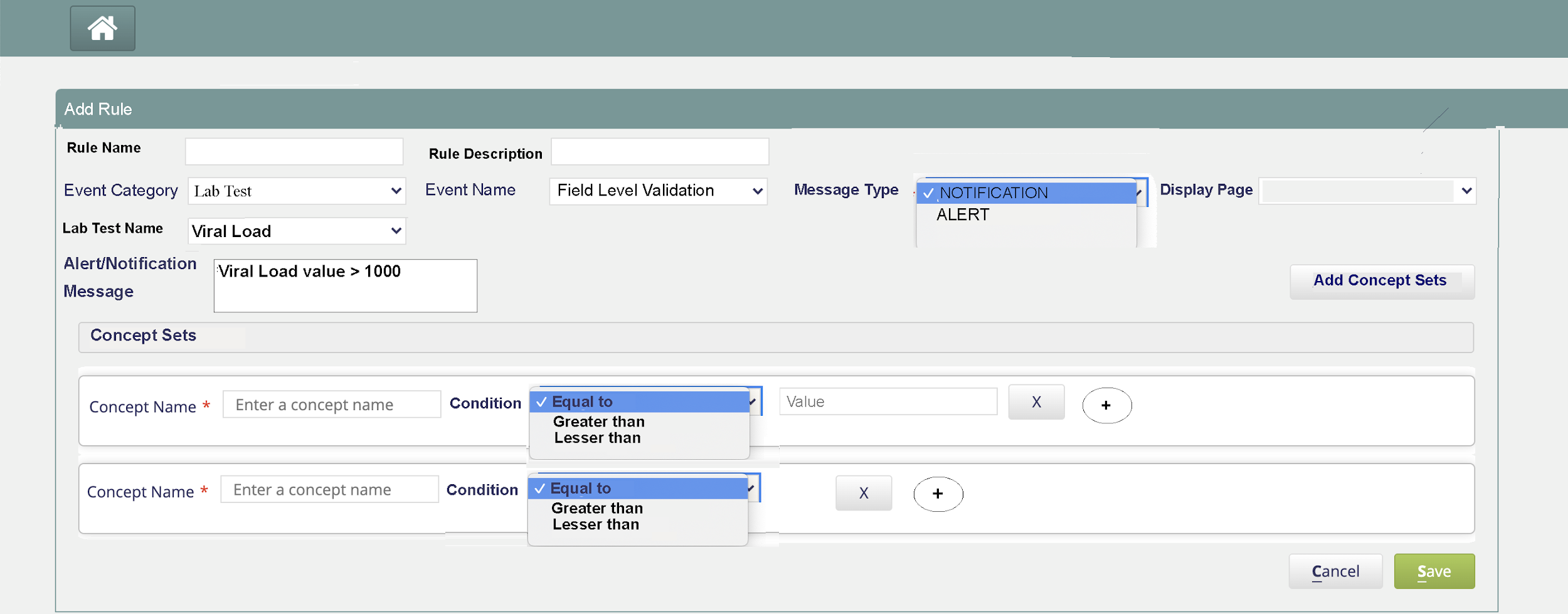


**Lab Results related Notifications configuration Screen**:

Lab results abnormality & Availability events: User will be able to list the Lab results to send notifications to provider who ordered lab test when lab results has abnormalities or once lab result is available for provider to review

****

**Field level lab results events configuration screen:** User will be able to configure notifications that needs to be sent to provider when specific fields in the lab results met the defined values in CDS.

****

**Complexity**: Medium

**Risk**: Low

**Technical details:**

* It could be implemented as an OWA (<https://wiki.openmrs.org/display/docs/Open+Web+Apps+Module>).
* The OWA would have its own configuration for the values for “the type of the notification (alert/notify)” and “the destination/place to display the notification”.
* The OWA would process all the concepts in OpenMRS and make it available to it’s UI in a drop-down format to be selected by the end-user.
* Upon saving the rule, the OWA would save the new rule to a rule-store.

#### **Advantages**

1. Freedom to implement in React. No worries to support AngularJS.
2. Ease of use with the simple GUI.
3. All the concepts are pre-loaded to be selected for setting the rules.
4. No need to look-up the exact concept’s unique ID/name

#### **Disadvantages**

1. The Bahmni-Connect PWA sync would require an additional step. Need to read the rules from the rules-store, save it in a JSON format in the bahmni-config, and push to the PWA.

### Configuration based

In the configuration based approach, the end-user could navigate to the ‘*bahmni-config → openmrs → apps → clinical → cdsRules.json’*. For each new rule a new key-value would be added to the JSON. For a single rule, end-user would have to select from various drop-downs for the concept/form, the type of the notification (alert/notify) and the destination/place to display the notification. The end-user could also enter the message to display in a free text area.

|  |
| --- |
| {  "Bahmni.Alerts" : [  {"ruleName" : "BP Check",  "ruleDescription" : "Check if the BP is in normal range",  "category":"Forms",  "subCategory":"Concept level validation",  "formName": "NCD Form Template",  "message": "BP is higher than normal.",  "displayPage": "Form on save",  "conceptSet" : [  {  "conceptName" : "Systolic (mm Hg)",  "conceptCondition" : ">",  "conceptValue" : "140"  },  {  "conceptName" : "Diastolic (mm Hg)",  "conceptCondition" : ">",  "conceptValue" : "90"  }  ]  }  ],  "Bahmni.Notifications" : [  {"ruleName" : "BP Check",  "ruleDescription" : "Check if the BP is in normal range",  "category":"Lab Result",  "subCategory":"Abnormality & Availability",  "formName": "Viral load Template",  "message": "Viral load results are abnormal",  "displayPage": "Patient Dashboard",  "conceptSet" : [  {  "conceptName" : "Systolic (mm Hg)",  "conceptCondition" : ">",  "conceptValue" : "140"  },  {  "conceptName" : "Diastolic (mm Hg)",  "conceptCondition" : ">",  "conceptValue" : "90"  }  ]  }  ]  } |

**Complexity**: Low

**Risk**: High

#### **Advantages**

1. No additional changes to support the PWA sync. The rules are immediately pushed to the PWA in the initial sync.

#### **Disadvantages**

1. Prone to manual error.
2. Need to know the exact concept’s unique ID/name.

## 5. Rule Store

This is a persistence store in MySQL for all the decision-rules created by the end-user for Bahmni in *‘Clinical-Dashboard → Admin → Clinical Decision Rules’*.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rules Store | | | | | | | | | |
| RuleId | RuleName | Type | Category | SubCategory | FormName | ConceptSet | Message | DisplayPage | Enabled |

## 

## 6. Alert Store

Each alert once generated would be saved in an Alert-Store by the CDS engine. It would be used for displaying the complete list of alerts for the patients generated so far.



The schema of the alert-store would be as below:

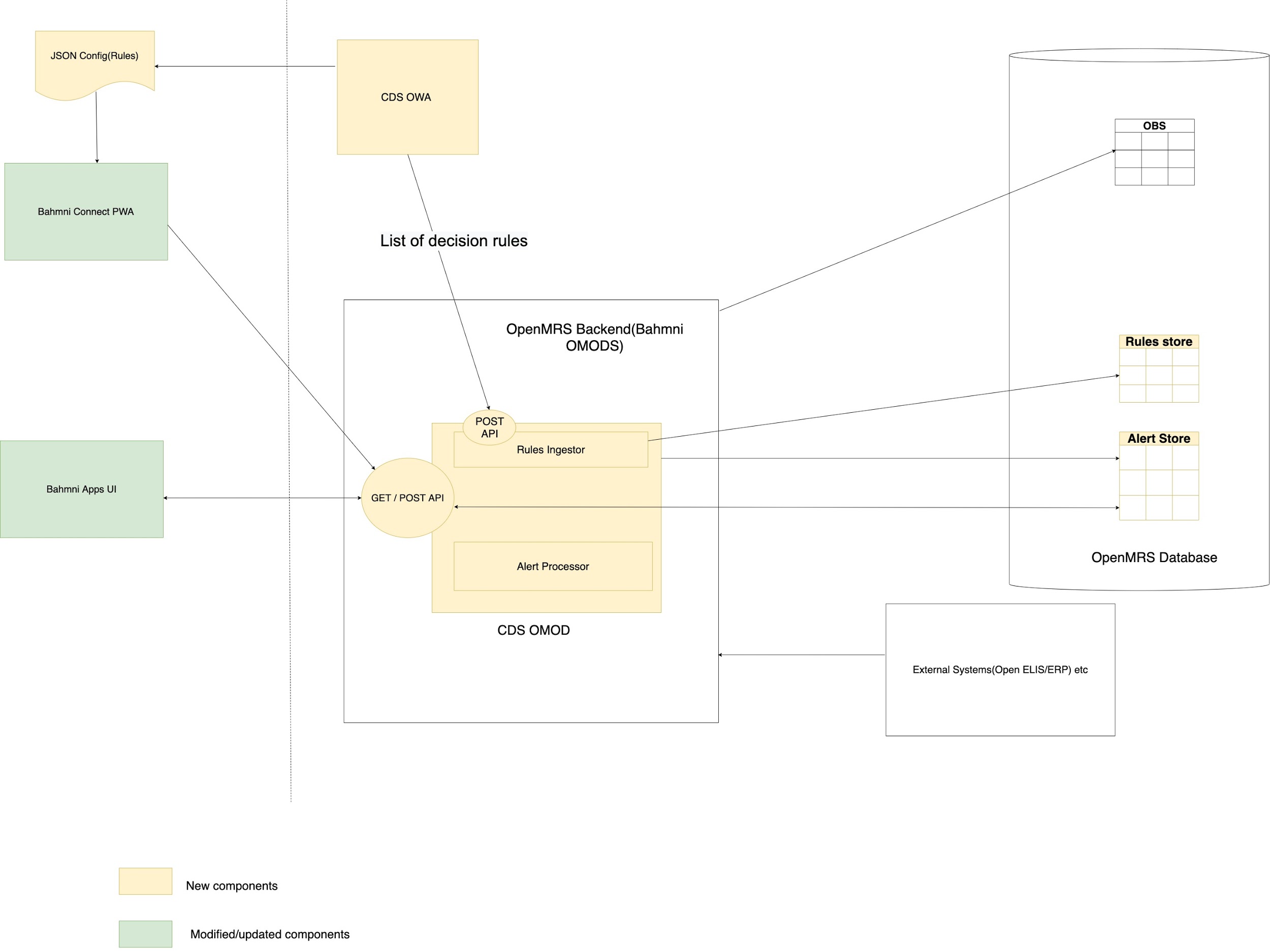
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alert Store | | | | | | | | |
| Id | RuleId | Type | Read | DateTime | EncounterId | PatientId | ProviderId | ProgramId |

## 

## 7. The CDS engine

### Architecture Diagram

The CDS engine consists of a front-end layer and a back-end module. Following is the high level architecture diagram.

[](https://drive.google.com/file/d/1JbK14vmSHlibIb5CG8HsocPZOM_vNfRK/view?usp=sharing)

Following are the various approaches for the CDS engine design.

### 

### **Approach 1:**

#### Front-End layer

As some of the notifications are required to be displayed immediately, CDS would have a front-end layer.

1. Bahmni-UI and Bahmni-Connect would be updated to facilitate various integration points with the CDS engine using this layer.
2. **The front-end layer would NOT load the decision-rules.**
3. This layer would be invoked every time the workflow touches these integration points (egs. Forms onSave event, AtomFeed data in-flow from OpenELIS etc.)
4. The data (which required validation) would be sent to the back-end module using a RESTful API (POST method).
5. Depending on the response an Alert would be displayed in the UI with the message (as preconfigured in the ‘CDS Decision Rules’).

#### Back-End module

1. The back-end module would be developed as a Bahmni OMOD (<https://wiki.openmrs.org/display/docs/Creating+Modules>).
2. **In this approach, the engine is completely centralized in the OMOD.**
3. This module would have a rule-ingestor which would pull the configured decision-rules from the OWA / JSON file.
4. **It would load all the decision-rules from the OWA / JSON and keep it in-memory.**
5. It would be responsible for processing the data flowing in from various subsystems:
   1. Bahmni-Web & Bahmni-Connect
      1. Once the front-end layer posts the data via the RESTful API to this back-end module, it would validate it against the decision-rules.
      2. If the conditions match then an entry would be made in the Alert-Store for the alert.
      3. It would collate all the messages and return response to the front-end layer (so that it may display an alert).
   2. OpenELIS
      1. As of today, the data flows from OpenELIS to Bahmni (OpenMRS) via an atom-feed flow.
      2. The engine would be integrated to listen to the atom-feed.
      3. The existing atom-feed pushes the data into the OpenMRS database.
      4. The CDS engine would listen to this data and validate against the decision-rules.
      5. If the conditions match then an entry would be made in the Alert-Store for the notification.
   3. Other 3rd party subsystem
      1. The CDS engine would be required to integrate with that subsystem’s adapter to capture the data in-flow.

**Advantages**:

1. The CDS engine is centralized in the back-end module.
2. Configurations are loaded only once.

**Disadvantages**:

1. Bahmni-Connect cannot display instant alerts. Only after syncing the data back to the Bahmni server, would it be possible to show the alerts as the CDS engine is in the back-end module.

### 

### **Approach 2:**

#### Front-End layer

As some of the notifications are required to be displayed immediately, CDS would have a front-end layer.

1. Bahmni-UI and Bahmni-Connect would be updated to facilitate various integration points with the CDS engine using this layer.
2. **The front-end layer would also load all the decision-rules from the configurations and keep it in-memory.**
3. This layer would be invoked every time a new encounter is created in the workflow touches these integration points (egs. Forms onSave event, Registration onSave event, other internal events etc.)
4. **Upon invocation the data (which required validation) would be validated against the decision-rules.**
5. If the conditions match then send the data to the back-end module using a RESTful API (POST method) which would store the alert in the Alert-Store.
6. Then an Alert would be displayed in the UI with the message (as preconfigured in the ‘CDS Decision Rules’).

#### Back-End module

1. The back-end module would be developed as a Bahmni OMOD (<https://wiki.openmrs.org/display/docs/Creating+Modules>).
2. This module would have a rule-ingestor which would pull the configured decision-rules from the OWA / JSON file.
3. **It would load all the decision-rules from the OWA / JSON and keep it in-memory.**
4. It would be responsible for processing the data flowing in from various subsystems:
   1. Bahmni-UI & Bahmni-Connect
      1. Once the front-end layer posts the data via the RESTful API to this back-end module, an entry would be made in the Alert-Store for the alert.
      2. Return response to the front-end layer (so that it may display an alert).
   2. OpenELIS
      1. As of today, the data flows from OpenELIS to Bahmni (OpenMRS) via an atom-feed flow.
      2. The engine would be integrated to listen to the atom-feed.
      3. The existing atom-feed pushes the data into the OpenMRS database.
      4. The CDS engine would listen to this data and validate against the decision-rules.
      5. If the conditions match then an entry would be made in the Alert-Store for the notification.
   3. Other 3rd party subsystem
      1. The CDS engine would be required to integrate with that subsystem’s adapter to capture the data in-flow.

**Advantages**:

1. Bahmni-Connect also has the front-end layer. So the alerts can be displayed instantaneously.

**Disadvantages**:

1. The CDS engine is split between front-end layer and back-end module.
2. Duplication of decision-rules loading (in both front-end layer and back-end module).

## 

## 8. Future Enhancements

* In the event of an Alert message displayed to the end-user, we could also provide a ‘redirect url’. On clicking it, the end-user would be redirected to the pre-configured url to address the alert message.

This ‘redirect url’ could be added when creating the decision-rule, as an optional field.