Inspection application

Technical Specification (draft)

Contents

[Application Overview 1](#_Toc530746569)

[Software Requirements 2](#_Toc530746570)

[Setup for Application Development Environment 2](#_Toc530746571)

[Supported Channels and Platforms 3](#_Toc530746572)

[Build Settings 3](#_Toc530746573)

[Android 3](#_Toc530746574)

[iPhone (XCode Settings) 3](#_Toc530746575)

[Client Application 3](#_Toc530746576)

[Application Flow 3](#_Toc530746577)

[Overview of Forms 6](#_Toc530746578)

[Components Used in the Application 15](#_Toc530746579)

[Global variables 18](#_Toc530746580)

[Offline Implementation 18](#_Toc530746581)

[Fabric Application 19](#_Toc530746582)

[Backend Database 19](#_Toc530746583)

[Schema 19](#_Toc530746584)

[Tables related to master data of Assets 20](#_Toc530746585)

[Tables related to master data of Measurements 20](#_Toc530746586)

[Tables related to Inspection 20](#_Toc530746587)

[Tables for capturing the Measurement Data during Inspection Execution 21](#_Toc530746588)

[Table(s) for User/Role management 21](#_Toc530746589)

[Operations performed on the tables 21](#_Toc530746590)

# Application Overview

The Inspection application supports the creation of Inspections against the assets and the execution of the inspections by capturing the values for measurements associated with the inspections. It supports capturing of the images during the execution of an inspection. The application also shows the history of previous inspections performed on the selected asset.

Admin and Technician are two types users in the context of this application. Admin user can create an inspection and assign it to a technician. Technician can execute an inspection (capture the values for measurements) assigned to him. Technician can also create an inspection that he can assign it to himself.

Based on the role of the logged in user, the application shows the appropriate landing screen. For Admin users, the landing screen would be Inspection Creation where it would show the list of Assets. The admin selects an asset and creates the inspection by selecting one of the applicable measurement set.

For the Technician user login, the landing screen would show the list of inspections assigned to the logged in technician. The technician would select one of the inspections and proceed to capture the values for the measurements.

The application uses the Sync V2 feature of the Kony Fabric to work in the offline mode. When the application is launched for the first time, the data is synced from the backend to the device database. Post this the application can work offline. The inspections can be created and executed even though there is no network. The user can perform manual sync by using the ‘Sync’ option available in the hamburger menu.

# Software Requirements

Kony Visualizer Enterprise 8.2.x and above

Kony Fabric 8.2.x and above

mySQL 5.7.x

# Setup for Application Development Environment

* Install mySQL (version 5.7.x),
* Import the provided DB Dump into the mySQL instance. This DBDump consists of master data of assets and measurements. There are also some sample inspections
* Make sure to have the 8.2.x or higher version of Visualizer and Fabric environment
* Import the provided Visualiser project of the Inspection Application. Please ensure to check ‘import services into mobile fabric’ while importing the project into Visualiser
* From the MF console, change the DB connection parameters configured in the object service of the MF application
* Setup the custom identity service endpoint:
  + Import the provided eclipse project into your eclipse setup
  + Modify the spring-config.xml to update the DB connection details
  + Generate a war file from this project
  + Create a Fabric application. Upload the above created war (by navigating to Client App Assets->Web)
  + This fabric application consists of only war file. Publish the Fabric application. Please note that this Fabric application does not work on MTMW.
* Configure the Fabric application (that was associated with Visualiser project) with the above published identity service endpoint url.
  + Go to custom identity and provide the custom identity service end point as ‘<*URL where the above Fabric application (the one with the war file) is published*>/services/LoginSvc’
* Publish the Fabric application (the one that was associated with Visualiser project).
* Build and launch the application
  + Refer to the ‘Build Settings’ section for XCode settings to be done for iPhone build
  + Build and launch the application for iPhone and Android
  + User details for login are available in the ‘inspection\_user’ table

# Supported Channels and Platforms

The application is supported on iPhone and Android mobile

# Build Settings

## Android

Android Manifest permissions

WRITE\_EXTERNAL\_STORAGE

CAMERA

ACCESS\_COARSE\_LOCATION

ACCESS\_FINE\_LOCATION

## iPhone (XCode Settings)

-Camera Usage permission

-Photo usage permission

-Deployment target should be set to 11 or higher

-App Transport Security Settings->’Allow Arbitrary loads’ should be set to ‘yes’

# Client Application

## Application Flow

Pre Login Flow:

No Data

No Data

frmLogin

frmIntro

frmSplash

Technician Flow:

frmInspectionExecution

Userdata

Userdata +InspDetails

frmInspectionsList

frmLogin

Userdata +AssetId

Userdata +InspDetails+CapturedData

frmInspectionHistory

`

Userdata +AssetId

Userdata

frmReview

frmInspectionConfirmation

Userdata + Inspection ID

Inspection creation flow of technician:

frmInspectionCreation

Userdata

frmLogin

Userdata

frmInspectionsList

Userdata

Userdata +AssetDetails

frmInspectionConfirmationScreen

Userdata +AssetDetails

frmMeasurementAssignment

Admin Flow

frmInspectionCreation

frmLogin

Userdata

Userdata +AssetDetails

Userdata

frmInspectionConfirmationScreen

Userdata +AssetDetails

frmMeasurementAssignment

## Overview of Forms

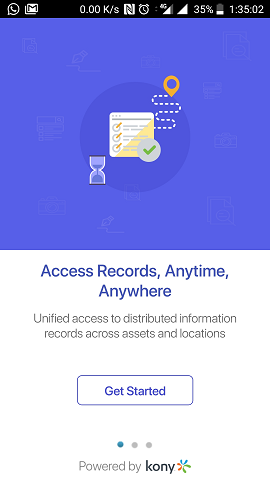
Some of the functions are common throughout all forms. These functions are named according to the events they triggered on

1. onNavigate : used to modify predefined UI and saving the parameters passed to a form level variable.
2. onFormPreshow : Loading screen handler
3. onFormPostShow: On Post show, different data Objects are fetched based on the form requirements.

**frmSplash**

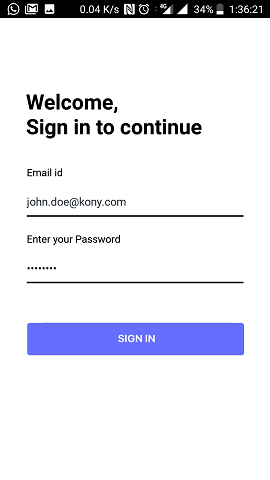
This form works as a splash screen and if the app is opened for first time then it will be automatically navigated to frmIntro otherwise to frmLogin from second time onwards. Navigation logic is called on form post show.

**frmIntro**

****

Show the Inspection App details and on ‘Get Started’ calls startsync to sync the application data from backend.

**frmLogin**

****

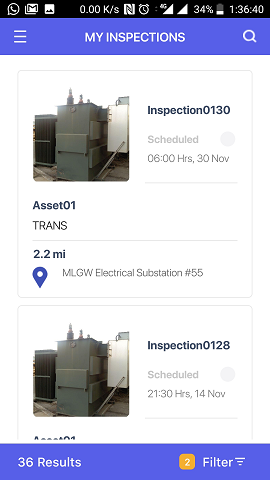
Login form for authentication and authorization. Based on user type(admin/technician), the next form will be shown. No Remember me or encryption of password is done for now.

onFormPostShow: sync configurations are set up.

doLogin : Performs the login using the custom identity service. If there is any change in the custom identity, same needs to be reflected in this function

\_loginSuccess: User data is retrieved which is passed to the next form as navigation data.

**frmInspectionList**



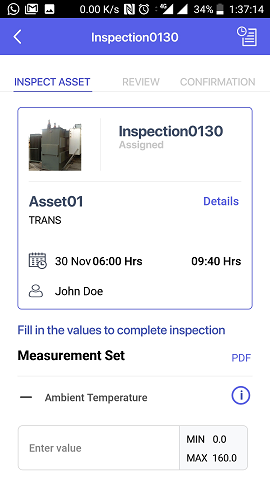
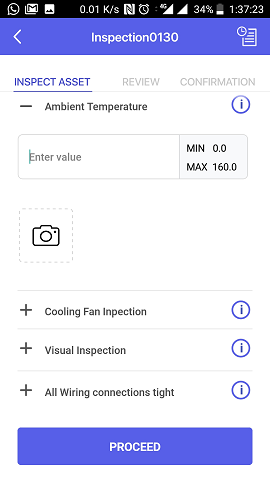
Shows the list of inspections assigned to particular member using a segmented UI. User can search and filter the inspections. On Click of an inspection in the segment, user will be navigated to inspection execution if the inspection is scheduled. If the user clicks on a completed inspection, it shows the measurement values captured in that inspection

onNavigate: navigation data is stored and hamburger data is set.

onFormPostShow: In this function, getInspectionListData is called which fetches asset, asset\_location, asset\_type and inspection data model .

processRecords: process the records from each data model to generate arrays of inspections JSON containing inspection and asset details.

**frmInspectionExecution**

This form shows the inspection details like the associated Asset details and the measurements to be captured. For showing the measurement details, the measurement component is used. Measurement component supports the measurement types: Number, Text, List of values and Boolean. Depending on the data set to the component, it would internally creates the components for number/text/list/Boolean.

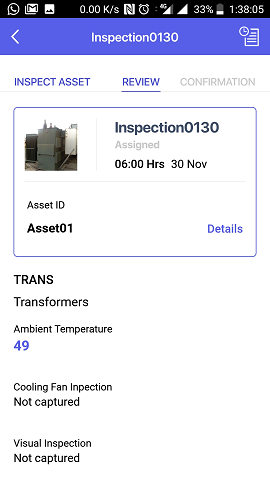
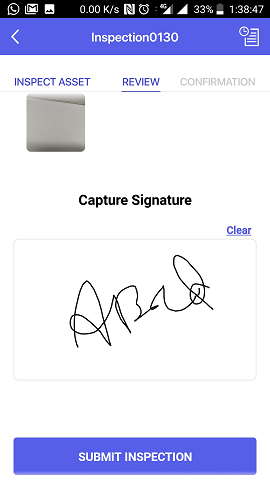
Clicking on info icon of a measurement, shows the previous values of that measurement. This info is displayed in a card component.

onNavigate: navigation data is saved to form variable and UI is reset for capturing the measurement data. Data binding for cards is done here since those details are coming as a part of navigation data.

onPostShow: getInspectionListData is called here which internally fetches inspection\_user, asset\_groupnames, groupnames , inspection\_measurement, measurementset\_measurementrange, measurement\_range, measurement, measurement\_hstry data model based on the inspection id.

populateInspectionInfo: This function populates all the data in the form like asset description, measurement info. This function is also calls processMeasurement internally which pass the data to measurement component to create measurement containers based on responseType.

**frmInspectionReview**

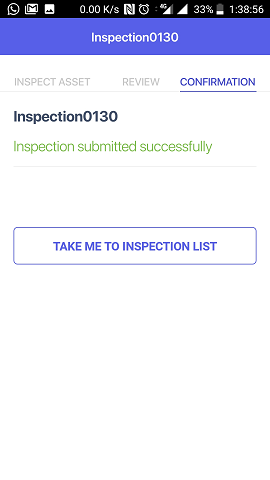
 

This form shows the captured measurement values from the form ‘frmInspectionExecution’ so that user can review the values. It uses the signature capture component to allow the user to signOnce user signs and taps ‘proceed’, the inspection status changes to ‘Completed’.

onNavigate: navigation data contains the captured measurements and user details which are saved into form level variables.

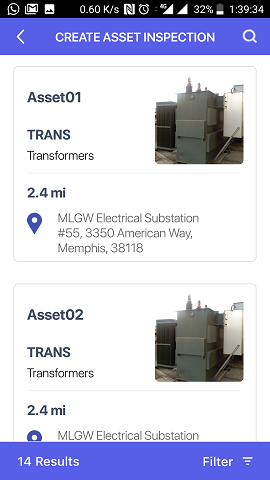
onFormPostShow: on post show it check the previous form to check whether it has to show completed Inspection UI or review UI and set the data accordingly.

**frmInspectionConfirmation**



Show the Inspection confirmation screen once the inspection completed.

**frmInspectionCreation**



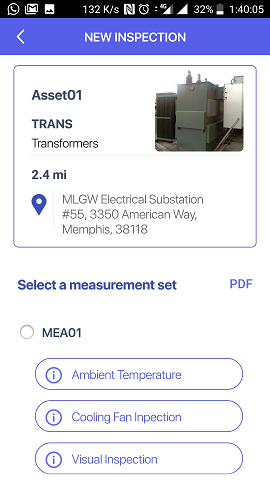
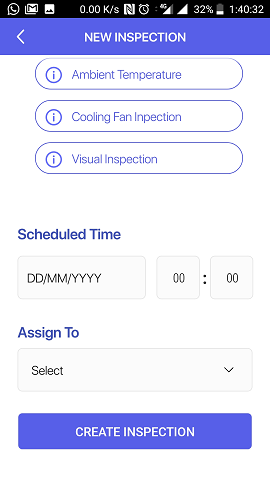
Shows the list of Assets. User can Search and filter for any assets on the list.

onNavigate: navigation data is saved to form level variable and UI is modified based on whether the user admin/member(technician).

onFormPostShow: getInspectionListData is called here which eventually fetches asset, asset\_location and asset\_type.

processRecords : process the record based on segment requirement.

**frmMeasurementAssignment**

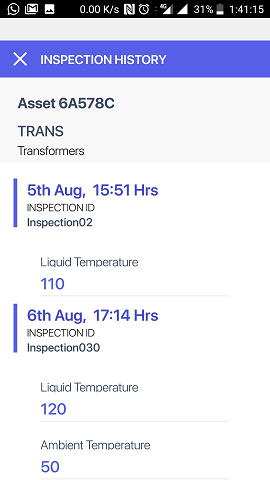
This form shows the measurement sets associated with the selected asset. It also lists the associated measurements for each measurement set

The user can select one of the measurement set, scheduled time and technician to create an inspection

onNavigate: navigation data is saved to form level variable and UI is modified based on whether the logged in user is admin/member (Technician).

onFormPostShow: asset\_measurement\_set, measurementset\_measurementrange, measurement\_range, measurement, inspection\_user and measurement\_hstry data model are fetched in this function.

**frmHistory**

****

History of inspections associated with an asset id are is shown in this form. All the past inspections are shown with the inspection dates and the respective measurements are shown in this form

onNavigate: navigation data is saved to form level variable and UI is modified based on whether the logged in user is admin/member (Technician). This function also fetches the data from inspection, measurement\_hstry, measurement\_images, media, measurement and measurement\_range objects.

## Components Used in the Application

Following market place components are used in the application

* Signature capture component
* PDF Viewer component
* Alert popup component
* Animated textfield component

Following components have been developed as part of this application

* Measurement list component (internally uses measurement list, measurement range, measurement Boolean, measurement description components)
* Image gallery component
* Image upload component
* Login Component
* Infocard component
* Loading screen component

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component** | **Component Description** | **Property/Method/Event** | **Type** | **Description** |
| measurement | This component is a parent container for different measurement types (number, Boolean, text, value from a list of values). There are separate child components for each measurement/response type. Based on the data passed to setData method, the component creates the child components dynamically. | title | Property | To add title text of the parent container |
| onMeasurementDone | Event | Trigger when measurement value are obtained |
| setData | Method | It takes a javascript object as a parameter. Its an array of measurements. Each element will have the details of a measurement like its name, its type, range of values. Depending on the type, respective child component is instantiated and sets the data to it. All the child components are then added to this component |
| getResult | Method | gets all the Consolidated data (the measurement values entered by the user) from all child component |
|  |  |  |  |  |
| ImageGallery | Component for showing images in gallery type of view. It has a thumbnail view and actual image view. User can click on the thumbnail to see the actual Image. | setImage | Method | Sets the images to the gallery. This method takes an array of images as the first parameter. The second parameter indicates whether the image data is base64 or URL. |
|  |  |  |  |  |
| imageupload | This component helps to capture from native camera or gallery of the phone and add to a container. | getResult | Method | It returns the array of images captured/selected |
| setData | Method | sets the measurement id to the component to track images selected against a measurement |
| onAddImageClick | Event | This event triggers when user clicks on the Camera Icon of the component |
| onClickImages | Event | Triggers when added images are clicked upon. |
|  |  |  |  |  |
| InfoCard | Info card component shows details of the data in a pre-defined template. | setData | Method | set the predefined formatted data to the info card. |
|  |  |  |  |  |
| LoginComponent | Login component utilizes animated text field component and do login by connecting to identity service | onSignIn | Event | Triggers when login button is clicked |
| getEmail | Method | gets the text from username textfield |
| getPassword | Method | gets the text from password textfield |
| setLoading | Method | sets the sign in button image |
| resetLoading | Method | removes any custom sign in button and add a default login image on button |
| setUserNameAndPassword | Method | set the username and password |
|  |  |  |  |  |
| loadingScreen | Show the loading screen based on the input. The loading screen is shown on the top of the screen. It supports 4 types of loading screens. Each type shows the message in a different color (blue/green/gray/red) | show | Method | Shows the loading screen component. It takes 2 parameters. First parameter is 'message' to be shown. Second parameter indicates the background color used for the loading screen. The supported values for the second parameter are 1/2/3/4 where 1 for blue, 2 for gray, 3 for green and 4 for red |
| hide | Method | hides the loading screen component |

## Global variables

The module global\_module.js has the global variables defined.

The variable ‘app\_constant’ defines the prefixes used for asset id, inspection id and the measurement set.

The variable ‘DATA\_MODEL’ defines all the objects of the object service used within the application. If there is any change in name of the objects in the Fabric application, same needs to be reflected in this variable

The variable OBJECT\_SERVICE defines the name of the object service. If there is any change in the name of the object service, same needs to be reflected for this variable.

InpsectionUtil has been defined with a set of utility functions used in the application.

## Offline Implementation

The application uses Sync V2 for its offline capabilities. The application allows login, creation of inspections, execution of inspections, viewing of inspection history even though the device is not connected to the network.

During the first launch of the application, introduction form (frmIntro) is shown which consists of button ‘Get Started’. Clicking this button, performs sync setup by invoking the following api

*KNYMobileFabric.OfflineObjects.setup(setupSuccess.bind(this), setupFailure.bind(this));*

Offline Objects ‘setup’ function initializes the creation of device database and sync environment. Once the setup is successful, the database structure is created without data in the device database.

In the success callback of the setup API, sync is started using the *startSync* API on the object service.

If any specific sizes to be given for upload/download batch size, same can be passed in the options parameter of the startSync API by using the keys ‘uploadBatchSize’ and ‘downloadBatchSize’.

Code snippet:

*var syncOptions={};*

*syncOptions.uploadBatchSize="200";*

*try{*

*var syncObjService= new kony.sdk.KNYObjSvc(OBJECT\_SERVICE.SYNC);*

*kony.application.showLoadingScreen(""," ",constants.LOADING\_SCREEN\_POSITION\_ONLY\_CENTER, true, true,null);*

*syncObjService.startSync(syncOptions,this.successCB,this.failureCB,this.progressCB);*

*}catch(excp){*

*kony.print("Exception: "+excp);*

*kony.application.dismissLoadingScreen();*

*}*

*},*

Apart from initial launch of the application, there is a provision to perform sync manually. In the Hamburger menu, the option ‘Sync’ would perform the sync. Same as the initial launch, it performs the sync on the object service level by invoking the startSync method of the object service.

When an inspection is created, Kony Sync would assign a negative value to the inspection id (primary key). In the application, instead of showing the negative number to the user, we show the inspection as ‘Inspection to Sync’ and suffix it with the value of the inspection id by excluding the negative symbol. Once the sync is performed, it shows the actual inspection id created in the DB.

# Fabric Application

Fabric application consists of an Identity service and an object service

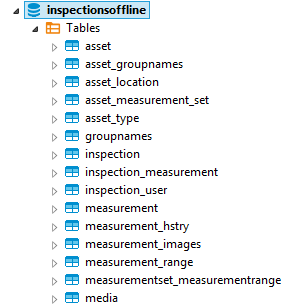
The object service is created using relational database (mySQL Database) end point. The object service is enabled for offline.

Identity service of type ‘Custom Identity’ has been used. Custom Identity is implemented by developing a REST service which queries the database and validate the credentials.

# Backend Database

## Schema

Following is the DB schema on mySQL



## Tables related to master data of Assets

* *asset*

This table consists of the asset details like asset id, asset name, image, description, asset type, asset location

* *asset\_groupnames*

This table has the entries for asset to group mappings. A particular asset can be part of multiple groups.

* *asset\_location*

This table consists of master data of all the locations.

* *groupnames*

This table consists of master data of all groups. It consists of group id, name, description

* *asset\_type*

This table consists of the master data of all asset types.

* *Asset\_measurement\_set*

This table consists of the applicable measurement sets for a given Asset.

## Tables related to master data of Measurements

* *Measurement*:

This table contains the list of all measurements. It consists of measurement id, measurement name and description

* *Measurement*\_range:

This table consists of the details of a measurement. It has the info whether the value of a particular measurement is a number or text or Boolean or a particular value from a list of values. It also consists of the information on the range of values for numeric measurement. It consists of list of values (separated by commas) for the measurements which accepts values from list

* measurementset\_measurementrange:

The table consists of the measurement sets and their associated measurements. One measurement set can be associated with multiple measurements (entries in measurement range table)

## Tables related to Inspection

* *inspection*

This table consists of the information related to inspections. A record is created in this table when a new inspection is created. As the inspection is executed the same record gets updated to reflect the status and other details

* *inspection\_measurement*

This table consists of the info on which inspection is associated with which measurement set. Currently the support is for one inspection can be mapped to one measurement set.

## Tables for capturing the Measurement Data during Inspection Execution

* *measurement\_hstry*

This table is used to capture the actual measurement values taken during the inspection execution.

For a given inspection, given measurement set, given measurement, the measured values are captured in this table.

* *measurement\_images*

For a given measurement history entry, there can be multiple images associated. This table consists of the mapping between a measurement history id with its corresponding media ids of the media table

* *media*

This table consists of the media id and the corresponding base 64 data of the images

## Table(s) for User/Role management

* *inspection\_user*

The users and their roles are maintained in this table. Supported roles are ‘Admin’ and ‘Member’(Technician)

## Operations performed on the tables

Inspection Application does not perform Create/Update/Delete operations on the following tables as they are used as master data. Only GET is performed on the below tables:

*Asset*

*Asset\_groupnames*

*Asset\_location*

*Asset\_measurement\_set*

*Asset\_type*

*Groupnames*

*Inspection\_user*

*Measurement*

*Measurement\_range*

measurementset\_measurementrange

Create/Update/Get is performed on the following tables:

*inspection\_measurement* (create is perfomed at the time of inspection creation)

*inspection* (create is performed when the inspection is created. Update is performed when the inspection is executed)

*measurement\_images* (create is performed when the inspection is executed)

*measurement\_hstry* (create is performed when the inspection is executed)

*media* (create is performed when the inspection is executed)