Forms Module

Mobile App

# Overview

* Developed Forms modules, for a mobile application using React Native Expo and SQLite.

Forms Module Mockups : <https://xd.adobe.com/view/233a5879-9826-4a58-9611-0c81543e35e7-7669/>

* Updated the application which ran on SDK 49 to the newer SDK 51 by updating the expo and other required dependencies.

# Goals

1. *Implement user input dynamic forms with validation and data storage.*
2. *Storing the form data in offline mode in local storage.*
3. *Update the app to target the latest Android version.*

# Workflow

1. Forms Count and Details Listing: Display the count and details of forms on the home page.
2. Dynamic Form Retrieval: Fetch dynamic forms from an API or local SQLite database (having options for offline form download).
3. Filter option : user can choose the asset or location to open the form
4. State Management: Store forms and their values in Redux.
5. Form Submission:
   1. Online Mode: Submit the form to the server using an API.
   2. Offline Mode: Store form data in a local SQLite database.
6. Background Sync: Sync stored form data with the server in the background when online.
7. Update: Updated to Latest Android version and all other dependencies.

# Tools And Technologies

* **Development Tools:**
  + **TypeScript, React Native, Redux Toolkit**
  + **Node.js:** A JavaScript runtime for building server-side applications.
  + **NVM (Node Version Manager):** Allows managing multiple versions of Node.js.
  + **Ngrok:** to run application on expo go mobile app creating a tunnel
  + **npm or Yarn:** Package managers for installing and managing dependencies.
  + **React Native CLI or Expo CLI:** Command-line interfaces for creating and managing React Native projects.
  + **Expo Go app**: App to run and test mobile app
* **Development Environment:**
  + **A code editor (such as VS Code)**: Software for writing and editing code efficiently.
* **Version Control:**
  + **GIT**: A distributed version control system for tracking changes in source code.
* **State Management:**
  + **Redux**
* **Local Database :** 
  + **EXPO SQLite**

# Components

## Home Screen:

This component fetches data using an API on basis of facility-id and user-id and stores it in homeScreenFormInterface and displays the number of forms and also filter out the forms if clicked.

## HomeScreenListCard:

This component displays each form as a single card. Each card contains detailed information about the form, such as whether it pertains to a location, asset, or facility, whether it is a single-time fill-up or a multiple-time fill-up, and whether it has been downloaded or not. Clicking "Take Assessment" will navigate you to the next screen (Asset Config, Facility Config, Location Config).

## LocationConfig & AssetConfig:

This component displays a list of all locations and assets contained within a particular facility. By selecting the desired asset or location, you will be redirected to the respective form.

## FormFill:

This component will render dynamic forms based on the class of asset or location. It also helps you to seamlessly travel between form sections and finally submit the form or save the data locally.

### Dynamic form components:

* DatePickerOne: handles single date input
* DatePickerTwo: handles two date input (i.e. from and to)
* Dropdown: handles dynamic dropdown according to different forms id
* NumberInput: handles number, decimal input
* TextInput: handle single line text input
* TextAreaInput:handle multi line text input.

## Form Reducer:

Whenever data is fetched in any component, it is added to the Redux store. This approach helps in managing form fill data and ensures the data from the store is submitted efficiently.

I have stored sections\_data twice in the reducer: once for displaying the form and once for storing the form fill data. This structure simplifies the code and enhances maintainability.

Utilizes Redux *(useSelector, useDispatch)* for state management, enabling components to interact with the application's global state, potentially managing form synchronization statuses or queues.

## FormsServices:

This component contains all the API calls required in the forms module i.e.

* **Get\_assigned\_asset\_location:** Gives the assigned asset or locations connected to the Forms selected. It requires 4 object i.e. *facility\_id, company\_id, ft\_id, works\_for* (here *ft\_id* is the form template id which we already stored in *hsscData)*

**Payload:**

**let** data = JSON.stringify({

"company\_id": "61617648f622bab9c917ebdb",

"facility\_id": "61715bda8cd8a684333ef6bf",

"ft\_id": "664f182898dbe720d2c344c9",

"works\_for": "asset"

});

**let** config = {

method: 'post',

maxBodyLength: Infinity,

url: 'https://stageform.xempla.io/w-api-form-builder-backend/api/user/usage-config',

headers: {

'Content-Type': 'application/json'

},

data : data

};

**Response:**

**{**

**"status": "Success",**

**"data": {**

**"message": "Data Fetched Successfully.",**

**"data": [**

**{**

**"\_id": "6671673e52cfd16c4c0afa93",**

**"fusl\_id": "6671673e52cfd16c4c0afa92",**

**"ft\_id": "664f182898dbe720d2c344c9",**

**"ftl\_id": "664f184898dbe720d2c344cc",**

**"ftsl\_id": "664f184898dbe720d2c344cc",**

**"works\_for": "asset",**

**"company\_id": "61617648f622bab9c917ebdb",**

**"facility\_id": "61715bda8cd8a684333ef6bf",**

**"asset\_class\_id": "61715d588cd8a684333ef6f7",**

**"asset\_id": "625ef093f87cb07eee5b73f8",**

**"location\_id": null,**

**"sla": 0,**

**"start\_date": null,**

**"asset\_name": "806",**

**"location\_name": null,**

**"form\_already\_field": true,**

**"form\_fill\_permission": true**

**},**

**{**

**"\_id": "6671673e52cfd16c4c0afa95",**

**"fusl\_id": "6671673e52cfd16c4c0afa94",**

**"ft\_id": "664f182898dbe720d2c344c9",**

**"ftl\_id": "664f184898dbe720d2c344cc",**

**"ftsl\_id": "664f184898dbe720d2c344cc",**

**"works\_for": "asset",**

**"company\_id": "61617648f622bab9c917ebdb",**

**"facility\_id": "61715bda8cd8a684333ef6bf",**

**"asset\_class\_id": "61715d588cd8a684333ef6f7",**

**"asset\_id": "625ef093f87cb07eee5b73f9",**

**"location\_id": null,**

**"sla": 0,**

**"start\_date": null,**

**"asset\_name": "815",**

**"location\_name": null,**

**"form\_already\_field": true,**

**"form\_fill\_permission": true**

**}**

**],**

**"status": "success"**

**},**

**"message": "Succesfully fetched usage config"**

**}**

* **Get\_template:**  Fetches the selected form template based on ft\_id (here *ft\_id* is the form template id which we already stored in *hsscData)*

**CONFIG:**

**method: 'get',**

**maxBodyLength: Infinity,**

**url: 'https://stageform.xempla.io/w-api-form-builder-backend/api/superadmin/template/664f182898dbe720d2c344c9',**

**headers: { }**

**RESPONSE:**

**{**

**"status": "Success",**

**"data": [**

**{**

**"created\_by": "Debdut Sarkar",**

**"\_id": "664f182898dbe720d2c344c9",**

**"name": "Asset Information",**

**"usage": "app",**

**"config\_type": "Asset Configuration",**

**"frequency": 0,**

**"works\_for": "asset",**

**"submission": "multiple",**

**"is\_deleted": false,**

**"ftl\_id": "664f184898dbe720d2c344cc",**

**"ftsl\_id": "664f184898dbe720d2c344cd",**

**"created\_date": "2024-05-23T10:19:20.029000",**

**"sections": [**

**{**

**"s\_name": "Asset rated details",**

**"questions": [**

**{**

**"question": "Rated kW",**

**"answers": [**

**{**

**"field\_show\_name": "Enter rated power",**

**"field\_type": "number",**

**"data": null**

**}**

**]**

**},**

**{**

**"question": "Date of Purchase",**

**"answers": [**

**{**

**"field\_show\_name": "",**

**"field\_type": "date",**

**"data": "single date"**

**}**

**]**

**},**

**{**

**"question": "Criticality of the Asset",**

**"answers": [**

**{**

**"field\_show\_name": "",**

**"field\_type": "dropdown",**

**"data": "65e5bdcb9199d34d91be3fbc"**

**}**

**]**

**},**

**{**

**"question": "Next overhauling date",**

**"answers": [**

**{**

**"field\_show\_name": "Provide the date range",**

**"field\_type": "date",**

**"data": "two date"**

**}**

**]**

**},**

**{**

**"question": "Asset Id",**

**"answers": [**

**{**

**"field\_show\_name": "",**

**"field\_type": "text",**

**"data": null**

**}**

**]**

**},**

**{**

**"question": "Remarks",**

**"answers": [**

**{**

**"field\_show\_name": "",**

**"field\_type": "textarea",**

**"data": null**

**}**

**]**

**}**

**]**

**}**

**],**

**"allocations": {**

**"facility": [**

**"6523c74ff4a2c0a2990f9103",**

**"630363450b2d2a589a804b6e",**

**],**

**"company": []**

**},**

**"version": 2**

**}**

**],**

**"message": "Succesfully retrived template"**

**}**

* **Get\_form\_dropdown**: Fetches dropdown list based on the dropdown id .

**CONFIG:**

**method: 'get',**

**maxBodyLength: Infinity,**

**url: 'https://stageform.xempla.io/w-api-form-builder-backend/api/superadmin/dropdowns',**

**headers: { }**

* **Form\_submission**: posts the form filled data and returns if submission was successful.

**CONFIG:**

{

method: 'post',

maxBodyLength: Infinity,

url: 'https://stageform.xempla.io/w-api-form-builder-backend/api/user/submit-form',

headers: {

'Content-Type': 'application/json'

},

data : data

};

**PAYLOAD:**

{

"fus\_id": "6671673e52cfd16c4c0afa9b",

"fusl\_id": "6671673e52cfd16c4c0afa9a",

"created\_by": "644266d9a93caafe273cd17f",

"status": "submitted",

"created\_date": "2024-05-23T10:19:20.029000",

"last\_updated\_date": "2024-06-25T13:09:53.286Z",

"sections": [

{

"s\_name": "Asset rated details",

"questions": [

{

"question": "Rated kW",

"answers": [

{

"field\_show\_name": "Enter rated power",

"field\_type": "number",

"data": "25"

}

]

},

{

"question": "Date of Purchase",

"answers": [

{

"field\_show\_name": "",

"field\_type": "date",

"data": "13/6/2024"

}

]

},

{

"question": "Criticality of the Asset",

"answers": [

{

"field\_show\_name": "",

"field\_type": "dropdown",

"data": "function (a0) { [bytecode] }"

}

]

},

{

"question": "Next overhauling date",

"answers": [

{

"field\_show\_name": "Provide the date range",

"field\_type": "date",

"data": "20/6/2024"

}

]

},

{

"question": "Asset Id",

"answers": [

{

"field\_show\_name": "",

"field\_type": "text",

"data": "Ytig"

}

]

},

{

"question": "Remarks",

"answers": [

{

"field\_show\_name": "",

"field\_type": "textarea",

"data": "Gihvbh"

}

]

}

]

}

]

}

# 

# Offline Data Storage and Synchronization

## OfflineDataTaskForm

This TypeScript class manages an SQLite table (*offline\_data\_task\_form*) using expo-sqlite. It supports table creation, data insertion, fetching with conditions, and deletion. Operations are performed within SQLite transactions for data integrity and reliability

## FtAssetLocation

This TypeScript class manages a SQLite table (*FtAssetLocation*) for asset locations. It supports table creation, indexing, altering, data insertion (single and batch), fetching with conditions, and deletion. It uses SQLite transactions for safe operations with a provided database object (*dbObj*).

## OfflineDataTaskUpload

This TypeScript class manages an SQLite table (*offline\_data\_task\_upload*) using expo-sqlite. It facilitates table creation, data insertion, updating existing data, fetching with conditions, and deletion operations. Transactions ensure reliable data handling with the provided SQLite database object (*dbObj*).

## DownloadedTask

This class manages an SQLite table (*downloaded\_task*) via expo-sqlite, providing methods for table initialization, data insertion, update, fetch, and deletion, with index creation support for efficient data retrieval and management.

## PushOfflineTaskForm

This TypeScript class handles form submission and synchronization tasks, including deleting corresponding offline and downloaded tasks upon successful submission using services and database models.

## PushOfflineTask

This TypeScript class orchestrates the synchronization of various offline tasks stored in a SQLite database. It imports necessary modules like AsyncStorage for user ID retrieval, database models (**OfflineDataTask, OfflineDataTaskUpload, DownloadedTask)**, service utilities (**FormService**), and components (**PushOfflineTaskForm**) to handle form data in offline mode. The startProcess method iterates through unsynchronized tasks, invoking respective synchronization processes based on task type.

# Offline Form

## Database Models

1. **OfflineDataTaskForm:** Handles offline form data storage and operations, including insertion, retrieval, update, and deletion.
2. **OfflineDataTaskUpload:** Manages uploading of form data tasks, tracks synchronization status and timestamps.
3. **DownloadedTask**: Deals with form tasks that have been downloaded and stored locally.

## Database Utilities :

***openDatabaseCon:*** Provides database connection utilities (*openDatabaseCon*) to establish SQLite database connections where form data is stored.

## Network Connectivity (*NetInfo*):

Monitors network connectivity (*NetInfo*), crucial for determining when to synchronize form data tasks, ensuring synchronization when the device has internet access.

## Task Synchronization Component:

**PushOfflineTaskForm:**

1. Handles synchronization of form submissions.
2. Initiates synchronization (*startProcess*) of form data stored offline.
3. Uses *FormService* to submit form data (*form\_submission*).
4. Deletes synchronized form data from local storage (*deleteOfflineDataTaskForm*).
5. Deletes associated tasks *(deleteOfflineDataTask, deleteDownloadTask)* upon successful synchronization.

## Workflow

1. **Initialization**: PushOfflineTaskForm initializes with a database connection (*openDatabaseCon*).
2. **Data Fetching:** Uses OfflineDataTaskForm to fetch unsynchronized form data (*fetchNoSyncedData*).
3. **Synchronization:** Initiates synchronization using *FormService* to submit form data (*form\_submission*).
4. **Local Data Management**: After successful synchronization (*status === "success"*):
5. Deletes form data locally (*deleteOfflineDataTaskForm*).
6. Deletes associated tasks (*deleteOfflineDataTask, deleteDownloadTask*) if necessary.
7. **Network Monitoring:** Utilizes NetInfo to ensure synchronization occurs only when the device has internet access.

# Flow Chart

# Updating Expo to SDK 51 and Updating Dependencies

## Update Expo CLI and SDK

First, ensure you are using the latest version of the Expo CLI and SDK. You can update them by running the following commands:

npm install -g expo-cli

npm install expo@latest

## Update Expo Webpack Config

Next, update the Expo webpack-config package to the latest version:

npm install expo/webpack-config@latest

## Addressing Installation Errors

After updating the Expo CLI and SDK, any subsequent installations may result in errors. To avoid these errors, use the *`--legacy-peer-deps`* flag:

npm install --legacy-peer-deps

## Check and Update Dependencies

To ensure all dependencies are up to date, perform the following steps:

1. Install the npm-check-updates package globally:

npm install -g npm-check-updates

2. Run npm-check-updates to update package versions:

ncu -u

3. Install updated dependencies using the `--legacy-peer-deps` flag:

npm install --legacy-peer-deps

## Final Confirmation

After performing these steps, the Expo version should be updated to the latest version:

“expo” : "^51.0.21",

Created by

**Shreyas Pathak**

SDE Intern, Xempla