Low Level Design

Introduction

About NMIS App

National Management Information System (NMIS) Mobile, a groundbreaking application developed by Tech Consultancy Team using JavaScript and React Native. This state-of-the-art app offers a robust solution for data collection across various services.

The NMIS app is built upon JavaScript and harnesses the versatility of React Native, enabling seamless cross-platform development. By utilising the feature flag parameter, the app can be easily transformed into an Android APK, offering users a wide array of customisation options and features.

The feature flag parameter plays a pivotal role in configuring the app's behaviour. It empowers administrators to selectively enable or disable specific features based on their requirements, define various authentication methods to enhance security, customise themes to align with organisational branding, set server URLs for seamless integration, and even specify different types of questions within forms to capture precise information.

This exceptional level of configurability ensures that the NMIS app can be adapted as a generic module for diverse services. Regardless of the sector, whether it's healthcare, education, public services, or any other domain, the NMIS app offers a flexible and scalable solution for efficient and standardised data collection.

With its user-friendly interface, intuitive design, and advanced integration capabilities, the Mobile App for National Management Information System (NMIS) sets a new standard in data collection and management. By prioritising data accuracy, efficiency, and accessibility, this app empowers organisations to streamline their information-gathering processes and make informed decisions.

The Purpose of NMIS App

The purpose of the NMIS mobile app is to provide enumerators in the field with a user-friendly and efficient tool for data collection, complementing the existing web form in the web application. The app is specifically designed to streamline data collection processes, enhance accessibility, and overcome limitations associated with web forms.

The key purposes of the NMIS mobile app include:

1. **Simplified Data Collection**: The app aims to simplify the data collection process for enumerators by providing an intuitive and mobile-optimised user interface. It allows enumerators to collect data in a straightforward and efficient manner, reducing the complexities associated with web forms.
2. **Offline Data Collection**: A crucial purpose of the app is to enable offline data collection. Enumerators can capture data even in areas with limited or no internet connectivity. The app securely stores the collected data locally on the device, ensuring data integrity and allowing for seamless synchronisation when an internet connection is available.
3. **Improved Efficiency**: The NMIS mobile app enhances the efficiency of data collection in the field. It eliminates the need for enumerators to rely solely on web forms accessed through browsers, which may involve additional login procedures or restricted access. The app provides a dedicated platform for enumerators, ensuring a smoother and faster data collection experience.
4. **Streamlined Field Operations**: By leveraging mobile-specific features, such as GPS integration for location tracking, the app streamlines field operations. Enumerators can efficiently capture geographical data alongside other relevant information, improving the accuracy and context of collected data.
5. **Enhanced User Experience**: The NMIS mobile app focuses on delivering an optimal user experience for enumerators. It leverages mobile devices' capabilities, such as touch gestures and responsive layouts, to provide a seamless and intuitive interface. This purposeful design promotes ease of use and reduces the learning curve for enumerators in the field.

In summary, the NMIS mobile app's purpose is to facilitate efficient and user-friendly data collection for enumerators in the field. It enhances accessibility, enables offline capabilities, and improves overall efficiency in capturing accurate and timely data.

Functional Overview

The NMIS mobile app offers a comprehensive range of functionalities tailored to support enumerators in their data collection tasks. This section provides a functional overview of the key features and capabilities of the app. With support for various question types, optional dependency logic, validation rules, offline data collection, seamless synchronisation, JSON form format, and a user-friendly interface, the app empowers enumerators to collect data accurately, efficiently, and conveniently in the field.

Data Collection Forms

- Enumerators can access and complete customised data collection forms using the app. These forms are designed to capture various types of information, including geolocation, text, number, options, multiple options, cascade, photo/video, and date.
- The app provides an intuitive interface for enumerators to input data based on the question type. Enumerators can provide accurate and precise information during the data collection process.

Optional Dependency Logic

- The app supports optional dependency logic, enabling dynamic display and requirement of questions based on responses to previous questions. This functionality enhances the data collection experience by adapting the form based on specific conditions or dependencies.
- Enumerators can seamlessly navigate through the form, with the app intelligently adjusting the display of questions based on the predefined dependencies. This ensures a smooth and efficient data collection process.

Validation Rules

- The app incorporates validation rules to ensure data accuracy and integrity. Real-time validation feedback is provided to enumerators, such as error messages or prompts for missing or invalid data.
- Enumerators receive immediate feedback, ensuring that data entered meets the specified criteria and reducing the likelihood of errors. This promotes data quality and reliability.

Offline Data Collection

- Enumerators can collect data even in offline environments, leveraging the app's offline functionality.
- The app securely stores the collected data on the device, allowing enumerators to continue their work without an internet connection.
- Once an internet connection is available, the app automatically synchronises the collected data with the central server.

Seamless Synchronisation

- The app seamlessly synchronises collected data with the central server once an internet connection is available. This ensures that no data is lost during the process.
- Enumerators can trust that their data is securely transmitted and integrated into the NMIS system. Synchronisation happens in the background, allowing enumerators to focus on their data collection tasks without worrying about manual data transfer.

JSON Form Format

- The app receives the data collection forms in JSON format.
- The JSON structure follows the format used in the **Akvo React Form**, which used by the **Burkina Faso PDHA**, **WAI-SDG**, **ISCO** or **Kenya RTMIS** Web Page.
- Enumerators can easily navigate and interact with the forms within the app, as the JSON structure provides a familiar format. This simplifies the transition from the web form to the mobile app and ensures a consistent data collection experience.

User-Friendly Interface

- The app offers a user-friendly interface designed specifically for mobile devices. It prioritises ease of use and intuitive navigation, optimised for the smaller screens and touch gestures of smartphones and tablets.
- Enumerators can efficiently navigate through the forms, input data, and access various features. The intuitive interface reduces training requirements and contributes to a seamless and pleasant data collection experience.

The functional overview of the NMIS mobile app highlights its robust capabilities in supporting enumerators during data collection activities. With support for various question types, optional dependency logic, validation rules, and the ability to handle JSON forms, the app provides a powerful tool that enhances the efficiency, accuracy, and convenience of field data collection processes.

Design Considerations

When designing the NMIS mobile app, several key considerations were taken into account to ensure a seamless and user-friendly experience for enumerators in the field. The following design principles guided the development process:

1. **Mobile-Optimised Interface:** The app's interface is specifically designed for mobile devices, prioritising ease of use and intuitive navigation. It takes into account the limited screen space and touch interactions of smartphones and tablets, ensuring a seamless user experience for enumerators.
2. **Offline Functionality:** Recognising the potential lack of internet connectivity in remote areas, the app includes robust offline capabilities. Enumerators can collect data without an active internet connection, and the app securely stores the data locally on the device. Once a connection is available, the app automatically synchronises the collected data with the central server.
3. **Responsive and Adaptive Design:** The app features a responsive and adaptive design, adjusting its layout and functionality based on the screen size and orientation of different devices. This ensures optimal presentation and usability, regardless of the device used by enumerators.
4. **Clear and Intuitive Form Design:** The design of data collection forms prioritises clarity and simplicity. Questions are presented in a logical and easy-to-understand manner, reducing cognitive load for enumerators and enabling efficient data entry.
5. **Visual Feedback and Validation:** The app provides visual feedback and validation to guide enumerators during data collection. Real-time validation checks highlight errors or missing information, ensuring the accuracy and completeness of collected data. Clear visual cues and informative prompts assist enumerators in providing accurate and valid responses.
6. **Streamlined Navigation:** The app features streamlined navigation, allowing enumerators to move through the data collection process effortlessly. Intuitive icons, buttons, and gestures provide smooth transitions between questions and sections,

reducing friction and improving efficiency.

7. **Accessibility Considerations:** The app adheres to accessibility guidelines to ensure inclusivity for all users. It incorporates features such as adjustable text size, colour contrast for readability, and support for assistive technologies. By prioritising accessibility, the app promotes equal access and usability for individuals with diverse needs.
8. **Security and Data Privacy:** Data security and privacy are paramount considerations in the app's design. The app incorporates robust security measures to protect collected data, including encryption during transmission and adherence to industry best practices. Enumerators can trust that the data they collect is handled securely and confidentially.

UI Design Overview

> Design can be found in: **NMIS - Mobile App Design V2**

The design in the provided Figma link represents the user journey and interface design of the mobile module for the NMIS app based on the discussions held during the planning phase. It showcases the layout, interaction, and visual elements to ensure a seamless and user-friendly experience for enumerators during the data collection process. Here's an overview of what the design entails:

1. **User Journey:** The Figma design depicts the step-by-step user journey, outlining the different screens and actions involved in the data collection process. It provides a visual representation of how users will navigate through the app, from entering the assignment code to selecting a survey, answering questions, and submitting the form.
2. **Interface Design:** The design showcases the visual elements, including the layout, color scheme, typography, and graphical representations used throughout the app. It offers a glimpse into the overall look and feel, ensuring a consistent and intuitive user experience.
3. **Screen Layout:** Each screen in the user journey is represented, illustrating the arrangement of elements, such as buttons, input fields, navigation components, and form sections. The layout is designed to optimise user interaction, ensuring ease of use and clear information presentation.
4. **Interactive Components:** The Figma design may include interactive components, such as clickable buttons, dropdown menus, and modals. These elements demonstrate the expected behavior and user interactions, providing a sense of how the app will respond to user inputs.
5. **Feedback and Iteration:** The design is a result of several planning discussions, indicating that it has gone through iterations and feedback from stakeholders and the development team. It reflects the collaborative effort to align on the app's visual design and user flow.

Software Architecture

Build Configuration Parameters

Server URL

During the development and build phase of the NMIS mobile app, the server URL parameter is used to specify the default endpoint or API URL of the NMIS server. This URL is typically determined by the development team based on the specific deployment environment (e.g., development, staging, production). By providing the server URL as a build parameter, the app is pre-configured with the appropriate endpoint to communicate with the NMIS server. This ensures that the built app connects to the correct server once it is installed and launched on users' devices.

The server URL parameter in the build process ensures that the app is ready to establish a connection with the designated NMIS server by default. This allows enumerators or users to seamlessly interact with the server without having to manually input or configure the server URL during the initial setup of the app.

Debug Mode

During the build process of the NMIS mobile app, developers can include a "Debug Mode" build parameter to control the activation of a dedicated mode designed to assist in debugging and troubleshooting issues during app development.

1. **Debugging Capabilities:** When the app is in Debug Mode, developers gain access to additional tools and capabilities for troubleshooting and identifying potential issues. This includes features such as enhanced logging, detailed error messages, runtime inspection, and the ability to set breakpoints for stepping through the code.
2. **Enhanced Logging:** Debug Mode often includes more extensive logging capabilities, providing developers with a detailed view of the app's internal workings. This helps in tracking the execution flow, identifying errors, and understanding the sequence of events leading up to specific issues.
3. **Diagnostic Information:** Debug Mode allows developers to collect and analyze diagnostic information, such as variable values, stack traces, and system-level details. This information is instrumental in identifying the root causes of errors or unexpected behavior, facilitating efficient bug fixing.
4. **Real-time Feedback:** Debug Mode enables real-time feedback during app development. Developers can observe the app's behavior, identify potential issues, and make immediate adjustments or corrections, improving the efficiency of the debugging process.
5. **Controlled Environment:** Debug Mode provides a controlled environment for developers to test and troubleshoot app features and functionality. It allows them to isolate and focus on specific parts of the app, enabling effective debugging and minimising interference from external factors.

6. **Development Workflow:** Debug Mode supports an iterative development workflow. Developers can make changes, test them, debug issues, and refine their code until the desired functionality is achieved. This iterative approach helps in improving the overall quality and reliability of the app.

> More about debugging: https://reactnative.dev/docs/debugging

App Version Control

This parameter ensures that the app is running on the correct version by specifying the version number or code. It helps in maintaining compatibility and consistency between the app and the server, ensuring seamless functionality and data syncing.

App Version Control is essential for maintaining a structured and organised approach to app development, deployment, and updates. It involves assigning unique version numbers or codes to each release of the app, allowing users to identify, track, and manage different versions effectively.

Key aspects of App Version Control include:

1. **Version Numbering:** App developers assign version numbers to each release of the app. Version numbers typically follow a structured format, such as X.Y.Z, where X represents a major version, Y denotes a minor version, and Z signifies a patch version. This numbering scheme provides a systematic way to indicate the significance of changes and updates.
2. **Release Management:** App Version Control helps manage the release process by distinguishing between major updates, minor feature enhancements, and bug fixes. Each version represents a distinct set of changes and serves as a reference point for development, testing, and deployment.
3. **Compatibility and Upgrades:** Version control allows users to determine whether their installed app version is compatible with the server or requires an update. It helps ensure that users are running the correct version that aligns with the server's capabilities and data structures.
4. **Bug Tracking and Issue Resolution:** Version control plays a crucial role in bug tracking and issue resolution. By referring to specific app versions, developers can identify when bugs were introduced, track fixes, and provide relevant updates to users. It facilitates efficient communication and collaboration among development teams, testers, and users.
5. **User Support and Communication:** Version control helps streamline user support by enabling precise identification of the app version in use. Users can easily communicate their installed version when reporting issues, allowing support teams to provide targeted assistance and troubleshoot problems effectively.
6. **Feature Rollouts and A/B Testing:** App Version Control facilitates controlled feature rollouts and A/B testing. Developers can release new features gradually, targeting specific app versions, user groups, or geographic regions. This approach allows for better evaluation of feature performance and user feedback before widespread deployment.

Build Parameters

Below are the build parameter formats:

```
{
  "authenticationType": ["code_assignment", "username", "password"],
  "serverURL": "https://api.example.com/nmis",
  "debugMode": false,
  "dataSyncInterval": 300,
  "errorHandling": true,
  "loggingLevel": "verbose",
  "appVersion": "1.2.0",
  "lang": "en",
}
```

- **authenticationType**: an array that includes multiple options: **code_assignment**, **username**, and **password**. This configuration allows users to choose one or more authentication methods based on their requirements.This reflects the optional nature of the authentication types and allows for flexibility in selecting authentication methods, including the use of a code assignment before entering a username (with or without a password). It also considers the scenario where multiple enumerators can utilise a single app, accommodating projects with limited budgets for purchasing individual devices.
- **serverURL**: Specifies the endpoint or API URL of the NMIS server as **https://nmis.akvotest.org/api**.
- **debugMode**: Indicates that debug mode is disabled with a value of **false**.
- **dataSyncInterval**: Defines the default data sync interval as 300 seconds (5 minutes).
- **errorHandling**: Specifies that error handling is enabled with a value of true.
- **loggingLevel**: Sets the logging level to **verbose**.
- **appVersion**: Represents the version number of the app as **1.2.0**.
- **lang**: Defines the default language for the UI components and Forms.

App Configuration Settings

The app configuration settings parameter provides a range of customisable options which already defined in Build Parameters. It allows users to tailor the app's behavior and appearance according to their specific preferences.

1. **Data Sync Options:** Data syncing options allow users to configure how and when the app synchronise data with the server. Users can choose to enable or disable automatic syncing, specify Wi-Fi-only syncing, or define the sync interval, giving them control over the data transfer process.
2. **Data Sync Interval:** The Data Sync Interval configuration setting allows users or enumerators to define the frequency at which the NMIS mobile app automatically syncs data with the server. This setting can be adjusted within the app's user interface or settings menu, providing flexibility for users to tailor the sync interval according to their specific needs and preferences.

3. **Language Selection:** Language selection settings enable users to choose their preferred language for the app's interface. This feature ensures that the app is accessible to users who are more comfortable with languages other than the default language.

State Management

State Management Library

We utilise the pullstate library for managing the state of the NMIS mobile app. pullstate is a lightweight state management library for React applications that provides a simple yet powerful solution for managing application state. With pullstate, we can easily define and update our application's state using custom stores and actions. The library follows a "pull-based" approach, where components explicitly "pull" the data they need from the state, ensuring efficient rendering and minimising unnecessary re-renders.

By using pullstate for state management in the NMIS mobile app, we can achieve the following benefits:

1. Lightweight and Minimalistic: pullstate is a small and focused library, which means it doesn't introduce unnecessary complexity or overhead to our application. It provides a clean and concise API for managing state.
2. Predictable and Immutable State: pullstate encourages immutability by default. This helps maintain the predictability of the state and simplifies debugging by preventing accidental mutations.
3. Efficient Rendering: The "pull-based" approach of pullstate ensures that only the components that directly depend on a specific piece of state will re-render when that state changes. This avoids unnecessary re-renders in the application and improves performance.
4. Easy to Use and Understand: pullstate follows a straightforward API design, making it easy to grasp and integrate into our existing React components. It promotes a declarative and functional programming style, enhancing the readability and maintainability of our codebase.

By leveraging the pullstate library for state management, we can efficiently manage and update the application's state, ensuring a smooth and responsive user experience in the NMIS mobile app.

> Documentation: https://lostpebble.github.io/pullstate/

Example use of Pullstate:

```
import { Store } from "pullstate";

export const UIStore = new Store({
  isDarkMode: true,
```

```
});
```

```
import * as React from "react";
import { UIStore } from "./UIStore";

export const App = () => {
  const isDarkMode = UIStore.useState(s => s.isDarkMode);

  return (
    <div
      style={{
        background: isDarkMode ? "black" : "white",
        color: isDarkMode ? "white" : "black",
      }}>
      <h1>Hello Pullstate</h1>
      <button
        onClick={() =>
          UIStore.update(s => {
            s.isDarkMode = !isDarkMode;
          })
        }>
        Toggle Dark Mode
      </button>
    </div>
  );
};
```

App Configuration State

```
const appConfiguration = {
  authentication: {
    authenticationType: ["assesment", "username_password"],
    serverURL: "https://api.example.com/nmis",
    authenticationCode: "",
  },
  buildParams: {
    debugMode: false,
    dataSyncInterval: 300,
    errorHandling: true,
    loggingLevel: "verbose",
```

```
    appVersion: "1.2.0"
  },
  formConfiguration: {
    form: {},
    questionGroups: [],
    questions: []
  },
  userConfiguration: {
    username: "",
    password: "",
    preferences: {}
  },
  currentPage: "Home"
};
```

DRAFT

Database

Database Library

We use the database to store something that stays in the application without any changes, for example: forms, submissions and personal configurations. To achieve this, we need a library that is natively easy to use and persisted across restarts. Expo-sqlite gives the app access to a database that can be queried through a WebSQL-like API. See the following example:

```
import { Platform } from 'react-native';
import * as SQLite from 'expo-sqlite';

/*Init DB*/

const openDatabase = () => {
  if (Platform.OS === 'web') {
    return {
      transaction: () => {
        return {
          executeSql: () => {},
        };
      },
    };
  }
  const db = SQLite.openDatabase('db.db');
```

```
  return db;
};

const db = openDatabase();

db.transaction((tx) => {
  tx.executeSql('\
    create table if not exists examples(\
        id integer primary key not null,\
        name text,\
        example_float real,\
        example_json text\
    );'
  );
});

/*Query*/

const addData ='\
  insert into examples\
  (name, example_float, example_json) \
  values (?, ?, ?)';

const getAllData = 'select * from examples';

db.transaction((tx) => {
    tx.executeSql(addData, [text, number, JSON.stringify(['Devin', 'Dan', 'Dominic'])]);
    tx.executeSql(getAllData, [], (_, { rows }) =>
      console.log('examples: ', JSON.stringify(rows)),
    );
  },
  null,
  forceUpdate,
);
```

> Documentation: https://docs.expo.dev/versions/latest/sdk/sqlite/

Database Schema

There are 4 tables that need to be created to store activities that remain even if the application is restarted. These tables are:

1. Users

Table name: **users**

Column Name	Type	Example
id	INTEGER (PRIMARY KEY)	1
active	TINYINT	1 (default: 0)
name	INTEGER	1
password	TEXT	crypto

2. Config

Table name: **config**

Column Name	Type	Example
appVersion	INTEGER	1
authenticationCode	TEXT	crypto
serverURL	TEXT	'https://rtmis.akvo.org'
syncInterval	REAL	3 (in Minutes)
syncWifiOnly	TINYINT	1
lang	VARCHAR(255)	"en"

3. Form List

Table name: **forms**

Column Name	Type	Example
id	INTEGER (PRIMARY KEY)	1
formId	INTEGER	453743523
version	VARCHAR(255)	"1.0.1"
latest	TINYINT	1
name	VARCHAR(255)	'Household'
json	TEXT	See: Example JSON Form

Column Name	Type	Example
createdAt	DATETIME	`new Date().toISOString()`

4. Form Submission / Datapoints

Table name: **datapoints**

Column Name	Type	Example
id	INTEGER (PRIMARY KEY)	1
form	INTEGER	1 (represent **id** in **forms** table, NOT formId)
user	INTEGER	1
name	VARCHAR(255)	'John - St. Maria School - 0816735922'
submitted	TINYINT	1
duration	REAL	45.5 (in Minutes)
createdAt	DATETIME	`new Date().toISOString()`
submittedAt	DATETIME	`new Date().toISOString()`
syncedAt	DATETIME	`new Date().toISOString()`
json	TEXT	`'{"question_id": "value"}'`

Navigating Between Components

The provided code snippet showcases the implementation of a navigation structure using React Navigation for a React Native app. **NavigationContainer** (imported from `@react-navigation/native`) and acts as the root component navigation, while the **createNativeStackNavigator** (imported from `@react-navigation/native-stack`) function is utilized to create a stack navigator. Each screen is defined using the **Stack.Screen** component, and the **screenOptions** prop is utilized to hide the header for all screens. The **Navigation** component wraps the navigation elements, facilitating the seamless navigation between screens using the provided navigation methods.

> All the corresponding Page components should be placed under **./src/pages**

```
import { HomePage, FormActionPage, FormDataPage } from '../pages';

const Stack = createNativeStackNavigator();

const RootNavigator = () => {
  return (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
      <Stack.Screen name="Home" component={HomePage} />
      <Stack.Screen name="AppSetting" component={AppSettingPage} />
      ...
      ...
      ...
      <Stack.Screen name="FormData" component={FormDataPage} />
      <Stack.Screen name="FormAction" component={FormActionPage} />
    </Stack.Navigator>
  );
};

const Navigation = (props) => {
  return (
    <NavigationContainer {...props}>
      <RootNavigator />
    </NavigationContainer>
  );
};
```

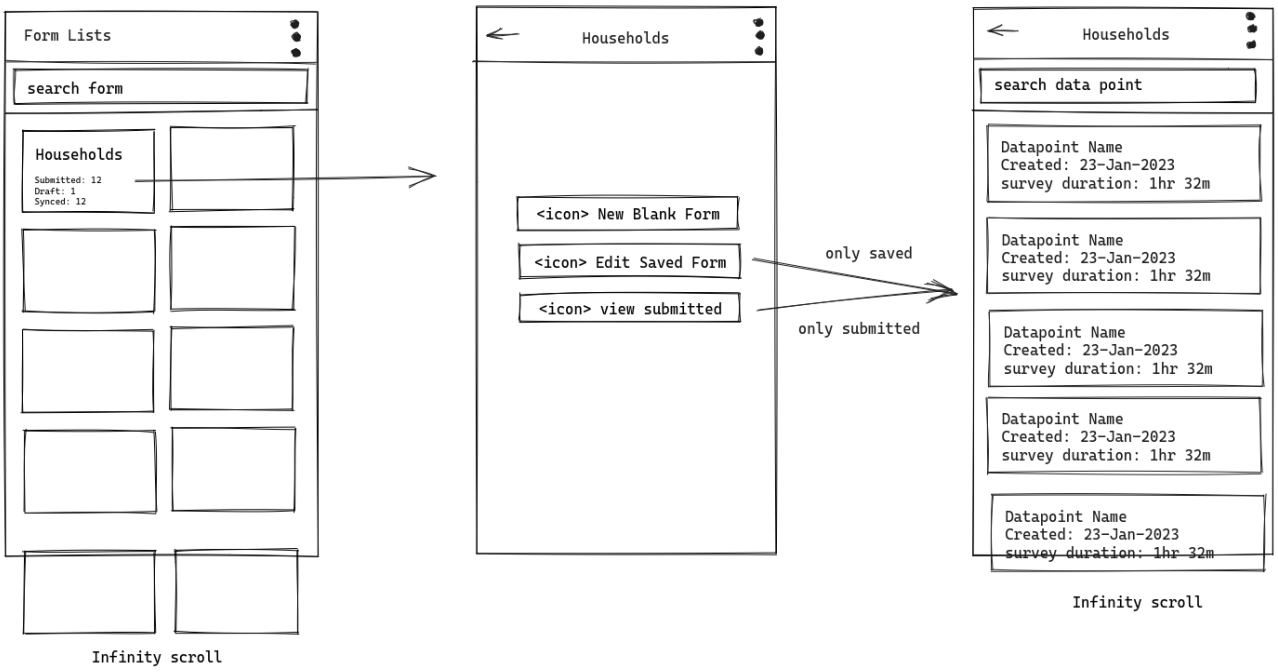
To navigate between screens: `navigation.navigate('Home')`

> Documentation: https://reactnavigation.org/docs/getting-started

Base Layout Components

We will utilise the pullstate library for managing the state of the NMIS mobile app. The provided page layouts serves as a configuration that allows for dynamic rendering of page components based on specific values.

List of Cards Layout




```
{
    title: "Household - Submitted",
    columns: 1 | 2,
    search: {
        placeholder: "Search Datapoints" | "Search Form",
        show: true | false,
    },
    action: () => {
        navigation.navigate('Other Page', {
          id: '23445'
        })
    },
    data: [{
        id: 23445,
        name: "Datapoint Name",
        subtitles: [
            "created: 23 Jan 2023",
            "survey duration: 1hr 32m",
        ]
    }]
}
```

1. **Title:** Displayed at the top of the page to provide a descriptive heading.
2. **Columns:** Determines the layout of the data section, with options for 1 or 2 columns.
3. **Search:**

- **Placeholder:** Specifies the text to display inside the search input field, such as "Search Datapoints" or "Search Form".
- **Show:** Determines whether the search input field should be displayed or hidden.

4. **Action:** Defines a navigation action to be triggered when a specific action is performed, such as clicking a button. The action can navigate to another page, passing relevant parameters, such as the ID of '23445' to the 'Other Page'.
5. **Data:** Represents an array of data objects to be displayed on the page.
   - **id:** Unique identifier for each data object.
   - **name:** Specifies the name of the datapoint.
   - **subtitles:** An array of subtitles or additional information related to the datapoint, such as the creation date and survey duration.

> The Action navigator uses the native APIs: **Fragment** on Android so that navigation built with **createNativeStackNavigator** will behave the same and have the same performance characteristics as apps built natively on top of those APIs.

Form Layout

The NMIS mobile app utilises JSON as the format for representing form structures and configurations. To ensure consistency and compatibility with existing form APIs, the app follows the format established by **Akvo React Form**. This allows for seamless integration with other form-related services and promotes interoperability across different platforms.

By adhering to the Akvo React Form structure, the NMIS app can leverage the functionalities provided by libraries like **Formik,** which is commonly used for form management in React applications. Formik simplifies the process of handling form inputs, validation, and submission by providing a standardised approach.

To map the JSON form structure to a format compatible with Formik, the NMIS app employs a mapping mechanism that transforms the JSON representation into a structure that Formik can readily work with. This mapping process ensures that the form data can be seamlessly handled within the app, facilitating smooth data entry, validation, and submission processes.

> To ensure consistency and compatibility with the Akvo React Form structure, the NMIS mobile app utilises a similar JSON format for representing form structures and configurations. Here is an **example JSON format** that can be used as a reference.

Akvo React Form	
Repository	https://github.com/akvo/akvo-react-form
Website	https://akvo.github.io/akvo-react-form/

Documentation	https://github.com/akvo/akvo-react-form/blob/main/README.md
Example JSON Format	https://raw.githubusercontent.com/akvo/akvo-react-form/main/example/src/example.json
Formik	
Repository	https://github.com/jaredpalmer/formik
Website	https://formik.org
Documentation	https://formik.org/docs

Example JSON Form

```
{
  "id": 519630048,
  "form": "Citizen",
  "version": "1.0.0",
  "languages": ["en", "id"],
  "defaultLanguage": "en",
  "translations": [
    {
      "name": "Penduduk",
      "language": "id"
    }
  ],
  "question_group": [
    {
      "name": "Registration",
      "order": 1,
      "translations": [
        {
          "name": "Registrasi",
          "language": "id"
        }
      ],
      "question": [
        {
          "id": 1,
          "name": "Weight",
          "order": 1,
```

```
          "type": "number",
          "required": true,
          "tooltip": {
            "text": "Information Text"
          },
          "rule": {
            "min": 5,
            "max": 10
          },
          "meta": true,
          "dependency": [
            {
              "id": 9,
              "options": ["Yes"]
            },
            {
              "id": 10,
              "min": 8
            }
          ],
          "translations": [
            {
              "name": "Berat Badan",
              "language": "id"
            }
          ]
        }
      ]
    }
  ]
}
```

- **name**: Specifies the name of the form, which in this case is "Health Center Facilities".
- **languages**: Represents an array of languages supported by the form, including "en" (English) and "id" (Indonesian).
- **defaultLanguage**: Indicates the default language for the form, which is set to "en" (English).
- **translations**: Contains an array of translations for the form's name. In this example, it includes a translation for the name in Indonesian as "Fasilitas Kesehatan".
- **question_group**: Represents an array of question groups within the form.
  - **name**: Specifies the name of the question group, which is "Registration".

- **order**: Indicates the order or position of the question group within the form.
- **translations**: Contains an array of translations for the question group's name. In this example, it includes a translation for the name in Indonesian as "Registrasi".
- **question**: Represents an array of questions within the question group.
  - **id**: Specifies the unique identifier for the question.
  - **name**: Represents the name of the question, which in this case is "Weight".
  - **order**: Indicates the order or position of the question within the question group.
  - **type**: Specifies the type of question, which is "number" in this example.
  - **required**: Indicates whether the question is required or not.
  - **tooltip**: Contains information about a tooltip associated with the question, such as additional instructions or details.
    - **text**: Specifies the text for the tooltip associated with the question.
  - **rule**: Represents a validation rule for the question, such as minimum and maximum values.
    - **min**: Specifies the minimum value allowed for the question.
    - **max**: Specifies the maximum value allowed for the question.
  - **meta**: Indicates whether the question collects metadata information for data point name.
  - **dependency**: Represents an array of dependencies for the question, indicating its visibility or validation rules based on the values of other questions.
    - **id**: Specifies the ID of the question that the current question depends on.
    - **options**: Specifies the specific options that need to be selected in the dependent question for the current question to be visible or valid.
    - **min**: Specifies the minimum value that needs to be entered in the dependent question for the current question to be visible or valid.
  - **translations**: Contains an array of translations for the question's name. In this example, it includes a translation for the name in Indonesian as "Berat Badan".

Form Components

Components Structure

The below tree will show the hierarchy of Form Components structure:

```
/src/form/
├── components
│   ├── index.js
│   ├── QuestionField.js
│   ├── QuestionGroup.js
│   └── Question.js
├── fields
│   ├── index.js
│   ├── __test__
```

```
│   ├── TypeDate.js
│   ├── TypeImage.js
│   ├── TypeInput.js
│   ├── TypeMultipleOption.js
│   ├── TypeNumber.js
│   ├── TypeOption.js
│   └── TypeText.js
├── lib
│   └── index.js
├── support
│   ├── FieldGroupHeader.js
│   ├── FieldLabel.js
│   ├── FormNavigation.js
│   ├── index.js
│   └── __test__
├── FormContainer.js
├── index.js
├── initial-values.json
├── example-form.json
├── styles.js
└── __test__
```

The form components are located in the ./src/form/ directory, which is further divided into four subfolders: components, fields, lib, and support. The **main file** that serves as the **entry point** for the form component is **FormContainer.js**.

1. Components Folder:
   The components folder contains three main component files: QuestionGroup.js, Question.js, and QuestionField.js. Each file is responsible for rendering question groups, questions, and fields based on the JSON form.
2. Fields Folder:
   The fields folder contains files that manage the input for each supported question type. These files, such as TypeDate.js, TypeImage.js, TypeInput.js, etc., control the behavior of the QuestionField component.
3. Lib Folder:
   The lib folder houses reusable functions that are utilized by other components within the form.
4. Support Folder:
   The support folder consists of reusable components that are used by other form components. It includes components like FieldGroupHeader.js, FieldLabel.js, and FormNavigation.js.Supported Field Type

Supported Field Type

Type	Description
input	Normal/short text input
number	Number input
text	Long text input
date	Date picker
option	Single select option (radio/dropdown)
multiple_option	Multiple select options (checkbox/dropdown)
image	Image picker (from gallery/use camera)

Form Parameters

Parameter	Description
forms	The forms parameter defines the structure and behavior of the form based on an example JSON Form. It serves as a blueprint for rendering the form and its fields.
initialValues	The initialValues parameter sets the initial values for each field when the form is first loaded. These values provide pre-filled data or placeholders, enhancing the user experience. By default is an empty object `{}`.

Form Events

Event	Description

onSubmit	onSubmit event returns data point name, geo location, and a set of values representing the answers provided for each question field. These values are returned as an object, where each question's ID serves as the object key, and its corresponding answer value is stored as the value. For example, the object may look like this: `{ 'name': 'Datapoint name', 'geo': 'lat|long', 'answers': [ { '1': 'John', '2': 1992-01-01T00:00:00.000Z, '3': '31', '4': ['Male'], '5': ['Bachelor'], '6': ['Traveling'], '7': ['Rendang'], '9': '8.9' } ] }`

Form Validation

The Form component relies on the powerful features provided by Formik and Yup to handle form validation efficiently. These libraries work together to validate user input and ensure data integrity.

Within the Form component, the validation schema is constructed at the question level. This means that each question in the form has its own validation rules and constraints defined by the generated schema.

To handle the generation of the validation schema, a dedicated function called generateValidationSchemaFieldLevel is implemented. This function is located in the ./src/form/lib/index.js file.

By structuring the validation schema generation in this way, the Form component achieves a high level of flexibility and maintainability. Each question can have its own unique validation rules, allowing for granular control over the form's validation process.

generateValidationSchemaFieldLevel function:

```
export const generateValidationSchemaFieldLevel = (currentValue, field) => {
 const { name, type, required, rule } = field;
 let yupType;
 switch (type) {
   case 'number':
     // number rules
     const isEmpyCurrentValue = currentValue === '';
     yupType = isEmpyCurrentValue ? Yup.string() : Yup.number();
     if (!isEmpyCurrentValue && rule?.min) {
       yupType = yupType.min(rule.min);
     }
     if (!isEmpyCurrentValue && rule?.max) {
       yupType = yupType.max(rule.max);
     }
     if (!isEmpyCurrentValue && !rule?.allowDecimal) {
       // by default decimal is allowed
       yupType = yupType.integer();
     }
     break;
   case 'date':
     yupType = Yup.date();
     break;
   case 'option':
     yupType = Yup.array();
     break;
   case 'multiple_option':
     yupType = Yup.array();
     break;
   default:
     yupType = Yup.string();
     break;
 }
 if (required) {
   const requiredError = `${name} is required.`;
   yupType = yupType.required(requiredError);
 }
 try {
   yupType.validateSync(currentValue);
 } catch (error) {
   return error.message;
```

```
  }
};
```

The generateValidationSchemaFieldLevel function takes two parameters: **currentValue** and **field**. currentValue represents the current value of the field being validated, while the field parameter holds the JSON structure of the question field, including details like id, name, type, required, and rule.

This function performs validation based on the field's type. It uses a switch statement to handle different field types and applies specific validation rules accordingly. It also checks for the required parameter to enforce mandatory field validation.

By using this design, the software ensures consistent and reliable validation for various field types. Each field's validation rules are applied based on its specific type. The function also ensures that required fields are properly validated, maintaining data integrity and adhering to user-defined validation criteria.

Example of Usage

The below code shows minimal requirements to use the form component:

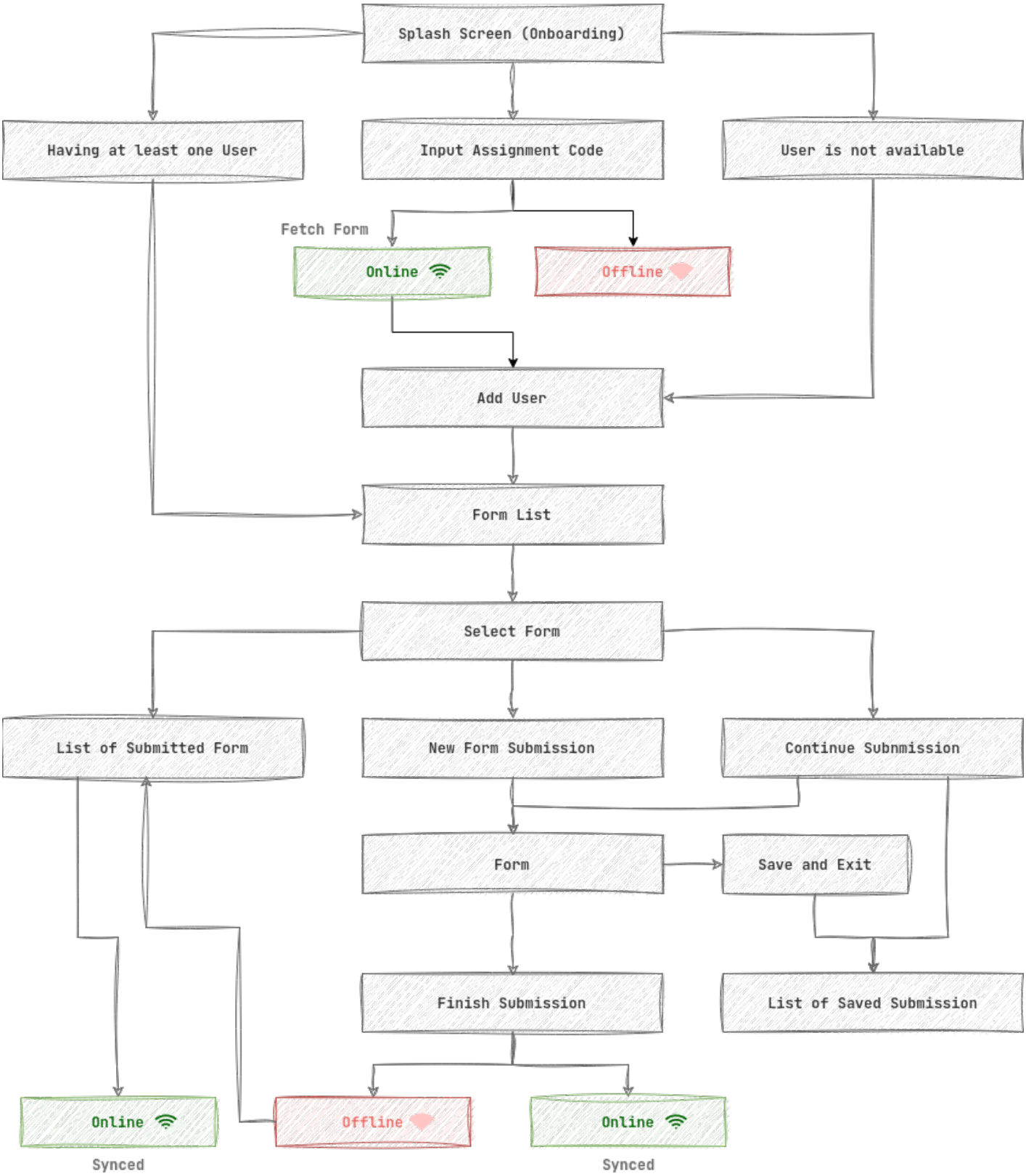
```
import React from 'react';
import { FormContainer } from '../form';
import * as formDefinition from '../form/example-form.json';

const FormPage = () => {
  return (
    <FormContainer forms={formDefinition} initialValues={{}} />
  );
};

export default FormPage;
```

Data Collection Workflow

Below explanation illustrates the step-by-step workflow of the NMIS mobile app, ensuring a seamless and user-friendly experience for data collection, user management, form selection, survey duration tracking, form navigation, and submission handling.



Ask Permissions

At the beginning of the app, the user is prompted to grant permissions for accessing the camera, location, and files on their device. This ensures that the app can utilise these features for data collection and other functionalities.

Input Assignment Code Page

The **Input Assignment Code** page is a crucial step in the NMIS mobile app's data collection workflow. This page serves as the entry point for enumerators to access the forms assigned to them. Here's a more detailed explanation of the Input Assignment Code page:

1. **Purpose**: The Input Assignment Code page aims to validate the enumerator's assignment by requiring them to enter a specific code assigned to them by the administrator. This code acts as a unique identifier for the enumerator's assigned tasks and helps filter the relevant forms for data collection.
2. **Internet Connection Requirement**: To ensure accurate assignment code verification, an active internet connection is necessary for the Input Assignment Code page. The app checks for connectivity before presenting the page. If no internet connection is available, the app displays an "offline page" informing the enumerator about the need for an internet connection to proceed.
3. **User Interaction**: Enumerators are prompted to enter the assignment code through an input field or a dedicated form on the Input Assignment Code page. The app may provide a clear explanation or instructions to assist enumerators in finding and entering the correct code.
4. **Assignment Code Validation**: Once the enumerator submits the assignment code, the app verifies its authenticity and relevance. The code is validated against the server's records to ensure that it corresponds to the enumerator's assigned tasks. If the provided assignment code is incorrect or invalid, the app displays a popup notification to inform the enumerator of the error.
5. **Filtered Forms Retrieval**: Upon successful validation of the assignment code, the app retrieves the relevant forms assigned to the enumerator from the server. The forms are filtered based on the administrative cascade sources associated with the assignment code. This ensures that enumerators only have access to the forms that are specifically assigned to them, streamlining the data collection process.

Add User Page

Since there is no user information available initially, the user is directed to the "Add User" page. At least one user needs to be added to proceed. Users can be added with or without a password. This page is displayed as the first step once a session is available. This page allows enumerators to create user profiles within the app. Here's a more detailed explanation of the Add User page:

1. **Purpose**: The Add User page serves the purpose of enabling enumerators to create user profiles within the NMIS mobile app. It ensures that each enumerator has a unique identity and allows for personalised data collection.
2. **User Profile Creation**: On the Add User page, enumerators can input relevant information to create their user profiles. This typically includes details such as their name and password.
3. **User Authentication**: The Add User page may also provide the option for enumerators to set up a password for their user profiles. This authentication feature ensures that only authorised individuals can access and utilise the app using their designated user

accounts.

4. **User Account Flexibility**: The Add User page allows for user accounts to be created with or without a password. This flexibility accommodates varying security requirements and administrative preferences. Enumerators can choose the appropriate account setup based on their specific needs.
5. **Accessibility upon Session Availability**: The Add User page is the initial page displayed when a session becomes available in the app. Enumerators are prompted to create their user profiles before proceeding to other data collection activities.

Survey Selection Page

Once a user is added, the user is directed to the "Survey Selection" page. This page displays the available surveys/forms for the user to select from, presenting an overview of the options for data collection.

1. **Purpose**: The Survey Selection page serves as the central hub where enumerators can browse and select surveys or forms specific to their assigned tasks. It provides an overview of available surveys and relevant statistics, aiding in efficient data collection.
2. **User-Specific Surveys:** The Survey Selection page displays surveys or forms based on the assignment code entered earlier. All users with the same assignment code will have access to the same set of surveys. This ensures consistency among enumerators working on the same project and facilitates centralised data collection.
3. **Survey List and Statistics**: Enumerators are presented with a list of surveys or forms relevant to their assignment code. Each survey entry includes basic statistics, such as the number of submitted responses or other relevant metrics. These statistics offer valuable insights into the progress of data collection and help enumerators make informed choices.
4. **Intuitive User Interface:** The Survey Selection page is designed with an intuitive user interface, allowing enumerators to navigate through the available surveys easily. Clear navigation elements, search functionality, or filtering options may be provided to enhance usability and enable efficient survey selection.

Select Action from Selected Form

After selecting a survey/form, the user is presented with options to perform actions related to the selected form. The available actions include starting a new survey, editing a saved submission, or viewing a submitted submission. This single page provides enumerators with clear navigation options for managing the selected form. Here's a more detailed explanation:

1. **Purpose**: The "Select Action from Selected Form" page aims to provide enumerators with clear and distinct actions they can take regarding the selected form. It serves as a centralised control panel for managing the form and offers intuitive navigation options.
2. **Single Page Design:** The decision to have a single page with the three buttons - Preview, Start New, Edit Saved Submission, and View Submitted Submission - is to present enumerators with a comprehensive overview of the available actions in a clean and straightforward manner. This layout minimise confusion and allows for easy access to different functionalities.

3. **Preview Button:** The Preview button enables enumerators to access a preview of the selected form. By clicking this button, enumerators can review the form's structure, questions, and any other relevant details before proceeding with data collection. This feature helps them become familiar with the form's content and layout.
4. **Start New Button:** The Start New button initiates a new data collection session for the selected form. Enumerators can begin entering data and providing responses based on the form's questions. This button is used when starting a fresh data collection process for the selected form.
5. **Edit Saved Submission Button:** The Edit Saved Submission button allows enumerators to modify a previously saved but incomplete submission for the selected form. By clicking this button, enumerators can access the partially completed submission and make any necessary changes or additions.
6. **View Submitted Submission Button:** The View Submitted Submission button provides access to previously submitted and completed submissions for the selected form. Enumerators can review the data they have previously submitted, ensuring data accuracy and allowing for reference purposes.

Starting The Submission

This process combines three key elements: The Timer, Question Group Navigation, and Submission. Here's a detailed explanation of how these components work together:

1. **The Timer:**
   - When the enumerator starts a new data collection session for a selected form, a timer is initiated in the background. The timer tracks the duration of the survey from the moment it is started until it is submitted. This feature provides valuable insights into the time taken for each survey and helps monitor data collection efficiency.
2. **Question Group Navigation:**
   - To streamline the data collection process and avoid overwhelming the enumerator, not all question groups are displayed simultaneously. Instead, the form is divided into logical sections or question groups. Enumerators proceed through the form by answering questions within the current question group.
   - The app enables question group navigation by presenting navigation buttons, such as "Next" and "Previous." The "Next" button is disabled until all questions within the current group are completed. Once the enumerator finishes answering the questions, the "Next" button becomes active, allowing them to proceed to the next question group.
   - This approach helps maintain focus and clarity, guiding enumerators through the form step-by-step and reducing the risk of missing or skipping questions.
3. **Submission:**
   - Once the enumerator completes all the questions in the form, the app provides a final step: the submission process. At this stage, the "Next" button for the question group transforms into a "Submit" button.
   - Clicking the "Submit" button submits the completed form to the server, finalising the data collection process. The app may display a confirmation prompt to ensure the enumerator intends to submit the form.

- Upon submission, the collected data is securely transmitted to the server for further processing and analysis. The enumerator receives a confirmation message or notification, indicating a successful submission.

Back Button Confirmation and Three Dots Menu Options

In addition to the primary submission workflow, there are an additional behaviour that enhances user control and flexibility. This feature allows users to utilise the back button on their Android device and provides options for managing forms, saving submissions, and selecting language preferences. Here's a detailed explanation:

Back Button Confirmation:

When users click the back button on their Android device while in the midst of data collection, a confirmation popup appears. This popup offers users a choice between two options:

- **Save and Exit**: Users can choose to save the current submission and exit, allowing them to resume data collection from the same point at a later time. The saved submission is stored for future retrieval and modification.
- **Exit without Saving**: Users can opt to exit the current form without saving the submission. This choice allows them to return to the form selection screen without storing any incomplete data.

Kebab Menu Options:

The NMIS mobile app includes a three dots menu, located in the top-right corner of the screen, which provides additional options in submission:

- **Save and Exit:** This option allows users to save the current submission and exit the form, ensuring that progress is retained for future continuation.
- **Exit without Saving:** This option enables users to exit the form without saving the submission, providing a quick way to return to the form selection screen.
- **Language Selection:** The three dots menu includes the language selection feature, allowing users to choose their preferred language for the app's interface. By selecting their desired language, users can customise the app to suit their language preference.

Back-end API Endpoints

To support the functionality of the NMIS mobile app, certain back-end API endpoints need to be implemented. These endpoints facilitate communication between the mobile app and the server, enabling various operations such as retrieving form lists, fetching individual form details, and syncing submissions. By interacting with these endpoints, the mobile app can seamlessly integrate with the back-end system and provide a smooth user experience. Here are the key API endpoints required for the NMIS mobile app:

Get the List of Assigned Forms

- Endpoint: **/auth**
- Method: **POST**
- Request Body:
  - ```
    {"code": "<assignment_code_provided_by_admin>"}
    ```
- Response:
  - ```
    {
      "message": "Success",
      "formsUrl": [
        {
          "id": 519630048,
          "url": "/forms/519630048",
          "version": "1.0.0"
        },
        {
          "id": 533560002,
          "url": "/forms/533560002",
          "version": "1.0.0"
        },
        {
          "id": 563350033,
          "url": "/forms/563350033",
          "version": "1.0.0"
        },
        {
          "id": 567490004,
          "url": "/forms/567490004",
          "version": "1.0.0"
        },
        {
          "id": 603050002,
          "url": "/forms/603050002",
          "version": "1.0.0"
        }
      ],
      "syncToken": "Bearer eyjtoken"
    }
    ```
- Example:

- ```
  curl -X POST -d "code=testing123" -s http://localhost:8080/auth | jq
  ```

Get Individual Form

- Endpoint: **/forms/{formId}**
- Method: **GET**
- Response:
  - Return the JSON representation of the form

Send Submission

- Endpoint: **/sync**
- Method: **POST**
- Request Headers:
  - `Content-Type: application/json`
  - `Authorization: Bearer <syncToken>`
- Request Body:
  - ```
    {
      "name": "Iwan - 30 - Purbalingga",
      "submitter": "Iwan Firmawan",
      "duration": 2,
      "submittedAt": "2023-06-22T01:52:57.357Z",
      "answers": [
        {
          "12352546": "value1",
          "307454380": "value2"
        }
      ]
    }
    ```
- Response:
  - ```
    {
      "message": "Success",
      "id": 123
    }
    ```
- Example:
  - ```
    curl -s -X POST \
         --header "Content-Type: application/json" \
         --header "Authorization: Bearer eyjtoken" \
    ```

```
--data '{
  "name": "Iwan - 30 - Purbalingga",
  "submitter":"Iwan Firmawan",
  "submittedAt": "2023-06-22T01:52:57.357Z",
  "duration":2,
  "answers":[{"12352546":"value1", "307454380":"value2"}]
}' \
http://localhost:8080/sync
```

These backend API endpoints provide the necessary functionality for the NMIS mobile app to interact with the server and perform actions such as retrieving form lists, fetching form details, and syncing submissions.

Testing Strategy

To ensure the robustness and reliability of the NMIS mobile app, a comprehensive testing strategy will be employed, utilising the Jest testing framework. The testing approach will cover various aspects of the app's functionality, including component views, user inputs, data submission, and error handling. Here's an overview of the testing areas:

1. **Component View and User Input Testing:**
   - Unit tests will be conducted to verify the rendering and behavior of individual components within the app's views. This includes testing the correct display of UI elements, proper navigation between screens, and accurate rendering of form fields.
   - Interactive tests will be performed to simulate user interactions and inputs, ensuring that input fields accept user data correctly, respond to user actions appropriately, and display accurate feedback or validation messages.
2. **Data Submission Testing:**
   - Integration tests will be carried out to validate the end-to-end flow of data submission. This includes simulating the complete data collection process, populating form fields with sample data, and verifying that the submitted data is properly processed and stored on the server.
   - Tests will be conducted to ensure that all required fields are correctly validated and that data is submitted in the expected format. This includes testing scenarios where all mandatory fields are filled, as well as cases where optional fields are left empty.
3. **Error Handling Testing:**
   - Error scenarios will be tested to ensure the app handles exceptions, errors, and invalid inputs gracefully. This includes simulating network connectivity issues, server errors, invalid user inputs, and unexpected system behaviors.
   - Error handling tests will verify that appropriate error messages are displayed to users when errors occur, guiding them to take corrective actions or providing

relevant information to troubleshoot issues.

The testing strategy will involve writing test cases using Jest's testing framework. These tests will cover a range of scenarios, including positive and negative cases, edge cases, and boundary conditions, to ensure maximum test coverage and identify any potential issues or bugs.

Assumption and Constraints

1. **Device Compatibility:** It is assumed that the NMIS mobile app will be developed for Android and iOS devices, targeting a specific minimum operating system version for each platform. Compatibility with older device models may be limited due to hardware or software constraints.
2. **Internet Connectivity:** The app assumes that users will have access to a stable internet connection for certain features, such as data synchronisation and form retrieval. However, the app is designed to handle offline scenarios, allowing users to collect data in the field and sync when a connection becomes available.
3. **Assignment Code Usage:** The app assumes that assignment codes provided to enumerators are valid and have been properly generated and assigned by the administrators. It is also assumed that each code will correspond to a specific set of forms and configurations for data collection.
4. **Data Security and Privacy:** The app assumes that appropriate security measures are implemented on the server-side to protect user data and ensure privacy. It is assumed that user authentication, data encryption, and access control mechanisms are in place to safeguard sensitive information.
5. **User Training and Familiarity:** The app assumes that enumerators using the NMIS mobile app have received proper training on its functionality and usage. It is expected that users are familiar with basic mobile device operations and have a reasonable understanding of data collection procedures.
6. **Limited Budget for Device Acquisition:** It is assumed that there may be limitations on the budget available for acquiring mobile devices. Consequently, the app is designed to be compatible with a range of device models, ensuring wider accessibility for enumerators without the need for high-end or expensive devices.
7. **Language Localisation:** The app assumes that it will support multiple languages to cater to diverse user groups. However, the initial implementation may focus on a primary language, with plans to expand language support based on user demand and available resources.
8. **Server Infrastructure:** The app assumes the presence of a reliable and scalable server infrastructure to handle data synchronisation, form retrieval, and submission processing. It is assumed that the server infrastructure can handle expected user load and concurrent requests effectively.
9. **Regular Maintenance and Updates:** The app assumes that regular maintenance and updates will be performed to address bugs, security vulnerabilities, and enhance functionality based on user feedback and evolving requirements.

10. **Regulatory Compliance:** The app assumes compliance with relevant laws, regulations, and data protection standards in the regions where it is deployed. It is assumed that necessary measures will be taken to ensure compliance with data privacy, security, and confidentiality requirements.

Dependencies

List of dependencies required for the development of the NMIS mobile app, including React Native itself and other necessary libraries:

1. **React Native**: A JavaScript framework for building native mobile apps.
2. **Leaflet**: A JavaScript library for interactive maps, used for the geolocation type of form in the NMIS app.
3. **Pullstate**: A lightweight state management library for React applications, providing an easy and intuitive way to manage and share state across components.
4. **React Native Elements**: A UI component library for React Native, offering pre-designed UI elements and components to enhance app development.
5. **Formik**: A popular form management library for React, providing a simple and efficient way to handle form inputs, validation, and submission.
6. **React Navigation**: A routing and navigation library for React Native, allowing for easy navigation between screens and managing app navigation flow.
7. **Axios**: A JavaScript library for making HTTP requests, used for communicating with the server and retrieving data for the NMIS app.
8. **Moment.js**: A library for handling dates and times, providing utilities for parsing, formatting, and manipulating dates in JavaScript.
9. **Jest**: A library for handling tests

Error Handling

Error handling is a critical aspect of the NMIS mobile app's development, aimed at providing a seamless and user-friendly experience. As with any software application, various errors and exceptions may occur during its usage. To ensure smooth operation and effective user guidance, comprehensive error handling mechanisms have been implemented. The app employs a proactive approach to detect and handle errors, offering clear and concise messages that assist users in resolving issues. Below are some common errors that may arise during app usage, along with brief explanations of each:

1. **Invalid Assignment Code:** Occurs when the user enters an incorrect or unrecognised assignment code during the initial setup, prompting them to verify and re-enter the code.

2. **Network Connection Failure:** Arises when the app encounters connectivity issues, such as a lack of internet connection or poor signal strength. Users are notified about the problem and are guided to check their network settings or retry when a stable connection is available.
3. **Form Retrieval Failure:** If the app fails to retrieve forms from the server, an error message is displayed, suggesting users try again later or contact support for assistance.
4. **Missing or Incomplete Data:** Alerts users when attempting to submit a form with mandatory fields left blank or incomplete, ensuring all required information is provided before proceeding.
5. **Server Error:** Indicates server-related issues during data submission, such as database connectivity problems or server timeouts. Users are informed about the error and provided options to retry or seek further support.
6. **Authentication Errors:** Occur when there are authentication issues, such as invalid login credentials or an expired session. Users are guided to re-enter correct information or re-authenticate as required.
7. **Unexpected App Crashes:** In the event of unexpected errors or crashes, users receive user-friendly error messages informing them of the issue and suggesting actions like restarting the app or contacting support.
8. **Input Validation Errors:** Alert users to invalid data entries, such as incorrect phone number formats or email addresses, and provide specific error messages to facilitate proper data input.
9. **Insufficient Permissions:** Notify users when the app requires specific permissions to access device features and guide them to grant the necessary permissions for proper app functionality.
10. **General Error Handling:** Implement a robust error handling mechanism to catch and handle any unforeseen errors or exceptions, ensuring users receive informative and helpful error messages for effective troubleshooting and problem resolution.

Performance Considerations

Ensuring optimal performance is essential for delivering a smooth and responsive user experience in the NMIS mobile app. The following performance considerations have been taken into account:

1. **Efficient Rendering:** Techniques such as minimising unnecessary re-renders and utilising memoisation will be implemented to optimise component rendering. This improves CPU and memory usage efficiency, resulting in a more responsive user interface.
2. **Network Efficiency:** To reduce latency and data transfer, network requests will be optimised. Methods like data compression, caching, and request bundling will be employed to enhance network efficiency and decrease load times for data synchronisation and server interactions.
3. **Image and Media Optimisation:** Images and media files used within the app will be optimised to minimise file size while maintaining acceptable quality. Image compression and lazy loading techniques will be applied to improve loading times and reduce data

consumption.

4. **Code Optimisation:** JavaScript code will be optimised by eliminating unnecessary computations, optimising algorithms, and minimising function call overhead. Minification and bundling techniques will be utilised to reduce file sizes, resulting in faster app loading and startup times.
5. **Memory Management:** Effective memory management techniques, such as efficient data structures and proper resource cleanup, will be employed to minimise memory consumption. This prevents memory leaks and ensures app stability during prolonged usage.
6. **Offline Functionality:** The app will be designed to support offline data collection, enabling users to work without an internet connection. Local data storage and synchronisation mechanisms will be optimised to ensure seamless offline functionality and efficient data syncing upon connection restoration.
7. **Performance Monitoring and Optimisation:** Performance metrics will be monitored and analised using suitable tools and techniques. This includes profiling app performance, identifying bottlenecks, and optimising critical areas to enhance overall responsiveness and efficiency.
8. **Device Compatibility:** Extensive testing will be conducted across a range of target devices to ensure compatibility and performance on different hardware configurations. Performance optimisations will be tailored based on device capabilities to provide an optimal experience for users.
9. **Scalability:** The app's architecture will be designed to handle increased user loads and growing data volumes. Scalability considerations, such as efficient database design, caching mechanisms, and load balancing, will be implemented to maintain performance as the app scales.

Risks and Mitigation Strategies

1. **Security Breach:** Implement robust encryption algorithms and secure communication protocols to safeguard user data. Regularly perform security audits and updates to address vulnerabilities and ensure compliance with security standards.
2. **Data Loss or Corruption:** Implement regular data backups and utilise reliable data storage mechanisms. Employ redundancy measures and perform data integrity checks to minimise the risk of data loss or corruption.
3. **Inadequate Performance:** Conduct thorough performance testing to identify and address any bottlenecks or performance issues. Optimise code, queries, and server configurations to enhance app responsiveness and scalability.

4. **Compatibility Issues:** Perform compatibility testing across various devices, operating systems, and screen sizes. Adhere to industry standards and best practices to ensure the app functions correctly across different platforms and configurations.
5. **User Adoption Challenges:** Conduct user testing and gather feedback throughout the development process to identify and address usability issues. Provide clear documentation, tutorials, and user support to enhance user adoption and satisfaction.
6. **Lack of Internet Connectivity:** Provide clear guidance on the app's offline mode and syncing process.
7. **Integration Challenges:** Conduct thorough integration testing with external systems or APIs to identify and address any compatibility or connectivity issues. Collaborate closely with third-party providers and ensure proper documentation and support are available.
8. **Inadequate Training and Support:** Develop comprehensive training materials and provide training sessions to users to ensure they understand how to use the app effectively. Establish a dedicated support system to address user inquiries and issues promptly.
9. **Budget Overrun:** Implement a well-defined budget plan and regularly monitor expenses throughout the project. Prioritise essential features and conduct thorough cost estimation to ensure alignment with the allocated budget.
10. **Scope Creep:** Establish a clear scope and change management process. Regularly review and assess requested changes, evaluating their impact on timelines and resources. Implement proper communication and documentation to manage scope effectively.
11. **Lack of User Feedback:** Actively seek user feedback through surveys, feedback forms, or user testing sessions. Engage users in the development process and incorporate their suggestions and needs into future updates.
12. **Regulatory Compliance Issues:** Conduct thorough research to understand applicable regulations and standards. Implement necessary measures to ensure compliance, such as data privacy protections and adherence to industry-specific guidelines.
13. **Project Delays:** Develop a comprehensive project plan with realistic timelines and milestones. Regularly monitor progress, identify potential bottlenecks, and allocate sufficient resources to mitigate delays effectively.
14. **Inadequate Testing:** Implement a robust testing strategy, including unit testing, integration testing, and user acceptance testing. Conduct thorough test coverage and prioritise test case creation to identify and rectify potential issues.
15. **Lack of Scalability:** Design the app with scalability in mind, considering future growth and increasing user demands. Implement scalable architecture and regularly assess performance to ensure the app can handle increased usage.
16. **Insufficient User Training:** Develop comprehensive training materials and provide ongoing user training sessions.
17. **User Privacy Concerns:** Implement strong privacy measures, such as data anonymisation and user consent mechanisms. Comply with relevant data protection regulations and clearly communicate the app's privacy policy to users.
18. **Limited User Acceptance:** Conduct thorough user acceptance testing to identify and address any usability issues or gaps in user expectations. Incorporate user feedback and iterate on the app's design and functionality to enhance user acceptance.

19. **Technical Dependencies:** Identify and manage dependencies on external systems or APIs by establishing clear communication channels and integration protocols. Have contingency plans in place to handle any disruptions or changes in dependencies.

Implementation Plan

Task Breakdown

1. **Project Setup**
   - Set up the project repository on GitHub.
   - Create initial project structure and configuration.
   - Establish Asana project board for task management.
   - Define project milestones and deliverables.
2. **Requirements Gathering and Analysis**
   - Collaborate with all the team to gather and refine project requirements.
   - Conduct meetings and discussions to identify key features and functionalities.
   - Document the finalised requirements and create a functional specification.
3. **Design and Prototyping**
   - Collaborate with the designer to create the app's UI/UX design using Figma.
   - Iterate on the design, incorporating feedback from stakeholders.
   - Develop interactive prototypes for user testing and validation.
4. **Development**
   - Break down the project into smaller tasks and allocate them among the developers.
   - Implement the app's core functionality, adhering to the established design and requirements.
   - Utilise GitHub workflow for version control, code reviews, and pull request management.
   - Employ Asana for task tracking, assigning tasks to developers, and tracking progress.
5. **Testing and Quality Assurance**
   - Develop and execute test cases to ensure the app functions as intended.
   - Perform unit testing, integration testing, and user acceptance testing.
   - Identify and resolve any bugs or issues encountered during testing.
   - Utilise GitHub workflow for automated testing, continuous integration, and release management.
6. **Documentation**
   - Create comprehensive documentation, including installation instructions, usage guidelines, and API references.
   - Document the project's architecture, design decisions, and implementation details.
   - Maintain up-to-date documentation throughout the development process.
7. **Deployment and Release**
   - Prepare the app for deployment, including configuring server infrastructure and ensuring scalability.

- Utilise GitHub workflow for automated build processes, continuous integration, and release management.
- Conduct final testing and quality assurance checks before releasing the app to production.

Timelines

With three developers and the project supervisor also working as a developer, tasks will be distributed based on expertise and workload. The project manager will oversee task coordination, ensure timely progress, and provide overall project guidance.

Phase	Duration (in days)
Project Setup	2
Requirements Gathering and Analysis	3
Design and Prototyping	5
Development	30
Testing and Quality Assurance	7
Documentation	3
Deployment and Release	2

Asana will be used for task management, allowing for task assignment, progress tracking, and collaboration among team members. GitHub will serve as the repository for version control, code management, and pull request reviews. GitHub workflow will automate testing, build processes, and release management.

By following this implementation plan, utilising the allocated timelines and utilising Asana for task management, GitHub for version control, and GitHub workflow for testing, build, and release, the team can effectively coordinate and deliver the NMIS mobile app within the specified time-frame while ensuring quality and adherence to project requirements.

Communication Channels

To facilitate effective communication and collaboration among team members, the following Slack channels will be utilised:

Channel	Purpose
#proj-mis-mobile-app-module	General discussions related to the NMIS mobile app module.
#proj-mis-mobile-app-tech	Technical discussions specific to the NMIS mobile app module.
#proj-mis-app-dev-notification	GitHub notifications and code coverage reports.

> In the**#proj-mis-mobile-app-module** channel, there is a list of bookmarks accessible anytime. These bookmarks serve as quick references to important resources, documents, or discussions related to the NMIS mobile app module. They provide easy access to relevant information for team members to quickly navigate and retrieve essential resources.

Documentation References

To provide comprehensive information and guidance on the usage and functionality of the NMIS mobile app, the following documentation references will be made available:

1. **README.md**: The project's README file will serve as the primary source of documentation. It will contain essential details about the app, including an overview, installation instructions, setup guidelines, and basic usage examples. The README.md file will be located in the root directory of the project repository and will be regularly updated to reflect the latest changes and improvements.
2. **Read the Docs**: A dedicated documentation platform, such as Read the Docs, will be utilised to host detailed documentation for the NMIS mobile app. This platform allows for easy navigation, searchability, and accessibility of the documentation, making it convenient for developers, contributors, and users to find the information they need. The documentation will cover various aspects of the app, including installation, configuration, usage guidelines, API references, and troubleshooting.

The documentation will provide step-by-step instructions, code samples, configuration examples, and explanations of the app's features and functionalities. It will serve as a valuable resource for developers, administrators, and users to understand how to set up, customise, and utilise the NMIS mobile app effectively.

Conclusion

The development of the National Management Information System (NMIS) mobile app is an important step towards enhancing data collection efficiency and improving the overall management of national services. By utilising the power of JavaScript and React Native, the app provides a user-friendly and efficient platform for enumerators in the field to collect data seamlessly.

With a mobile-optimised interface and support for offline data collection, the NMIS app streamlines the data collection process and eliminates the complexities associated with web-based forms. Enumerators can easily access and fill out forms using various question types such as geolocation, text, number, options, cascade, photo/video, and date. The optional dependency logic and validation rules ensure the accuracy and integrity of the collected data.

The app's build parameters allow for customisation, enabling administrators to tailor the app to their specific requirements. The server URL parameter ensures seamless integration with the NMIS server, while the data sync interval parameter ensures efficient data synchronisation. The error handling and logging, as well as the debug mode, contribute to a robust and stable app experience.

The implementation plan outlines the necessary steps and timelines for developing the NMIS mobile app. With a team of developers, a project supervisor, a designer, and a project manager, the project will be well-coordinated and efficiently executed. Communication channels such as Slack will facilitate effective collaboration, while tools like Asana and GitHub will aid in task management, version control, and continuous integration.

By prioritising security considerations, the app ensures the protection of user data through data encryption, secure communication, and user authentication mechanisms. Performance considerations are also taken into account to provide a fast, responsive, and optimised user experience.

The project concludes with thorough documentation, testing strategies, and adherence to development best practices. Through regular updates, documentation references, and continuous improvement, the NMIS mobile app will meet the needs of national services and contribute to more efficient data management and decision-making processes.

Revision #48
Created 2023-06-13 16:59:39 UTC by Deden Bangkit
Updated 2024-06-17 05:00:22 UTC by Galih Pratama