TCS UI Framework & Solution Documentation

Version 1.0.1

December 18,2024

|  |
| --- |
| Notice |
| © 2024 Tata Consultancy Services Limited  This is a controlled document. Unauthorized access, copying, replication or usage for a purpose other than for which it is intended, are prohibited.  All trademarks that appear in the document have been used for identification purposes only and belong to their respective companies. |

|  |
| --- |
| Tata Code of Conduct |
| We, in our dealings, are self-regulated by a Code of Conduct as enshrined in the Tata Code of Conduct. We request your support in helping us adhere to the Code in letter and spirit. We request that any violation or potential violation of the Code by any person be promptly brought to the notice of the Local Ethics Counsellor or the Principal Ethics Counsellor or the CEO of TCS. All communication received in this regard will be treated and kept as confidential. |

Document Release Note

Notice No. :

Customer : Gulf International Bank

Project : Digital Channel Modernization

Document Details

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version Number | Created by | Reviewed by |
| 12-01-2024 | 1.0.1 | Jeswin Simon | Naman Shah |
| 12-19-2024 | 1.0.2 | Gopinath Venkatesan | Ashwani |
| 12-29-2024 | 1.0.3 | Gopinath Venkatesan | Ashwani |
| 01-08-2025 | 1.0.4 | Gopinath Venkatesan | Ashwani |
| 01-13-2025 | 1.0.5 | Gopinath Venkatesan | Ashwani |
| 01-27-2025 | 1.0.6 | Ashwani | Naman |
| 02-03-2025 | 1.0.7 | Latha Arjunan | Ashwani |

# About this Document

## Purpose of Documentation

This document provides comprehensive guidance and resources for Developers, ranging from beginners to experienced professionals and helps them understand and utilize the features of React effectively. This document details the approach that will be implemented to upgrade the technology stack of GIB, from version 1.0 to 2.0

## Target Audience

This section covers the categories of users who would refer to this document. The intended audience for the document is Tech Leads, Architects, Project Managers, Developers, new team members, and external Consultants or Partners.

Authorized Use Permission

This system and its associated data, software, reports, and documents are protected by copyright laws and are intended for authorized use only. Unauthorized access, use, or distribution of this system, or any part of it, including copying, modification, or redistribution of its contents, is strictly prohibited.

## Unauthorized Use

Accessing or using this system without proper authorization may result in disciplinary action, including but not limited to revocation of access rights, legal consequences, and potential criminal charges.

## Unauthorized Copies

Making unauthorized copies, reproductions, or redistributing any of the system’s data, software, reports, or documents is prohibited. This includes both physical and digital copies. Violation of these terms may result in legal action, including potential civil or criminal liability.

## Obtaining Permissions

If you require permission to use or copy any part of the system’s contents beyond what is granted by you authorized access, you must submit a formal request to the appropriate authority. Please ensure that you have obtained all necessary permission before using or distributing any system content beyond your authorized access level.

Failure to comply with these regulations may result in legal action and revocation of access to the system.

Contents

[About this Document 3](#_Toc717550852)

[Purpose of Documentation 3](#_Toc1644414)

[Target Audience 3](#_Toc670545882)

[Unauthorized Use 4](#_Toc2121373677)

[Unauthorized Copies 4](#_Toc519380357)

[Obtaining Permissions 4](#_Toc1070727710)

[2. Document Control 5](#_Toc907966777)

[2.1 Change Policy Document Management 6](#_Toc2053377450)

[3. Introduction 6](#_Toc1890265724)

[3.1 Brief Overview of the Framework 6](#_Toc1216319338)

[3.2 Key Features and Benefits 6](#_Toc589725844)

[4. Tech Stack 8](#_Toc1636060834)

[4.1 Architecture Overview for Mobile Application 8](#_Toc587492463)

[4.2 Architecture Overview for Web Application 9](#_Toc1354086672)

[4.3 Supported Versions 11](#_Toc655412542)

[4.4 Design Patterns 11](#_Toc446124907)

[4.5 Hooks Pattern 11](#_Toc1909886607)

[4.6. Atomic Design Pattern 12](#_Toc802604332)

[4.7 Code Dependency 12](#_Toc1588273802)

[5. Why Use TypeScript? 12](#_Toc1111700358)

[6. Project Structure, Coding Standards and Naming Conventions 13](#_Toc559365223)

[6.1 Project Structure 13](#_Toc100216775)

[6.2 Coding Standards 15](#_Toc519785894)

[6.2.1 TypeScript Compiler 15](#_Toc1705643514)

[6.2.2 ESLint 16](#_Toc1142696638)

[6.2.3 Prettier 16](#_Toc293899286)

[6.2.4 Naming Conventions 16](#_Toc2109571449)

[6.2.5 Enforcement of Coding Standards 16](#_Toc1951811965)

[6.2.6 SonarQube 16](#_Toc952295977)

[6.2.7 Local Storage 17](#_Toc38578312)

[6.2.8 Local Image Caching 17](#_Toc1587401157)

[7. Internationalization or Right to Left Support 17](#_Toc1569646290)

[8. Security 17](#_Toc1028379678)

[8.1 SSL Pinning for Mobile Application 18](#_Toc1074574617)

[8.2 Code Obfuscation for Mobile Application 18](#_Toc995427556)

[8.3 Prevent Screenshot or Screen Capture for Mobile Application 18](#_Toc1578528727)

[8.4 Two Factor Authentication for Mobile Application 18](#_Toc1694264274)

[8.5 Two Factor Authentication for Mobile Application 18](#_Toc653719204)

[8.6 SQL Injection for Web Application 18](#_Toc181779805)

[8.7 Session Management for Web Application 18](#_Toc888788549)

[8.8 Application Session Timeout for Web Application 19](#_Toc698593937)

[8.9 Local Storage- Encrypting Local Storage Using Crypto Library for Web Application 19](#_Toc1133123955)

[8.10 Authentication Mechanisms for Web Application 19](#_Toc1159576467)

[8.11 Data Security in Transit for Web Application 19](#_Toc1119575485)

[8.12 Disable Right Click on Web Page for Web Application 19](#_Toc1644529397)

[9. Network Service Request 19](#_Toc250690704)

[9.1 Service Integration 19](#_Toc600568320)

[10. Error and Exception Handling 20](#_Toc288442752)

[10.1 Logging Setup 20](#_Toc15756214)

[10.2 Log Levels 20](#_Toc553808207)

[10.3 Sending Logs to Logstash 20](#_Toc1521238167)

[10.4 Key Fields in the Log Format 20](#_Toc1579391408)

[10.5 Catching and Logging Errors 20](#_Toc134897102)

[10.6 Viewing Logs in Kibana 20](#_Toc622730720)

[11. Event or Crash Logging, Monitoring and Analytics 20](#_Toc557295426)

[12. Debugging Tools 21](#_Toc485222585)

[12.1 React Native Debugger 21](#_Toc1745382595)

[12.2 Profiler 21](#_Toc266036888)

[12.3 React Developer Tools 21](#_Toc1248183595)

[13. Sample User Journey 21](#_Toc2004456160)

[14. Validation 22](#_Toc1425010252)

[15. Utility 22](#_Toc36377703)

[16. Environment 22](#_Toc2080935690)

[17. Native Bridging 22](#_Toc762745146)

[18. Branching Strategy 23](#_Toc1635501129)

[19. Mobile App Versioning 23](#_Toc477601382)

[20. Separation of Concerns: 23](#_Toc1493317302)

[21.Clean Code Practices: 23](#_Toc399630597)

[22.Performance Benchmarks: 23](#_Toc1761432409)

[23. Admin Configurations: 24](#_Toc342899045)

[24. Bundled Certificate Management (SSL Pinning): 24](#_Toc1088041780)

[25. CI/CD Configuration: 24](#_Toc1129779115)

[26. Testing Strategy: 24](#_Toc96811218)

[27. Cache Invalidation: 25](#_Toc395680581)

[28. Client-Server Interaction 25](#_Toc1169565533)

[29. Code Reusability – npm Approach 26](#_Toc2011262708)

[Overview 26](#_Toc406887794)

[Key Features 26](#_Toc1472057797)

[1. Network Layer 26](#_Toc718630650)

[2. Theming 26](#_Toc1465251890)

[3. Utilities 27](#_Toc1739491754)

[4. Constants 27](#_Toc534853626)

[5. Types/Schemas 27](#_Toc294874425)

[Development Workflow 27](#_Toc795223369)

[1. Clone Repository 27](#_Toc1082717551)

[2. Install Dependencies 27](#_Toc41669265)

[3. Link Package: 27](#_Toc2131102766)

[30. Customized Themes 27](#_Toc2013756541)

[API Endpoints: 28](#_Toc1478681744)

[Fetch Active Theme 28](#_Toc585346672)

[31. Customized Menu 29](#_Toc1742489971)

[Sample Api Response: 29](#_Toc491712000)

[Backend Logic 31](#_Toc1540879607)

[32. Feature & Button Enablement 31](#_Toc1098155642)

[Sample Api Response: 32](#_Toc329724370)

[33. Glossary of Terms 33](#_Toc837812891)

# **2. Document Control**

A Document change refers to any alteration, modification, or update made to a document.

## 2.1 Change Policy Document Management

It refers to the process of systematically managing changes to documents, ensuring that all revisions are tracked, documented and implemented accurately.

# 3. Introduction

This section contains a brief overview of the framework, purpose of this documentation and its target audience.

## 3.1 Brief Overview of the Framework

React Native lets you write a code once and run it on both iOS and Android platforms. This significantly reduces development time and effort. It uses native components instead of web components, as a result, your app will have the look and feel of a native app.

## 3.2 Key Features and Benefits

The key features and benefits table provides a concise overview of essential functionalities and their advantages, helping in understanding how the tools can enhance their experience and streamline their operations.

Table 1: Key Features and Benefits for Mobile Application

|  |  |  |
| --- | --- | --- |
| Two Factor Authentication | Addsan extra layer of security for user accounts. | Implementing Touch ID for two-factor authentication. |
| Secure Storage | Ensures sensitive data is stored securely. | Utilizing Secure Store or Async Storage with encryption for storing user credentials. |
| Caching | Improves app performance by storing data locally for quick access. | Implementing React Query for caching API responses. |
| Root detection | Checks if the device is rooted to prevent security risks. | Implementing checks to restrict app functionality on rooted devices using libraries such as Jail Monkey |
| SSL Pinning | Technique to enhance the security of network communications in applications. | Implementing Certificate Pinning approach for SSL Pinning, to ensure only trusted certificates are accepted, enhancing security of network communications. |
| Code Obfuscation | To make our app small and fast as possible. | Using Android ProGuard to shrink and optimize our app. |
| Error Tracking | Tools for monitoring and reporting errors in the app. | Integrating Dynatrace to capture and report runtime errors and performance issues. |
| Exception Handling & Logging | Error and exception handling is essential for maintaining a robust and reliable application. | Implementing ELK to handle the exception and error logging mechanism. |
| Notification | Enables sending alerts and updates to users. | Integrating AppIce for push notification |
| Multi Language Support | Enables the app to support multiple languages for a global audience. | Using libraries such as i18next for internationalization and localization. |
| Customer Engagement | Mechanisms for user interaction and feedback. | Integrating Appice for customer engagement |
| Theming | Customizable themes for branding and visual appeal. | Allowing users to switch between light and dark modes using styled components. |
| Performance Improvement | Enhancements to optimize app speed and responsiveness. | Implementing Reactotron to identify the performance issue. |
| Code Scanner | Scans the app for security vulnerabilities. | Using tools such as SonarQube to identify and fix security issues in dependencies. |
| Analytic Capability | Tools for tracking user behavior and app performance. | Integrating AppIce analytics to monitor user interaction and app usage. |
| Coding standards | Enforces best practices and consistency in code. | Using ESLint and Prettier for maintaining code quality and formatting. |

Table 2: Key Features and Benefits for Web Application

|  |  |  |
| --- | --- | --- |
| Theming | Customizable themes for branding and visual appeal. | Allowing users to switch between light and dark modes using styled components. |
| Multi Language Support | Enables the web app to support multiple languages for a global audience. | Using libraries such as i18next for internationalisation and localisation. |
| Secure Storage | Ensure sensitive data is stored securely. | Utilising SecureStore with encryption for storing username, passcode. |
| Error Tracking | Tools for monitoring and reporting errors in the web application. | Integrating Error Boundaries to capture and report runtime errors and performance issues. |
| Coding standards | Enforces best practices and consistency in code. | Using ESLint, Sonar Lint and Prettier for maintaining code quality and formatting. |
| Code Scanner | Scans the web application for security vulnerabilities. | Using tools such as SonarQube to identify and fix security issues in dependencies. |
| Performance Improvement | Enhancements to optimize web application speed and responsiveness. | Implementing Profiler to identify the performance issue. |
| Exception Handling | Error and exception handling is essential for maintaining a robust and reliable application. | Implementing ELK to handle the exception and error logging mechanism |

# 4. Tech Stack

React Native is a cross-platform mobile development framework for building apps on both Android and iOS using a single codebase. It utilizes JavaScript or TypeScript and React and creates native mobile experiences with code reusability. With hot-reloading and access to native APIs, React Native enables rapid development and testing, making it popular for efficient app development.

## 4.1 Architecture Overview for Mobile Application

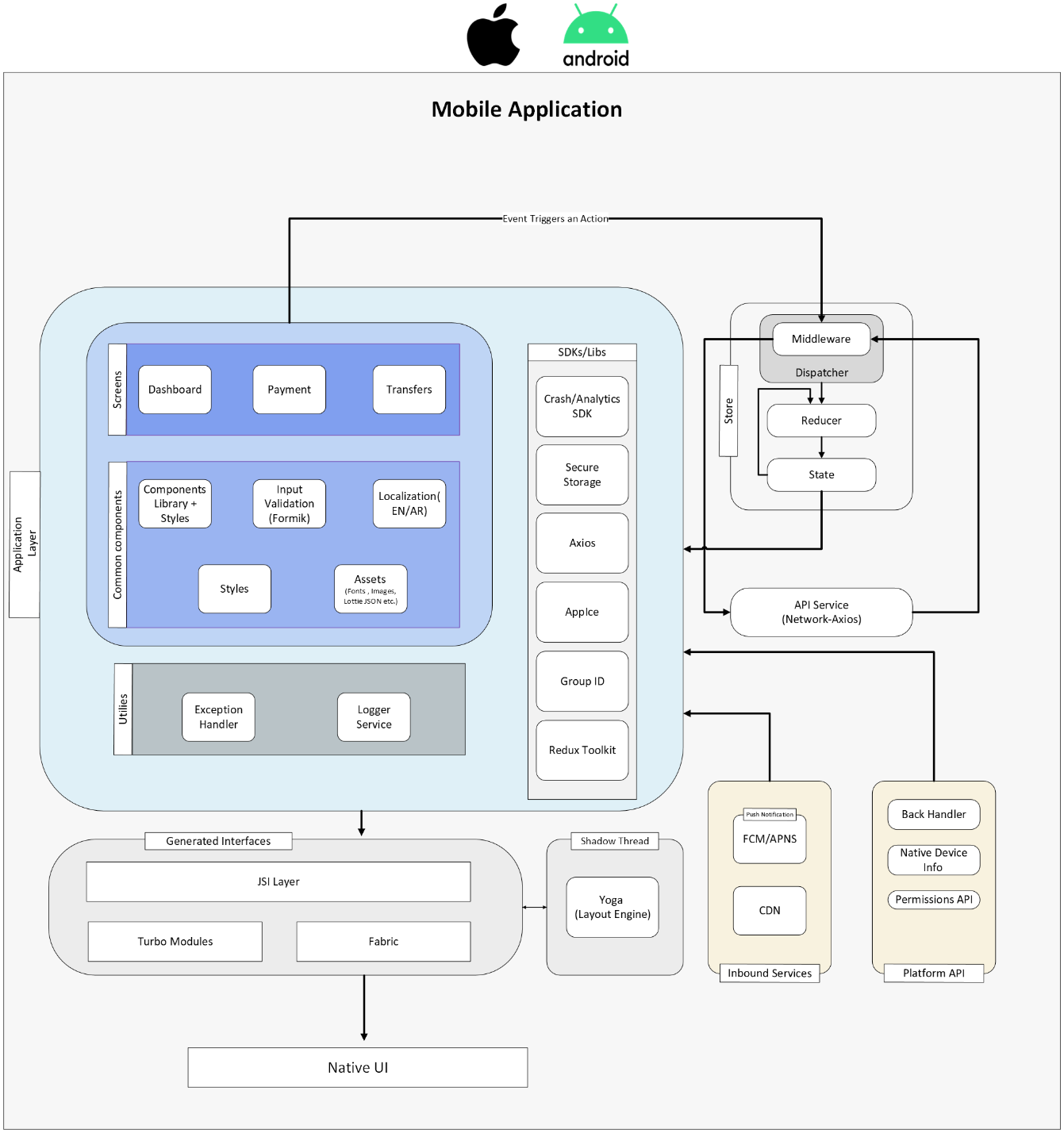
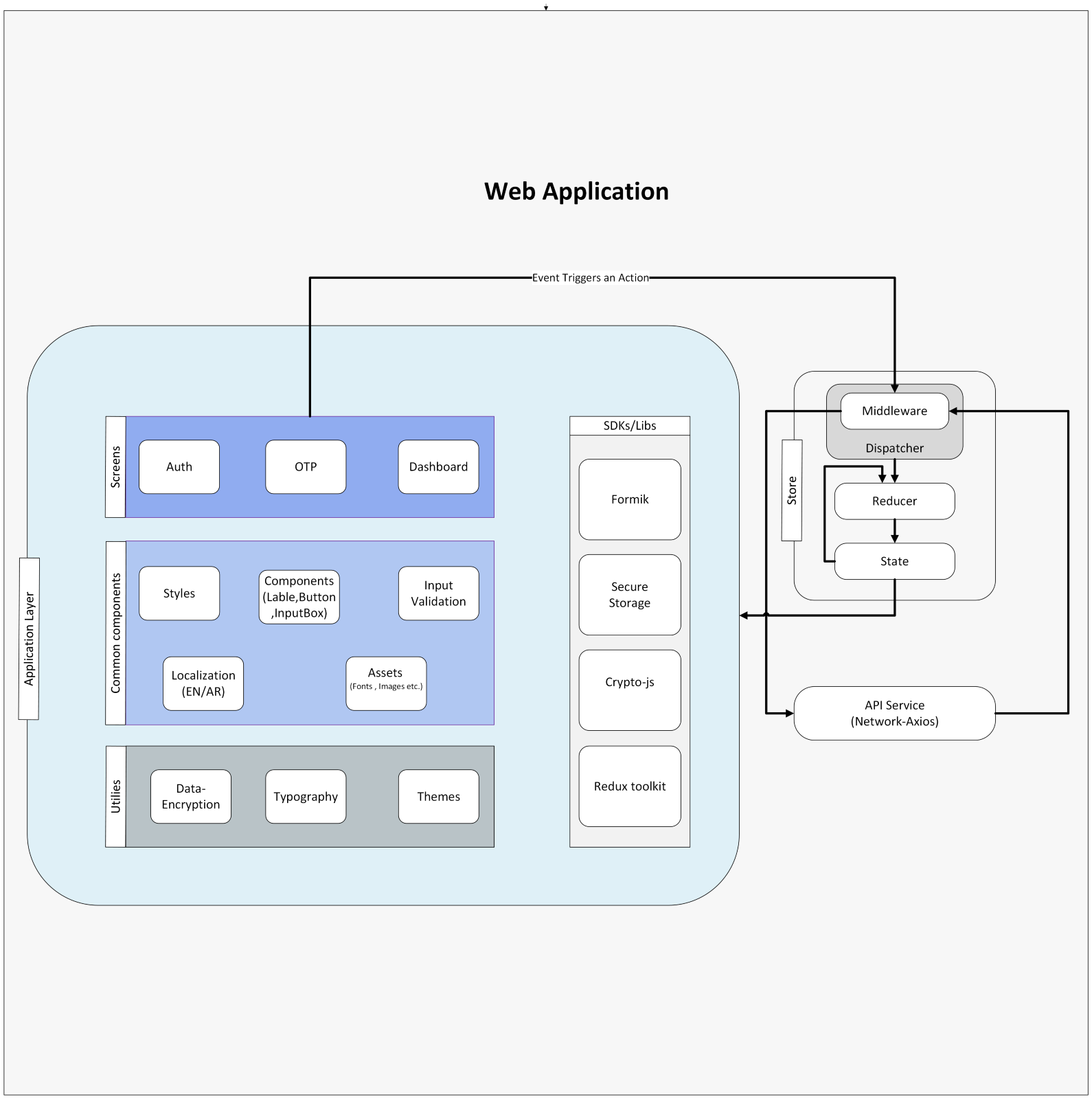


Figure 1: Platform Architecture Diagram for Mobile Application

## 4.2 Architecture Overview for Web Application

  
**Figure 2: Platform Architecture Diagram for Web Application**

**Application Layer:**

Application Layer constitutes all the User Interface related things which will be rendered by the Fabric. It also includes the SDKs, and the other Libraries used for the UI.**Generated Interface:**

Generated interfaces aim is to improve performance, flexibility, and the developer experience, particularly when dealing with native modules and rendering UI components.

**SDK / Library:**

SDK/Library is a collection of reusable code components and development kit that a developer can incorporate into their applications to handle specific tasks without having to write all the code by themselves.

**Inbound service:**

Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably send messages at no cost.

A content delivery network that caches content close to end users. A CDN allows for the quick transfer of assets needed for loading Internet content, like styles images etc.

**Platform API:**

React Native modules provide a bridge between JavaScript running in your application and native code running on the host platform. To access platform-specific APIs, you need to create a native module that exposes these APIs to your JavaScript code.

**Native UI:**

React Native provides a core set of platform agnostic native components like View, Text, and Image that map directly to the platform’s native UI building blocks

**Store:**

Stores the state and provides methods to use the state inside any component directly or to manipulate the state in a defined manner.

## 4.3 Supported Versions

Table 3: Supported Versions

|  |  |  |
| --- | --- | --- |
| Platform | Min Supported Version | Latest Version |
| iOS | 12 | Latest Version at the time of Go Live |
| Android | 6 (API Level 13) | Latest Version at the time of Go Live |

## 4.4 Design Patterns

Design patterns are established solutions to common problems that Developers encounter while building mobile applications. These patterns help organize codes, improve maintainability, and enhance performance.

## 4.5 Hooks Pattern

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Add state to a functional component

Reuse stateful logic among multiple components throughout the app.

Manage a component's lifecycle

Besides built-in hooks, such as useState, useEffect, and useReducer, we can create custom hooks to easily share stateful logic across multiple components within the application.

## 4.6. Atomic Design Pattern

Atomic Design is a design pattern that involves breaking down the application into smaller and reusable components. This pattern involves creating components at the atomic level, such as atoms, molecules, and organisms. Atoms are the smallest components, such as buttons and inputs, while molecules are groups of atoms, such as a form. Organisms are groups of molecules, such as a navigation bar. This pattern helps make the code more modular and reusable.

For example, imagine you have a form that includes a text input and a submit button. Instead of creating the form as a single component, you can break it down into smaller components, such as an input component and a button component. These components can be reused across the application.

The sample shared has this pattern implemented, for example, we have created atomic reusable button component. Sample code shared can we referred for such components.

## 4.7 Code Dependency

Incorporating dependency libraries into your React Native projects not only enhances functionality but also streamlines development and ensures your app is built on a solid foundation.

**Core Libraries**

Core Libraries is an essential set of components, APIs and Modules to develop React Native applications for Banking requirements. If any new library is installed in our application that will be directly added to our **package.json** file.

**Feature Libraries**

Feature Libraries are required to support additional features requirement of various features in typical modern applications with various images, animation, native integration, design requirements. This is as per current understanding of requirements in the app. All the feature libraries will be available in our **package.json** file.

**Optional Libraries**

Optional Libraries will be added based on requirement during development detailed design of react native application. All the optional libraries will be available in our **package.json** file.

# 5. Why Use TypeScript?

* TypeScript is a superset of JavaScript that adds static typing, making it an excellent choice for modern software development. Here’s why we use it and its purpose:
* **Type Safety**: TypeScript introduces types to JavaScript, which help prevent type-related runtime errors by catching them during development.
* **Better Tooling and IDE Support:** TypeScript enhances the development experience with features such as IntelliSense, autocompletion, and advanced refactoring. Code editors such as VSCode offer richer support when using TypeScript.
* **Detect Errors Early**: TypeScript compiles to JavaScript, but during this process, it checks for errors in the code, ensuring fewer runtime bugs.

# 6. Project Structure, Coding Standards and Naming Conventions

Organizing a React Native project with a clear structure, adhering to coding standards, and following consistent naming conventions is critical for developing scalable, maintainable, and collaborative applications.

## 6.1 Project Structure

The code within the framework is structured to follow the mentioned atomic design pattern and container pattern.

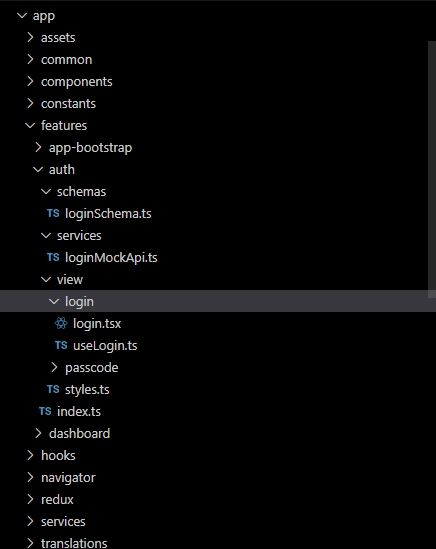


Figure 2: Project Directory Structure

The following table outlines the essential directories in our well-organized React Native project, along with their descriptions:

Table 4: Folder Structure

| Directory | Description |
| --- | --- |
| android (Mobile App) | Contains the native Android code and configuration files required to write or modify native Android code. |
| ios (Mobile App) | Contains the native iOS code and configuration files essential for writing or modifying native iOS code. |
| app (Mobile App) | Main folder for JavaScript/TypeScript code such as components, screens, and services. |
| public (Web App) | Store the location of the required files in the environment variable. This allows the files to be accessed directly in the components by using process. |
| src (Web App) | Main folder for JavaScript or TypeScript code~~,~~ such as components, screens and services. |
| assets | Contains all static resources, including images, fonts, and icons. It contains fonts, images and lottie files. |
| components | Includes all atomic components utilized within the application. Each component directory within this directory contains separate files for its declaration, type definitions, styles, and story definitions. It contains the following files: button, pressable, image-background, input, image, icon, loader, checkbox, label etc. |
| features | This directory includes all the view containers, and the services used within the application. Each feature is composed of view and services folder. It contains the following folders: app-bootstrap, dashboard, auth folder. Inside each of these folders there are two files, a service file and a view file. |
| hooks | Includes custom hooks for reusable business logic with separation based on Reusability, Readability, Testability. |
| config | Holds all the environment specific configuration data. |
| constants | Holds all the declared constants such as error messages, image paths and endpoints. It contains the following files: images, routes, error-values. |
| navigator | Includes all the routing files and stack for navigation within the application. It contains the following files: root-navigation, main-section, auth-section. |
| redux | Contains the definition of the redux store. It contains the files related to overall app centralized state management and data persistence in which data caching can be incorporated based on the use cases |
| schemas | Contains the structure of data models. |
| services | Contains the app-global, mock-service, endpoints and http services. |
| translations | Contains the default language labels in English and Arabic. It contains the ar.json and en.json files. |
| utils | Contains defining shared utilities methods such as animations, formatter and validations. |

## 6.2 Coding Standards

This section outlines the guidelines and best practices that Developers must follow to ensure code quality and collaboration efficiency.

## 6.2.1 TypeScript Compiler

TypeScript is a powerful tool for adding static types to JavaScript, enhancing code quality, and reducing runtime errors. Type-checking rules can be configured within the **tsconfig.json** file in our sample code to enforce strict type adherence.

## 6.2.2 ESLint

ESLint is a powerful, open source linting tool designed to maintain quality and enforce coding standards by identifying syntax errors, style issues and potential bugs. Our current ESLint configuration extends from **.eslintrc.js** file in our sample code, which allows further customization of the rules according to specific requirements.

## 6.2.3 Prettier

Our current Prettier configuration extends from **.prettierrc.js** file in our sample code, which helps to maintain a uniform codebase by enforcing a consistent code style, which improves collaboration and reduces the time spent on code reviews.

## 6.2.4 Naming Conventions

The following naming convention rules are to be followed:

* Use kebab case for all directory names (for example, assets/, image-picker/).
* Use camelCase for all file names (for example, index.tsx, imagePicker.tsx).
* Use PascalCase for all class names (for example, ImagePicker).
* Use camelCase for all function names (for example, fetchUserData()).
* Use UPPER\_SNAKE\_CASE for all constants (for example, BASE\_URL).
* Use camelCase for prop and state names.
* Start all custom hook names with use and the names must be in camelCase (for example, useTheme).
* Use snake\_case for all image and asset file names (for example, user\_avatar.png).

## 6.2.5 Enforcement of Coding Standards

Coding standards are enforced locally by configuring pre-commit hooks with [Husky](http://github.com/typicode/husky). Husky can be configured to automatically format code, check for type or lint errors before each commit and prevents commits if there are unresolved issues.

GitHub actions can be configured to run the same test and prevent a pull request from being merged if there are any unresolved issues.

## 6.2.6 SonarQube

SonarQube is an open-source platform, which provides detailed reports on code quality metrics such as code duplication, complexity, and test coverage.

## 6.2.7 Local Storage

Sensitive data such as user identifier, which needs to be persisted beyond the user session are securely stored in iOS keychain and Android keystore using react-native-mmkv

## 6.2.8 Local Image Caching

React Native Fast Image is a library that provides fast and efficient image caching. It improves performance by caching images locally.

# 7. Internationalization or Right to Left Support

The application supports both English and Arabic languages with Right-to-Left (RTL) support for Arabic. Our application contains two files **ar.json** for Arabic translations and **en.json** for English translations.

Internationalization is handled using the **react-i18next** library. It provides a seamless way to manage translations and localization within the app.

# 8. Security

Table 5: Security for Mobile Application

|  |
| --- |
| Security Checklist |
| Root or jailbreak detection |
| VPN detection |
| SSL pinning (bundled cert) |
| Code obfuscation |
| Geo-location data capture or restriction |
| Use of secure storage |
|  |
| Prevent debugging tools |
| Application security testing – SAST (using DevOps and IDE) |
|  |
| Secure communication/TLS |
| Authentication and session management |
| Handling multi device login |

## 8.1 SSL Pinning for Mobile Application

Our devised approach of using SSL certificates directly for pinning in applications is commonly referred to as ‘Certificate Pinning’. This method involves embedding the actual SSL certificate within the application, allowing it to verify that the server's certificate matches the pinned certificate during a secure connection.

## 8.2 Code Obfuscation for Mobile Application

Hermes, the default JavaScript compiler used in React Native, includes a built-in code obfuscator that obfuscates React Native code written in JavaScript. Additionally, the rules are configured in **android >app>build.gradle** file for android to handle code obfuscation and optimization [ProGuard](https://developer.android.com/build/shrink-code) for Android builds. And in project settings in iOS, using LLVM-Obfuscator settings.

## 8.3 Prevent Screenshot or Screen Capture for Mobile Application

To enhance mobile app security such as unauthorized access and to ensure that sensitive data cannot be easily captured, we have implemented screen capture disable in the mobile app by using Android custom Native function.

## 8.4 Two Factor Authentication for Mobile Application

Touch ID is a [React Native](http://facebook.github.io/react-native/) library for authenticating users with biometric authentication methods on both iOS and Android app by integrating [react-native-touch-id.](https://www.npmjs.com/package/react-native-touch-id)

## 8.5 Two Factor Authentication for Mobile Application

The following points should be noted to ensure security:

* Ensure that all communication between the web and server is done over HTTPS, which uses SSL or TLS to encrypt data.
* Ensure all URLs associated with the application are using the industry standard https: protocol.
* This includes ‘GET’ and ‘POST’ requests for images, documents, user login credentials, and other commonly transferred data.

## 8.6 SQL Injection for Web Application

All inputs provided by a user in a web application must be filtered at the device level to prevent the intentional or unintentional injection of invalid characters.

## 8.7 Session Management for Web Application

Session management and lifetime may be an issue for web applications running on smartphones. Since the browser does not exit, non-expiring cookies may persist for a long time. Authenticated web applications that may be used on a smartphone should have explicit session lifetimes defined.

## 8.8 Application Session Timeout for Web Application

Implementing application timeouts in React applications that handle sensitive data is crucial. Logging users out automatically after a given amount of inactivity reduces security risks and prevents unauthorized access if they leave their session unattended. React application timeout increases security by automatically logging users out after a preset inactivity. Other benefits include the following:

* **Security Enhancement**: Ends idle sessions to prevent data leaks and unwanted access.
* **Improved User Experience:** Notifies users about session expiration so they can act or re-authenticate without interruptions.

## 8.9 Local Storage- Encrypting Local Storage Using Crypto Library for Web Application

The data encryption file contains the functions used to encrypt and decrypt data. Encryption ensures that sensitive information such as user credentials and API keys are protected from unauthorized access.

## 8.10 Authentication Mechanisms for Web Application

Loginusing username and password through OAuth2 and Token-Based Authentication. When a user accesses the application, they are redirected to keycloak for login.

Upon successful authentication, keycloak issues an access token which is used to authenticate API calls. Optionally, a refresh token is issued to obtain a new access token after expiration without requiring the user to login again.

## 8.11 Data Security in Transit for Web Application

The key focus is on encryption to protect data in transit. It is not yet implemented in our framework and will be implemented in future.

## 8.12 Disable Right Click on Web Page for Web Application

This feature is not yet implemented in our framework and will be implemented in future.

# 9. Network Service Request

Axios is used to handle network service requests within the application. Axios is a promise-based HTTP client for JavaScript that lets you make requests to both their own server and third-party APIs.  
  
We have also utilized Interceptors in the framework to intercept requests or responses before they are handled by `then` or `catch`.

## 9.1 Service Integration

The service folder in our framework encapsulates the business logic and functionality related to features, leveraging Redux Toolkit Query to handle asynchronous operations.

# 10. Error and Exception Handling

Error and exception handling is essential for maintaining a robust and reliable application. We integrated Elasticsearch, Logstash, Kibana (ELK) stack to track and analyze logs. By centralizing logs, ELK stack enables comprehensive monitoring of errors and exceptions.

## 10.1 Logging Setup

The logging setup is designed to capture, format and send logs from the application for monitoring, debugging, and analytics purposes.

## 10.2 Log Levels

Our application categorizes logs into four levels to distinguish between types of messages.

The ‘console’ object in JavaScript provides several methods for logging messages, each serving a different purpose. Here is a detailed explanation of the ‘info()’, ‘error()’, ‘warn()’, and ‘log()’ methods:

## 10.3 Sending Logs to Logstash

Logs are sent to Logstash using Axios requests, where they are processed and indexed in Elasticsearch for retrieval in Kibana.

## 10.4 Key Fields in the Log Format

Following are the fields in the log format: aapName, userId, level, message, description, timestamp, component, headerContext, additional Info.

## 10.5 Catching and Logging Errors

React Native allows error handling within components and functions. It uses try-catch blocks around risky operations and logService.error to capture these incidents.

## 10.6 Viewing Logs in Kibana

Once the application logs are successfully sent to Elasticsearch through Logstash, the user can view and analyze these logs in Kibana.

# 11. Event or Crash Logging, Monitoring and Analytics

Dynatrace crash analytics is a robust for application performance, monitoring and crash reporting tool that helps Developers track and analyze crashes in their applications. With the integration of Dynatrace into our React Native app, we can gain valuable insights into the stability of the app, identify issues, and improve the overall user experience.

# 12. Debugging Tools

## 12.1 React Native Debugger

React Native provides a bundled debugger for debugging and performance monitoring. It includes a set of features, including a JavaScript debugger, performance monitor and memory profiler.

## 12.2 Profiler

React Profiler is a tool that helps measure the performance of react components by tracking their render times and updates. It should be used only when necessary. Each use adds some CPU and memory overhead to an application.

## 12.3 React Developer Tools

React Developer Tools is a Chrome DevTools extension for the open source React JavaScript library. It allows us to inspect the React component hierarchies in the Chrome Developer Tools.

There are two new tabs in your Chrome DevTools namely, Components and Profiler.

The Components tab shows you the root React components that were rendered on the page, as well as the subcomponents that ended up rendering.

By selecting one of the components in the tree, you can inspect and edit its current props and state in the panel on the right. In the breadcrumbs, you can inspect the selected component, the component that created it, the component that created that one, and so on.

# 13. Sample User Journey

This section outlines a typical user journey through the registration process in a mobile application built using React Native framework. Understanding this journey helps Developers set up, customize, and troubleshoot the registration flow to enhance the user experience.

The registration flow allows new users to create an account by entering their personal details, verifying their credentials, and setting up their profiles. The development process of this involves the following high-level steps.

* Setup boilerplate code for registration screen
* Write UI screen component for registration screen
* Setup state management
* Setup validations and exception
* Setup navigation
* Setup localization support
* Setup of data models
* Setup of API integration
* Create a new folder for your service in the feature folder.
* Create the api service file setup for your service.
* The queries can be triggered inside the feature hooks just by importing them.

# Testing and deployment

# 14. Validation

Formik simplifies form handling by providing a set of components and hooks that manage form state, validation, error handling, and submission processes. It provides a comprehensive set of validation rules to ensure the form data integrity.

**Key validation rules of Formik : Field-Level Validation, Form-level Validation, Yup Integration, Synchronous and Asynchronous Validation.**

# 15. Utility

Utility functions in programming are reusable pieces of code designed to perform common tasks. They help streamline development by reducing redundancy and improving code organization. Here is a deeper look into utility functions. A few of the function utilities we use in our framework are Typography Utility, Animations Utility, Network Connect Utility, Theme Utility, Date Utility, Number Utility.

# 16. Environment

For setting up multiple environments in React Native, we have to create separate configuration files for environments such as development, staging, and production. Please refer to the sample framework (.env.production,. env.staging, env.development)

# 17. Native Bridging

Native bridging in React Native is a mechanism that enables communication between JavaScript and native code, allowing Developers to utilize platform-specific features not directly accessible through React Native. This bridge facilitates the exchange of data and function calls between the two environments. Native Modules, Native UI Components, Events are various methods of Native bridging.

**Native Modules:**

The Native Module system exposes instances of Java/Objective-C/C++ (native) classes to JavaScript (JS) as JS objects, thereby allowing you to execute arbitrary native code from within JS.

# 18. Branching Strategy

We have established a robust branching strategy to streamline the development process. Each feature will have dedicated sub-branches for granular development and testing, which will ultimately be merged into the main feature branch for integration.

We will have a separate branching strategy document.

# 19. Mobile App Versioning

We will follow the Semantic Versioning (SemVer) format:

* Major: Increased when there are significant changes or updates that may include breaking changes. (e.g., 1.0.0 → 2.0.0)
* Minor: Increased when adding new features that are fully functional without altering existing ones. (e.g., 1.0.0 → 1.1.0)

# Patch: Increased when fixing bugs or making small improvements that do not affect existing functionality. (e.g., 1.0.0 → 1.0.1)

# 20. Separation of Concerns:

* Organize the project by grouping related files (e.g., components, hooks, utilities) to keep responsibilities clear and isolated.
* Keep UI components focused on rendering, while delegating logic and state management to separate hooks.
* Centralize state management in redux-toolkit, avoiding logic in the UI components.
* API calls to reside in RTKQuery files.

# Maintain theme/styling separately.

# 21.Clean Code Practices:

* Use meaningful names for functions, variables, and files to ensure clarity.
* Break tasks into small functions that each handle a single responsibility.
* Eliminate duplication by reusing common logic through services or utilities.
* Automate code formatting with tools like ESLint and Prettier to maintain consistency.

There will be a separate document for Coding Practices & Guidelines.

# 22.Performance Benchmarks:

We will follow industry wide performance benchmarks as below:

* API response times should remain optimal, with a focus on speed and reliability.
* Memory usage will be kept below 150 MB for mobile apps and 300 MB for web apps.
* A consistent UI Thread FPS of 60 and JS Thread execution time of 16 ms per 60 frames. Except for the low-end devices. Overall, we should follow the React Native Performance Benchmarks. <https://reactnative.dev/docs/performance>
* CPU usage will stay under 70-80% during peak usage and under 50% during regular operations.
* Battery consumption will be managed to stay under 10-15% for one hour of active use.

# 23. Admin Configurations:

We will have the ability to enforce app upgrades across all users to ensure they are using the latest version with critical patches.

Admins will be able to enable or disable main and sub-features based on user segments or customer needs, allowing flexibility in feature rollouts and tailored experiences.

Features can be selectively enabled or disabled for specific user segments, enabling targeted functionality.

Certain features can be enabled or disabled based on customer-specific configurations, providing a customized app experience for different customers. Admins will manage this configuration through the central admin interface.

We will have a Rest based API, to fetch config on the app launch, only if the config is updated on the server.

# 24. Bundled Certificate Management (SSL Pinning):

We will handle bundled certificate management, ensuring timely renewal and replacement of certificates before they expire, to avoid any potential downtime or service disruptions. This will be addressed during the development phase of the project. In the worst-case scenario, we should have some mechanism to disable Pinning/Validation and allow requests without certificate validation, subjected to, if it is possible to do this in React Native.

# 25. CI/CD Configuration:

As part of the DevSecOps document, CI/CD configurations will automate the build, testing, and deployment processes. This will help ensure that code is integrated and delivered smoothly and quickly, making the entire process more reliable and efficient without manual effort.

26. Testing Strategy:  
The TCS IQA team will be responsible for writing unit test cases for the project. These test cases will cover various functionalities and ensure code correctness. After writing, the test cases will be reviewed by the GIB team to ensure they meet the required standards and are comprehensive. Developers should use the same tests for unit testing. The tests will be conducted manually to validate functionality.

We should implement StoryBook approach only for the components that are complex in nature, for example chart or seek bar components. We can plan this before starting any task. We will not use this approach for simpler/dumb components.

# 27. Cache Invalidation:

In memory cache invalidation, Redux reducers will be reset on a case-by-case basis. We will utilize the RTKQuery cache mechanism to ensure proper cache invalidation, keeping data accurate and up to date across the application. For example. For mutations like account creation, it should invalidate the previous related RTKQuery response and fetch the new list of accounts (including new one).

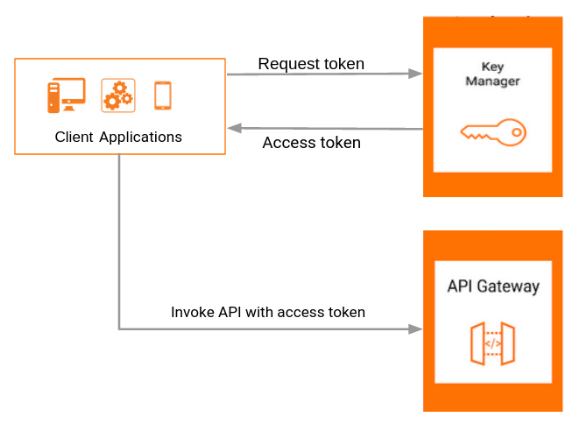
Another example, for Onboarding once the onboarding is completed it should reset the relevant state(reducer) and free the memory.

# 28. Client-Server Interaction

The app communicates with backend services using REST APIs secured with OAuth 2.0 and JWT tokens. WSO2 API Management and Identity Server are used to handle API management and authentication, ensuring secure and efficient client-server communication.

Endpoints of Key manager will be used to generate OAuth Access Token.

APIs exposed by WSO2 API Gateway will be secured. When a client (mobile or Web) invokes an API with an OAuth 2.0 access token or an API Key.



# **2**9. Code Reusability – npm Approach

# **Overview**

This NPM package provides a reusable, modularized network layer, theming system, utilities, types and constants designed to streamline development for both mobile (React Native) and web (React) applications. It ensures consistency, scalability, and maintainability across projects.

## Key Features

## 1. Network Layer

Simplified API calls with Axios integration.

Built-in support for interceptors (e.g., authentication, error handling).

Configurable base URLs and headers.

## 2. Theming

Unified theming system compatible with react-native and styled-components.

Provides light and dark mode support.

Centralized theme definitions for consistency.

The themes will be configurable, change in one/central place, will apply app wide.

Common theme attributes are colors, bg colors, font sizes, font families, margins (as per business requirements)

## 3. Utilities

Common utility functions for formatting, validation, and transformation

Date, string, and array manipulation helpers.

Platform-specific utilities (e.g., for React Native or browser).

## 4. Constants

Centralized app constants (e.g., API endpoints, error messages).

Environment-specific configurations (e.g., dev, staging, production).

## 5. Types/Schemas

Types of files should contain all the TypeScript interfaces

Crucial for code maintainability and readability

## Development Workflow

## 1. Clone Repository

git clone <https://github.com/your-repo-name.git>

cd your-repo-name

## 2. Install Dependencies

npm install

## 3. Link Package:

npm link

Now move to your app folder and execute the following command to install this as dependency and use the re-usable module code in your application.

npm link pm-package-name

The npm repo and artifact will be private to GIB (on premises)

# 30. Customized Themes

We will have pre-defined themes and the ability to have segment-based themes. Theme list and configuration will be managed in CMS and integrated through API. Our application supports dynamic theme management by fetching themes from the server based on the user's login status. Each user will have one active theme at any given time, Themes are customized based on several criteria: user segments, Customer Identifier Number (CIF), and event-based themes.

The backend manages the priority order dynamically, ensuring the most relevant theme is applied. For each session, themes are fetched and applied without exposing priority details to the client. Event-based themes ensure consistency throughout the user's session and only update at the next app launch, maintaining user experience during active sessions.

## API Endpoints:

## **Fetch Active Theme**

This API fetches the active theme during launch, applicable for both logged-in and non-logged-in users.

* Endpoint: /api/themes/active
* Method: GET

**Request Headers**

http

Authorization: Bearer <token> // Only for logged-in users

Content-Type: application/json

**Sample Api Response:**

{

"status": "success",

"data": {

"theme": {

"id": "theme\_id",

"name": "Theme Name",

properties: {

}

},

"message": "Active theme fetched successfully."

}

Once the theme is fetched, the application applies the theme's properties, such as color scheme, font, and background image.

Consistency is maintained throughout the session, with updates occurring on subsequent app launches based on the backend logic.

During onboarding, the same API is used without an authorization token. After the user logs in and registers, the theme API is called again with the authorization token to apply user-specific themes.

The backend determines the active theme based on user-specific criteria: segments, CIF, and events. It manages the priority order and ensures the theme's validity during session transitions.

By utilizing this API and logic, the application delivers a dynamic and personalized user experience through theme customization, enhancing overall engagement.

# 31. Customized Menu

We will have configurable menu items. The menu items can have hierarchy like menu -> sub-menu. The menu items also like themes will be configurable based on customer segment, customer centric (CIF based). This information will be configured in CMS and integrated with UI via API. The menu response should be cached on the server. App will preserve the previously fetched menu response and apply that until the new menu response is fetched on next app launch. App will have a default menu for the fallback, in case there are some errors in fetching

## **Sample Api Response:**

{

"menu": [

{

"id": 1,

"name": "Accounts",

"order": 1,

"enabled": true,

"sub\_menus": []

},

{

"id": 2,

"name": "Payments",

"order": 2,

"enabled": true,

"sub\_menus": [

{

"id": 3,

"name": "Pay Bills",

"order": 1,

"enabled": true

},

{

"id": 4,

"name": "Transfer Money",

"order": 2,

"enabled": false

},

{

"id": 5,

"name": "View Transactions",

"order": 3,

"enabled": true

}

]

},

{

"id": 6,

"name": "Settings",

"order": 3,

"enabled": true,

"sub\_menus": []

}

]

}

**Backend**: Manages complex logic to determine the enable/disable status of menu items based on global, segment, and user-specific flags.

**Frontend**: Receives a simple JSON structure with enabled flags, ensuring no changes are needed in the frontend application for dynamic menu rendering.

This ensures a seamless and efficient way to manage menu visibility without requiring frontend updates.

### **Backend Logic**

1. Determine Visibility:
   * Master Flag: Check the global enabled flag for the menu item.
   * User Segment Flag: If the global flag is enabled or disabled, check the user segment flag.
   * User-Specific Flag (CIF based): Finally, check if there is an override for the specific user.
2. Aggregate Results: The backend compiles the aggregate enabled/disabled status based on the above checks

# 32. Feature & Button Enablement

The application will dynamically manage the enabling and disabling of features and specific buttons using a combined JSON configuration structure. This configuration is fetched from the backend via an API call and applied to the frontend components. The JSON structure includes keys for feature-specific settings, each containing nested keys for button configurations.

**Combined Configuration JSON Structure:**

## Sample Api Response:

{

"features": {

"onboarding": {

"enabled": false,

"buttons": {

"register": false,

"skip": true

}

},

"login": {

"enabled": true,

"buttons": {

"login": true,

"forgotPassword": true

}

},

"profile": {

"enabled": true,

"buttons": {

"editProfile": true,

"changePassword": false

}

}

}

}

# 33. Glossary of Terms

It contains the definitions for terms in the main text that may be unfamiliar or unclear.

Table 6: Glossary of Terms

|  |  |
| --- | --- |
| Term/Acronym | Definition |
| API | Application Programing Interface |
| UI | User Interface |



Tata Consultancy Services is a purpose-led transformation partner to many of the world’s largest businesses. For more than 50 years, it has been collaborating with clients and communities to build a greater future through innovation and collective knowledge.  
TCS oﬀers an integrated portfolio of cognitive powered business, technology, and engineering services and solutions. The company’s 469,000 consultants in 46 countries help empower individuals, enterprises, and societies to build on belief.

Visit [**www.tcs.com**](file:///E:/Kapil%20Job%20Folder/2021/03%20Mar/Milind%20Arte/29%20Mar/www.tcs.com) and follow TCS news at [**@TCS\_News**](https://twitter.com/tcs_news?lang=en).

All content / information present here is the exclusive property of Tata Consultancy Services Limited (TCS).  
The content / information contained here is correct at the time of publishing. No material from here may be copied, modified, reproduced, republished, uploaded, transmitted, posted or distributed in any form without prior written permission from TCS. Unauthorized use of the content / information appearing here may violate copyright, trademark and other applicable laws, and could result in criminal or civil penalties.

**Copyright © 2024 Tata Consultancy Services Limited**