# Approval

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Function** | **Date** | **Signature** |
| Bhargavi Upadhya | System Architect |  |  |
|  |  |  |  |

# Revision History

| **Date** | **Revision number** | | **Status** | **Author** | **CR/PR ID** | **Changes/Comments** |
| --- | --- | --- | --- | --- | --- | --- |
| 2016-06-29 | 0.1 | Draft | | Ajay Das |  | First version |
| 2016-07-20 | 0.2 | Draft | | Deepthi Shivakumar |  | Added micro app framework section |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |

# Open Issues

The following open issues are identified:

| **Id** | **Category** | **Issue** | **Owner** | **Due date** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

# Table of Contents

[Approval 1](#_Toc456820414)

[Revision History 1](#_Toc456820415)

[Open Issues 1](#_Toc456820416)

[1 Table of Contents 2](#_Toc456820417)

[2 Document Introduction 4](#_Toc456820418)

[2.1 Purpose and Scope 4](#_Toc456820419)

[2.2 Intended Audience 4](#_Toc456820420)

[2.3 References 4](#_Toc456820421)

[2.4 Definitions, Acronyms and Abbreviations 4](#_Toc456820422)

[3 Product Overview 4](#_Toc456820423)

[3.1 Purpose and Intended Use 4](#_Toc456820424)

[3.2 Network Security 5](#_Toc456820425)

[3.3 Architectural Goals and constraints 5](#_Toc456820426)

[3.3.1 Goals 5](#_Toc456820427)

[3.3.2 Constraints 5](#_Toc456820428)

[4 Software Architecture 6](#_Toc456820429)

[4.1 Decomposition 8](#_Toc456820430)

[4.1.1 Software Item – UI State Manager 8](#_Toc456820431)

[4.1.2 Software Item – Event bus 9](#_Toc456820432)

[4.1.3 Software Item – Component interface. 9](#_Toc456820433)

[4.1.4 Software Item – Sync Manager 9](#_Toc456820434)

[4.1.5 Software Item – Philips UIKit 10](#_Toc456820435)

[4.1.6 Software Item – App Infra 10](#_Toc456820436)

[4.1.7 Software Item – BLE Manager ( BlueLib) 10](#_Toc456820437)

[4.1.8 Software Item – Data Model 11](#_Toc456820438)

[4.2 Common components 11](#_Toc456820439)

[4.3 Third party and open source software 11](#_Toc456820440)

[4.4 Data Flow 11](#_Toc456820441)

[4.4.1 Moment Data 11](#_Toc456820442)

[4.4.2 Data storage and cloud sync-up 12](#_Toc456820443)

[4.4.3 Configurable User interface. 12](#_Toc456820444)

[4.5 Development/Implementation view 15](#_Toc456820445)

[4.6 Process view 15](#_Toc456820446)

[4.7 Deployment view 15](#_Toc456820447)

[4.8 Use cases 15](#_Toc456820448)

[4.8.1 Retrieving stored sessions 15](#_Toc456820449)

[5 Micro app framework guidelines 15](#_Toc456820450)

[5.1 Configuration 15](#_Toc456820451)

[5.2 Launching mechanism 16](#_Toc456820452)

[5.2.1 Android 16](#_Toc456820453)

[5.2.2 IOS 17](#_Toc456820454)

[5.3 ActionBar or NavigationBar 17](#_Toc456820455)

[5.3.1 Android 17](#_Toc456820456)

[5.3.2 IOS 17](#_Toc456820457)

[5.4 Back key handling 17](#_Toc456820458)

[5.4.1 Android 17](#_Toc456820459)

[5.4.2 IOS 17](#_Toc456820460)

[5.5 Common interface or protocol 18](#_Toc456820461)

[5.5.1 Android 18](#_Toc456820462)

[5.5.2 IOS 18](#_Toc456820463)

[6 Appendix 18](#_Toc456820464)

[6.1 Storage considerations 18](#_Toc456820465)

# Document Introduction

The purpose of this document is to identify the architectural blocks and explain the top level design of App Framework for iOS and Android. The document provides an overview of the various functional blocks of the app framework, design patterns and the dynamic behavior of the framework. This forms a reference for detailed design and implementation.

## Purpose and Scope

This document describes the high level software architecture of the App Framework. The contents of this document apply to both the iOS and Android.

The interface and detailed design for each component is out of scope of this document.

Below are the core components of App Framework, App Framework is built for Android phones with version 4.4 and above and IOS 8.0 and above.

* Data
* Navigation and Logic
* Content
* Common Components
* Platform centric approach

## Intended Audience

CDP2 Architects and App Framework development team.

## References

| Reference | Identification | Title / additional remarks |
| --- | --- | --- |
| [REQUIREMENT] |  |  |
| [DATA\_MODEL] |  |  |
| [APP\_SIZE] |  | App size optimization version 0.3 |
| [INTERNATIONALIZATION] |  | App UI Internationalization, Version 1.0 |
| [CM\_PLAN] |  | SCM Plan |
| [RULES\_ENGINE] |  |  |

## Definitions, Acronyms and Abbreviations

| Term | Description |
| --- | --- |
| MVP | Model View Presenter |
| MVC | Model View Controller |
| UI | User Interface |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Product Overview

## Purpose and Intended Use

This document describes the high level software architecture of the App Framework. The contents of this document apply to both the iOS and Android .

## Network Security

to be updated.

## Architectural Goals and constraints

### Goals

#### Extensibility

The App Framework is intended as a platform for all next CDP2 propositions. As such the App Framework needs to be extensible with sensors, algorithms and user interfaces relevant for upcoming propositions whilst keeping functionality of micro apps. This is done by making use of state machine for user interface and plugins with respect to common components.

#### Compatibility

The App Framework should build modular UI components to handle upgrades and extensions. UI interfaces should be developed adopting the responsive design, density independent layouts and constraints based interface principle in order to support current and future smartphones/tablets with different properties.

#### Design rules

The design uses Object Oriented concepts which are supported by iOS and Android. The design patterns like State and Observer are used.The app framework follows the layered approach.

User Interface is built from different micro components, each user interface component or micro app in App Framework exhibits different behavior based on the state of the user interface. State manager manages all the states in the App flow. App flow consist of a set of states. App flow is a transition from one state to another. Each state holds a navigator and a logic.

Component interface provides the plugin mechanism to add a common component to App Framework. It provides the basic interface that all common components will have to adhere to plugin in to App Framework. The construction of the common component is done by the factory which returns the correct component type based on configuration.

Event bus handles communication between components and micro apps. Event bus simplifies the communication between components. It decouples event senders and receivers and performs well with Activities, Fragments, and background threads. Event bus avoids complex and error-prone dependencies and life cycle issues and makes code simpler

### Constraints

#### Device constraints

App Framework is optimized for smartphones and compatible to tablets. The below table lists various device constrains in App design and development.

|  |  |  |
| --- | --- | --- |
| Constraints | Risk | Mitigation |
| OS version | App incompatible with new versions of OS  - inconsistent style  - incompatible rights  - | Minimum OS requirement  iOS>=9  Android >=4.4    Performance:  Check on first use/update if real time requirements can be met. Should guard for incorrect results, yet users are likely unhappy if their mobile is stated as unsupported. |
| Screen size | Incorrect rendering  Incorrect use of real screen estate | Responsive layout  Test on range of products |
| Devices | App incompatible with must support devices. | iOS: iPhone 5 and above  Android: Huge device diversity. Test on hero list of devices and with different device + OS version combination. |
| Performance | Sluggish UI behavior | The response to the user actions should be completed in less than the human perceivable duration (Typically 200 msec). Operation that take more time (Ex: network operations) should not be executed in UI/Main thread. |
| App size | Huge app size | The guidelines provided in [APP\_SIZE] shall be enforced. |
| Localization/Internationalization | App becomes inflexible to add new language or support more countries. | The guidelines provided in [INTERNATIONALIZATION] |
| Security | The data stored in the app can be viewed by hackers. | Encrypt the data stored locally in the app using the guidelines provided in TBD. |

# Software Architecture

The purpose of the app framework is to provide a quick starting point for application development. The app framework is a base application that pre-integrates all CDP2 common components. The app framework provides UX concepts that are common to most applications. The app framework provides UX components that can be configured by proposition to build their own UI flows.

The app framework guarantees that the combination of integrated common component versions functions properly. Although the app framework delivers a functional application, it is the proposition’s responsibility to deliver an application that meets the intended user requirements. This can be achieved by configuring the framework, removing unrequired components, and adding proposition specific logic to the app framework.

The following block diagram provides a high-level overview of the functional components in the app framework



Figure : Block Diagram App Framework

The core functionality of app framework is to provide a quick starting point for application development. The app framework is a base application that pre-integrates all CDP2 common components. The App Framework is defined to have the below functional areas

* Data model.
* Modular / configurable user interface.
* Pluggable approach to common components.
* Dependency management of common components.
* Common configuration.



Figure : App Framework Function areas

The following diagram illustrates the top level architectural view of App Framework with interfaces to external system.



Figure 3: Top level architecture

## Decomposition

The following section explains the important building blocks of App Framework.

### Software Item – UI State Manager

**Main responsibility:**

User Interface consists of the Model, view and the controller/Presenter associated with the view. The User Interface sends user actions to the controller/presenter. The user interface works on the principle of state machine. UX flows are composed of states and UI navigation is state transitions.

State Manager handles state transitions across the application

* Micro app request a new state from the UI state manager.
* Micro app requests UI state manager to navigate to a particular state.
* UI state manager navigates based on navigation rules defined in State machine.

UI works on state change mechanism where the User Interface updates from the UI state Manager.

**Constraints:**

* Navigation rules based on view elements are decided by the controller / presenter.

### Software Item – Event bus

**Main responsibility:**

Event bus handles communication between components and micro apps. Event bus simplifies the communication between components. It decouples event senders and receivers and performs well with Activities, Fragments, and background threads. Event bus avoids complex and error-prone dependencies and life cycle issues and makes code simpler.

Event bus handles data transfer across the micro application

* Micro app sends model data to other micro apps through event bus.

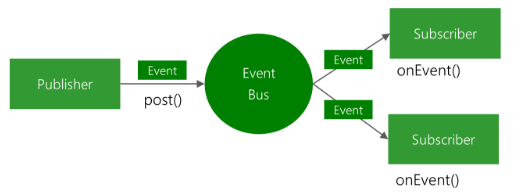


Figure 4: Event bus

### Software Item – Component interface.

**Main responsibility:**

Component interface provides the plugin mechanism to add a common component to app framework. It provides the basic interface that all common components will have to adhere to plugin in to app framework. The construction of the common component is done by the factory which returns the correct component type based on configuration.

Component interface provides a common interface to plugin any common component in App Framework.

* All common components that are plugged in to IPL lumea app will implement the interface.
* The component are provided by the component factory.
* The component to be created by the factory can be configured through a config file.

**Constraints:**

* Adding and removing components from the Application requires resolving build dependancies.

### Software Item – Sync Manager

**Main responsibility:**

The Sync Manager synchronizes the data stored locally in the app with cloud. It interacts with data core to synchronize data. Rest APIs provided by Data Core will be used to sync to Data Core. The data synced is linked to the user's profile. With this the data sync provides the following features:

* Sync data of a user.
* Recover from database corruption, app uninstallation and any other errors.
* Reflect the update in data.

The trigger for synchronization comes from the following events.

* App start or wake up from standby
* Data insertion in local storage
* Change in network

**Constraints:**

* The Sync Manager runs in its own thread to ensure that the UI is not blocked during its operation.
* In order to reduce the network traffic Data Sync would perform bulk synchronization.

### Software Item – Philips UIKit

**Main responsibility:**

The UIKit provides a large set of UI primitives like: buttons, form fields, toggles, action bar, tab bar, etc. These primitives are to be used to build the UI of an application. The use of these primitives’ guarantees that the visual appearance and behavior achieved is in accordance with the brand design guidelines.

UIkit provides .

* The app theme and Ux elements that is used in app makes use of UIkit elements.

**Constraints:**

* Themes specified in UIkit can only be used in IPL lumea app.

### Software Item – App Infra

**Main responsibility:**

App infra provides functionality that is common for most propositions and provides a base layer for optimal app development.

App Infra .

* Provides logging / tagging / secure storage that can be used by application.

**Constraints:**

* App Infra has to be created in the app contect and dependency be injected in to common components.

### Software Item – BLE Manager ( BlueLib)

**Main responsibility:**

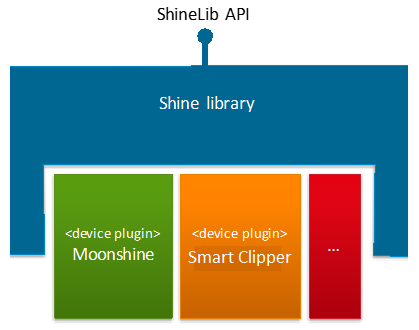
Handles connectivity, BLE manger provides the interface for Bluetooth connectivity to a connected device. Different BLE devices typically expose their functionality by a set of significantly different BLE services. For example, a BLE weight scale exposes the Weight Scale Profile. The BlueLib library is not intended to be monolithic and consequently hard to extend. Instead, Bluelib library supports different BLE devices by means of so-called device plugins. A device plugin contains the device-specific implementation of various capabilities .

The main functionalities are:

* BLE communication with devices.

**Constraints:**

* Plugin has to be written for every new BLE device that has to be added.
* Distributed as source.



### Software Item – Data Model

App Framework provides a data model to communicate between UI elements. A data model is an abstract model that organizes elements of data. The data model can been seen as a wrapper for data that is passed around between micro apps, the content of the data model is app specific.

Data in model context is the representation of database contents to the User Interface. The following represents the data context.

|  |  |
| --- | --- |
| Data | Context |
| User Profile | Holds details on user profile , specifically user login data and other specific user details required by app. |
| Content | Consists of articles, cards and multimedia (Images and Videos) |

## Common components

The App Framework pre integrates the following common (or horizontal) app components developed in CDP2.

|  |  |
| --- | --- |
| Common component | Description |
| User Registration | The User Registration component provides a standard user experience for the user registration flow including social login. The App framework integrates this component which abstracts the interaction towards the HSDP identity management service and various SDKs for social login. |
| BlueLib | BlueLib handles the communication between App and Bluetooth device. The services and characteristics specific to device are implemented using BlueLib plugin. |
| InApp Purchase | eCommerce is the component that provides interface to purchase products, services and accessories via mobile app. It provides the User Interface, connection to payment gateway and store. |
| Philips UI Kit | Philips UI Kit component provides the interface to create UI elements that are complaint with Philips design guidelines for Mobile Apps. |
| DI-COMM | DI-COMM library is a component that manages the app to device communication. |
| Consumer Care | The consumer care component is an off the shelf component providing consumer care functionality |
| Product Registration | The Product Registration component provides a standard user experience to register product. |
| App Infra | App infra provides functionality that is common for most propositions and provides a base layer for optimal app development. App Infra provides logging , tagging , secure storage , service discovery and many other functions. |
| Moments DB | Database with moments. Will be developed as a common component that can work with moments. |

## Third party and open source software

A list of all used third party software and relevant licenses is described in the configuration management plan [CM\_PLAN]

## Data Flow

### Moment Data

**Introduction:**

Any module that handles data should interacts with Sync Manager for storing information. The User Interface subscribes for the events from Sync Management component.

The communication between the app and the Sync manager is as described below.

Application defines the data to be send to sync managar.The application converts its data to a moments data. The moments data is passed to sync manager.The sync manager stores the data locally in the local data base, The sync manager communicates with data core using a rest client to push data to data core.

### Data storage and cloud sync-up

All data obtained from application is persisted in the app. The backend sync module syncs the data from peristence to cloud. The data associated with user is stored in Data Core backend.



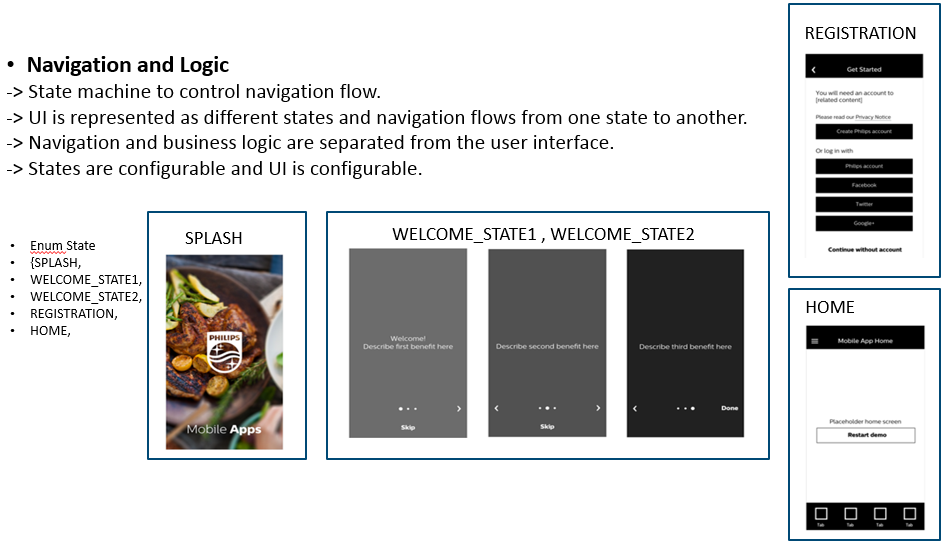
Figure : Data storage and cloud sync up

### Configurable User interface.

The User interface is build on the concept of micro app. The micro app boundaries are well defined.The microapps do not communicate with each other. The microapps communicate through the flow manager.

The flow manager manages all the flow in the app.Navigation across micrapps is based on the concept of state machine.

Each micro app decides the next state to navigate. The flow manager creates the state and navigates the app to the next state as defined by the microapp.



1. UIFlowmanager holds all the states in the flow, A particular flow in the app is managed by the flow manger, A flow is a transition across the states.
2. UIStateBase is the base class for the UIState.
3. UIState is the container that holds the state, It holds the navigation and logic for UIview.
4. UIBaseNavigation is the interface for navigation logic. It abstracts all the navigation rules in a view.
5. UINavigation implements the navigation logic for a particular state of the UIview.
6. UILogic holds the presentation logic for a UIview in a particular state.
7. Activity map is the map of state and corresponding Activities.
8. UIStateDef is the enum of All states in the flow.

The figure below shows the class diagram of Configurable UI.

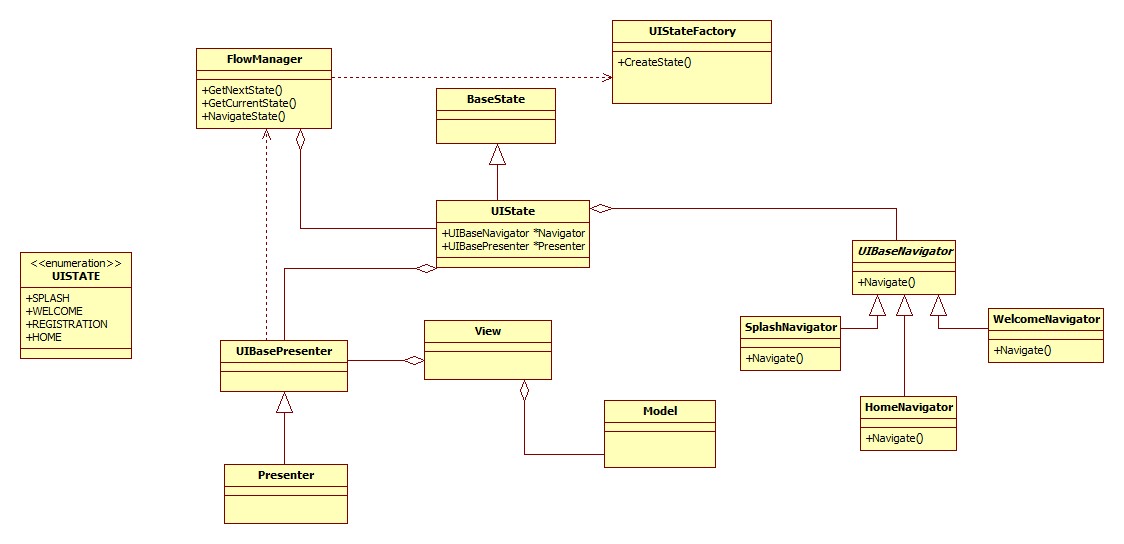


Figure : Configurable UI design

Component Interface is the interface that has to be implemented to plug in a common component to app framework.

Component Factory is the factory that produces the common component based on app framework configurations.

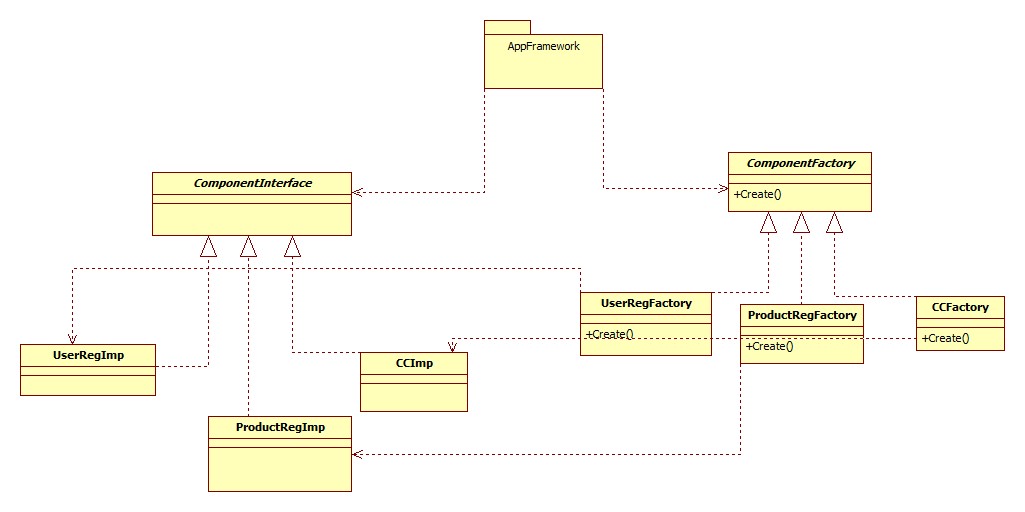


Figure 9: common component plugin

## Development/Implementation view

Refer to Configuration Management document [CM\_PLAN].

## Process view

The primary requirement from User Interface is that it should never be blocked and always responsive. For the functions that require communication with cloud and device, platform provided mechanism is to be used to ensure that the UI thread is not blocked.

While the operating system supports concurrent mechanism of accessing and modification of Model data (Persistence), it would complicate the implementation and debugging would be hard. In order to streamline the mechanism, the Model data is accessed and modified in one thread only.

In case of database corruption, which is indicated the platform APIs by specific error, the app will recreate database and sync with cloud.

## Deployment view

The App Framework is delivered as source. Vertical apps based on the framework will be deployed via Apple App Store and Google Play Store.

## Use cases

### Retrieving stored sessions

Retrieving of stored sessions is performed whenever App is reinstalled on a different device. It happens without involvement of the user. Once the sessions are retrieved, user can view them in the User Interface.

The exceptions that are not handled in the flow are:

* App closed forcibly by user
  + Solution: Undefined. Any database corruption will result in a cleanup when app is initiated again.

# Micro app framework guidelines

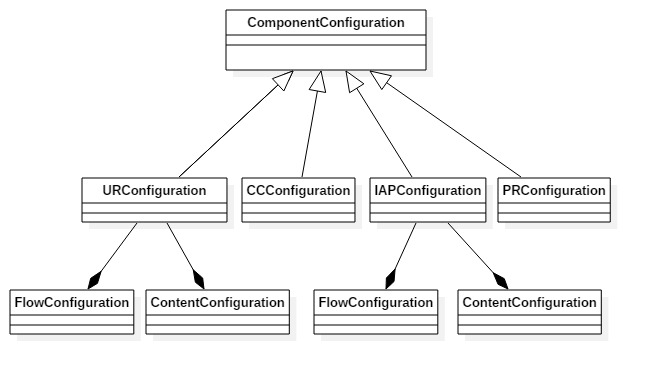
This section is about the set of interfaces or base classes or guidelines that each micro app framework should follow comply with.

## Configuration

Each micro app creates container of different set of different configurations which may be key value pair with setters and getters or group them into relevant classes with APIs to set and get configuration. The structure of configuration items is very much component specific but as per the guideline, configuration class exposed by component should extend from one base class provided by micro app framework as depicted in below diagram.

Below diagram depicts sample structure of configuration classes from micro apps.

Micro app (component) framework is responsible for creating base class.

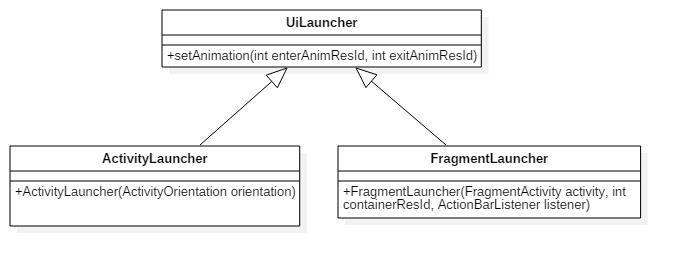


## Launching mechanism

### Android

An app can launch micro app either as activity or fragment. Each micro app need to facilitate both kind of launching mechanism. Micro app (component) framework enables this by providing set of base classes as shown in below diagram.

App or any component which launches a micro app is expected to create an object of either Fragment launcher or Activity launcher and inject to micro app launch API which is described in below sections.



Here ActivityOrientation is a nested class inside ActivityLauncher with set of defined static constants of various screen orientation types supported by Android.

Refer: <https://developer.android.com/reference/android/content/pm/ActivityInfo.html#screenOrientation>

### IOS

In case of IOS, each micro app is expected to return view controller which needs to be launched by the app.

Micro app framework defines standard launch API as part of common protocol which needs to be implemented by each micro app as described in below sections.

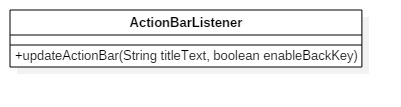
Also as part of launch API, we need to have completion handler with error block returning NSError type.

## ActionBar or NavigationBar

### Android

If a micro app is launched as activity action bar is handled by micro app itself. If micro app is launched as fragment, then there should be standard mechanism to handle action bar events like updating title , back key etc.

Micro app framework defines the below interface which needs to be implemented by base app or any vertical app and inject as a parameter of launcher API of micro app as defined in Launcher section.



**Note: Custom view in action bar is yet to be standardized.**

### IOS

In case of IOS, micro app can access Navigation controller there by it can update Navigation bar but if an app has custom view, it is not standardized yet .

Each micro app is also expected not to have its own Navigation controller.

## Back key handling

### Android

If a micro app is launched as activity back key is handled by micro app itself. If micro app is launched as fragment, then there should be standard mechanism to handle back key events both on action bar or hard back key press.

Micro app framework defines an interface like below and each micro app is expected to implement this interface. Micro app needs to return true if they need to consume and handle back key event else return false if they do not want to any actions there by app calls super.onBackPressed().



### IOS

In case of IOS, there is no need of an interface or class to handle back key events by micro app.

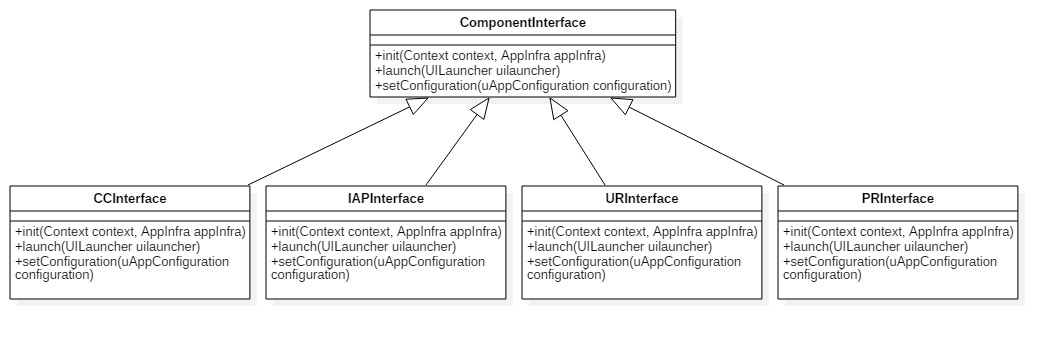
Micro app can access Navigation stack and modify as per the requirement.

## Common interface or protocol

### Android

Micro app framework provides below interface which has standard APIs to initialise, launch and set configuration.

Each micro app will expose one single class for vertical to launch UI where in it needs to implement this interface.



**Note:**

1. **The names of classes may be subjected to change.**
2. **Init API signature may also change.**

### IOS

Micro app framework provides below protocol which has standard APIs to initialise, launch and set configuration.

Each micro app will expose one single class for vertical to launch UI where in it needs to implement this interface.

-(void)initWithConfiguration:(ComponentConfiguration \*)configuration;

-(UIViewController \*)invokeComponent:(void (^)(NSError \*error))completionHandler;

Note: App infra’s injection is yet to be decided.

# Appendix

## Storage considerations

The app generates lot of contents during its lifetime. The following table indicates how long they need to be stored in the sandbox of the app.

|  |  |
| --- | --- |
| Data | Storage period |
| User profile and other user data. | data is stored in internal DB and synced with Data Core. |
| Contents | Maximum cache size will be set to 50MB. Least used contents will be removed once the threshold is crossed. |