

System Design Description

<Wipro Technologies> [Global Mobile Sales Platform]

|  |  |
| --- | --- |
| **Version Number:** | 1.0 |
| **Draft/Final as of:** | 10/02/2015 |
| **Printed on:** | [Date] |
| **Author** | Chella,  Mobile Solution Architect  Wipro Technologies, Bangalore  chella.pandeeswari@wipro.com |
| **Owner** | MetLife |

|  |  |
| --- | --- |
| **Approval Signatures** | |
| Name | Title |
| Komal | Architect |
|  |  |

|  |  |
| --- | --- |
|  |  |

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Reviewed By** | **Change Description** |
| 10/02/2015 | 1.0 | Chella | Komal | Initial version |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

1 Scope 4

1.1 System Design Definition Overview 4

1.2 Intended Audience 4

1.3 Identification 4

1.4 References 4

2 System Architectural Design 5

2.1 System Architecture 5

2.2 Application Logical Architecture 6

2.3 Environment (Dev, QAT, UAT, Production) 6

2.3.1 System Hardware Environment and Dependencies 6

2.3.2 System Software Environment and Dependencies 7

2.4 System Integration 7

2.5 Third Party Software 8

3 Interface Definition 8

3.1 Interface Standards 8

3.1.1 Standards for Global Mobile Sales App Webservice calls 8

3.1.2 Request Header Information 8

3.1.3 Response Error Information 10

4 Processes 11

4.1 Global Components 11

4.1.1 File Transfer 11

4.1.2 Embedded Calculator Integration 13

4.1.3 Sample Web service creation 14

4.1.4 Sample Web service invocation through Worklight adapter 14

4.1.5 JSON storage library 19

4.1.6 Capture Signature 21

4.1.7 UI configurator 21

4.1.8 Multi-language support 28

4.1.9 Graph Control 30

4.1.10 PDF generator 30

4.2 User Login 30

4.2.1 User Authentication 30

5 Glossary and Acronyms 31

# Scope

## System Design Definition Overview

A System Design Definition (SDD) is a document containing detailed information about the design of a specific customer application solution. This document will describe all aspects of the system that will be created. This SDD may include some or all of the following: Architecture Diagram, process flow diagram, third party software, content repositories, software interfaces, processing logic, error handling, security, and general design considerations for requirements specified in the System Requirements Definition (SRD).

## Intended Audience

This document is intended for the following project team members:

**Architects** – Used to articulate customer and architect expectations on how approved requirements will be implemented.

**Development Leads** – Used to create Software Design Document and develop the system.

**Developers** – Used to develop the system

## Identification

A major-minor-patch-build identification number (e.g. version w.x.y.z) will be used to track the completion of project. W refers to the major release. X refers to the minor release, Y refers to patch release, Z refers to build. During development the build release will increment with each build to QA.

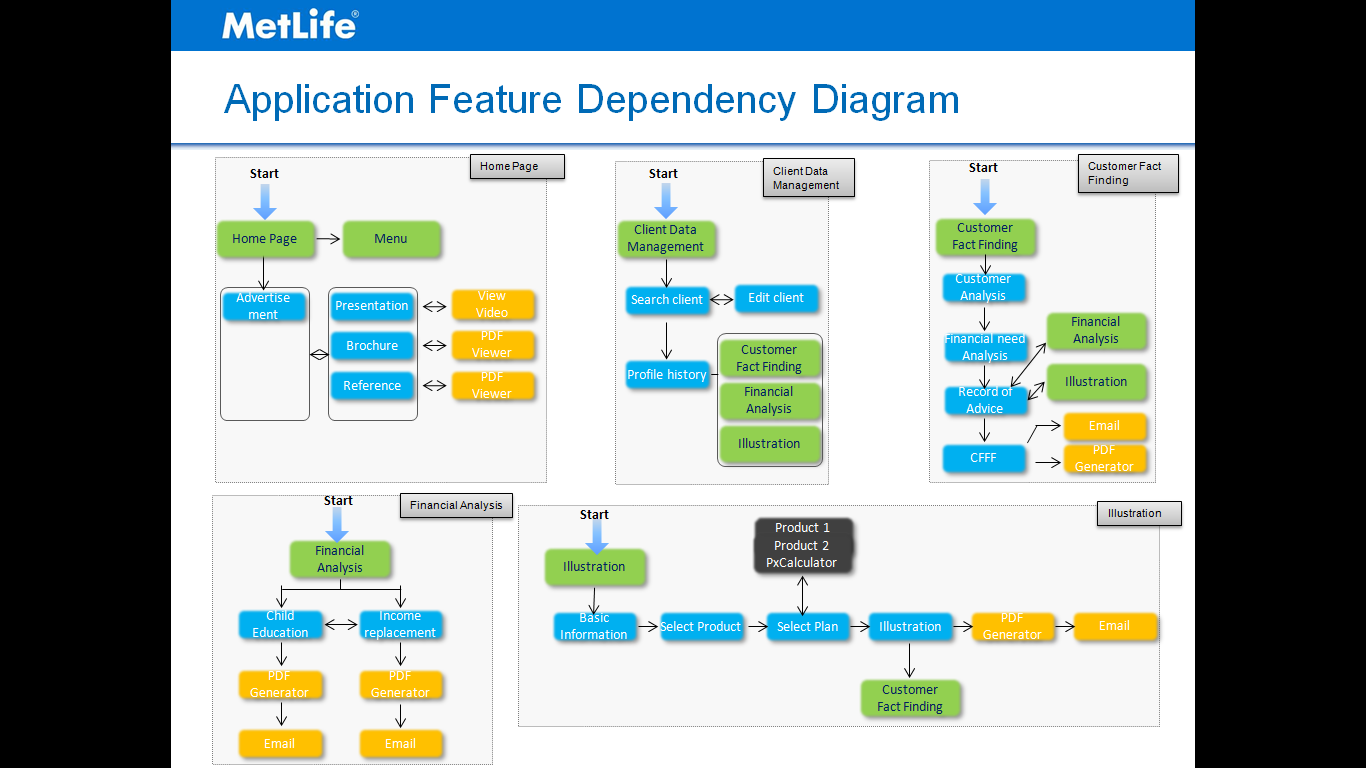
The entire system produced by this project should be referred to as the [Global Mobile Sales Platform] version *[major, minor]*.

## References

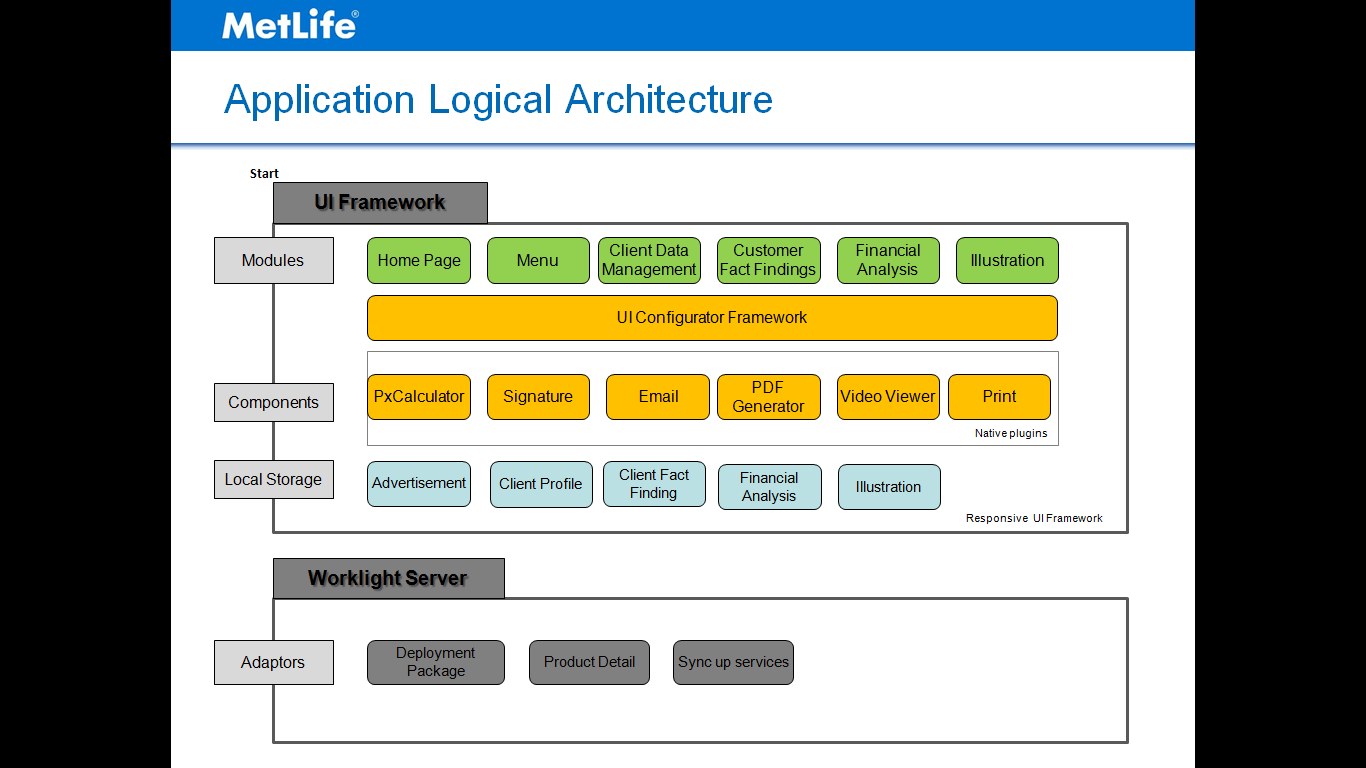
|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | **Author** | **Version** | **Date** |
| MetLife- GSP SRD | Mahadevan | As SRD is not finalized, version details will be filled later |  |
| PX\_Calculator\_Specification.docx | HP |  |  |
| ProductExpress Deployment Pkg.zip | HP |  |  |
| Interface Specification Document | Komal |  |  |

# System Architectural Design

## System Architecture



## Application Logical Architecture



## Environment (Dev, QAT, UAT, Production)

### System Hardware Environment and Dependencies

Hardwares for Dev:

Mac machine / Windows Machine

Hardwares for QAT:

Devices (Android screen size above 8 inch devices)

Windows Machine

Hardwares for UAT:

<to be provided by MetLife>

Hardwares for Production:

<to be provided by MetLife>

### System Software Environment and Dependencies

Softwares for Dev:

1. Eclipse-juno classic
2. Android OS-Android 4.2 and above
3. XCode 5 and above
4. iOS- 8 and above
5. Cordova-4.2.0
6. AngularJs-1.3.13
7. MobileFirst Platform (worklight)-6.1.0.02
8. Svn for Source Control Management
9. D3 JavaScript library to create graphs
10. Worklight Server access from MetLife

Softwares for QAT:

1. Android 4.2 and above
2. Software to capture the bugs
3. Worklight Server access from MetLife

Softwares for UAT:

<to be provided by MetLife>

Softwares for Production:

<to be provided by MetLife>

## System Integration

HP Embedded Calculator will be integrated with Global Mobile Sales platform to provide the flexibility to do the Sales illustration working offline.

ProductExpress Deployment Pkg.zip

* PX\_Calculator\_Specification.docx and
* PX Test App will be the reference for integrating the Embedded Calculator in this app.

## Third Party Software

1. Cordova plug in for File Transfer

Reference URL:

<https://github.com/apache/cordova-plugin-file-transfer/blob/master/doc/index.md>

1. Cordova FileSystem API along with crypto-js javascript secure algorithm for File encryption and decryption

<http://www.html5rocks.com/en/tutorials/file/filesystem/>

<https://code.google.com/p/crypto-js/>

# Interface Definition

## Interface Standards

### Standards for Global Mobile Sales App Webservice calls

Following are the standard list of parameters that must be included in all webservice calls.

* Header information
  1. Must be included in all Webservices request
* Footer information
  1. Must be included in all Webservices response
* Exception to above rule
  1. It will be clearly marked in the corresponding Webservice detail section (Webservice Detail section overrides the parameter definition included in this section
* No information in HTTP headers

Refer “Interface Specification” document for more information on this.

### Request Header Information

| **Request**  **Parameter** | **Required** | **Type** | **Request Parameter Description** | **Valid Values** |
| --- | --- | --- | --- | --- |
| tenantCode | Y | String | Tenant code details. This is always same for all requests. | Constant value set to “DC.HKG.SALES.<tenant\_Name>” |
| locale | Y | String | This represents the language in which this Webservices should send the response in | If Language is ENGLISH  locale = “en\_US”  If Language is VIETNAMESE  Locale = “vn\_VN” |
| authenticationToken | Y | String | This parameter is used to track if this user is already authenticated. This ensures that once authenticated each subsequent service request need not re-validate the user  GSP application created this Authentication token as part of response to “Authentication” service response.  The calling application is expected to manage the same and include it in the subsequent Webservices request parameter  Validity  TTL for the Token will be defined and will get reset at every Successful service call  Note: GSP is not doing session management for MOS | Randomly assigned security token |
| guid | Y | String | This is random number generated to keep track of transaction across Front end and Back end.  Requesting application will generate and send it as part of request | Randomly assigned 16 digit HEX Number |
| userId | Y | String | User ID of the logged in user |  |
| deviceId | Y | String | Device ID which is associated with the user.  Calling application determines this and sends it as part of all service requests  Note: This information is planned to be validated against information stored in Active Directory (One to one relation b/w Device ID and User ID) |  |
| sourceType | Y | String | Indicates the origin of the request | Should always be “mobile” for MOS |

**Sample request with header information:**

{

"transaction": {

"header": {

"tenantCode": "DC.HKG.SALES.VNM",

"locale": "en\_US",

"authenticationToken": "HEXA0000123456",

"guid": "HEX0000000001111",

"userId": "admin",

"deviceId": "macid13",

"sourceType": "mobile"

},

"type": "illustrationId",

"parameters": {

"productCode": "UL",

"planCode": "ULA1"

}

}

}

@Komal: Please confirm the value for "tenantCode": "DC.HKG.SALES.VNM",

### Response Error Information

Following standard data field will be included in the error scenario response body (Standard header information will be included as defined above.

| **Response Parameter** | **Required** | **Response Parameter Description** | **Valid Value** |
| --- | --- | --- | --- |
| errorCode | N | This tag is mandatory only if responseStatus = “ERR”  Standard Format will be followed which will be as follows  **Format: ERR**MNN   * ERR – Prefix code stands for error * M – Major error code. There are five major error code –   + 1 – JSON Payload/ Information in JSON related   + 2 – Data Validation related (Mandatory etc..)   + 3 – Business Rule validation failed   + 4 – FILE attachments related   + 5 – Other unrecoverable exception * NN – Minor error code. There will be various possible value for minor code returned. * Example of error code: ERR101, ERR204 |  |
| errorMessage | N | This tag is mandatory only if responseStatus = “ERR”  Brief description of the error that caused the rest service call failure |  |

**Sample response with error information:**

{

"transaction": {

"header": {

"tenantCode": "DC.HKG.SALES.VNM",

"locale": "en\_US",

"guid": "HEX0000000001111",

"sourceType": "mobile",

"responseStatus": "ERR"

},

"type": "authenticate",

"parameters": {

"errorCode": "ERR101",

"errorMessage ": " Invalid JSON request - JSON Structure mismatch found "

}

}

}

# Processes

## Global Components

### File Transfer

* The files which are downloaded from the server will be stored in the mobile app sandbox.
* To transfer any type of file from server to mobile app, FileTransfer cordova plugin will be used.
* FileTransfer API helps to download one particular file at a time.
* We can create a webservice with the url https://int.sales.MetLife.com.<tenantname>/gsp/rest/common/downloadfile and with the below major parameters:

filesource

filedestination

* To download the files securely from the server, authentication token, guid, device id etc will be passed in the request header like other webservice calls.
* Source and Destination needs to passed as parameters for the FileTransfer API.
* FileTransfer API gives the way to show the progress bar during each file download.
* Onprogress property will be called with a ProgressEvent whenever a new chunk of data is transferred.

var ft = new FileTransfer();

ft.onprogress = function(progressEvent) {

if (progressEvent.lengthComputable) {

loadingStatus.setPercentage(progressEvent.loaded / progressEvent.total);

} else {

loadingStatus.increment();

}

};

* Sample code for FileTransfer:

**var** app = {

       // Application Constructor

       initialize : **function**() {

**this**.bindEvents();

       },

       // Bind Event Listeners

       //

       // Bind any events that are required on startup. Common events are:

       // 'load', 'deviceready', 'offline', and 'online'.

       bindEvents : **function**() {

              document.addEventListener('deviceready', **this**.onDeviceReady,**false**);

       },

       onDeviceReady : **function**() {

**var** sourcePath = "http://10.207.52.24:8080/test/video.mkv";

              app.fileDownload(sourcePath);

       },

       fileDownload : **function**(path) {

**var** ft = **new** FileTransfer();

              ft.download(path, "/storage/emulated/0/.video.mkv",

**function**(entry) {

                     alert("success");

              }, **function**(err) {

                     alert(JSON.stringify(err));

              });

       }

};

app.initialize();

* Reference URL:

<https://github.com/apache/cordova-plugin-file-transfer/blob/master/doc/index.md>

### Embedded Calculator Integration

* Include the below PX Embeded Calcualtor Runtime environment and other Utility jar files under libs folder of Global Mobile sales app project structure.

a.FiaJNI.jar

b.pxjavaruntime.jar

c.PxUtils.jar

d.PxVal.jar

=> Total jars size is 64 KB

* Include the below native libraries under \libs\armeabi of Global Mobile sales app project structure.

a.libCalculatorJNI.so

b.libexslt.so

c.libiconv.so

d.libxml2.so

e.libxslt.so

=> Total Native size is 24 MB

* Total Size of the Jars + Native Libraries is around 24.64 MB approx. All the distributables are necessary and PX system cannot execute with anything missing.
* Enable “Write External Storage Permission” in "AndroidManifest.XML".
* Sample code for loading the deployment package using the Embedded Calculator

PxCalculatorHome calculatorHome = PxCalculatorHomeJNI.instance();

calculatorHome.initialize(getProductXpressInstallPath());

calculatorHome.loadDeploymentPackage("TestInput/MyDeployment2\_1\_0.pxdp", null, null, null);

PxPushCalculator calculator = calculatorHome.getPushCalculator();

String request;

// Construct the request string

String result = calculator.calculate(request);

* Refer “PX\_Calculator\_Specification.docx” for Request and Response object specification.
* Android sample app (PxTestApp) will be wrapped as a Javascript custom plug-in so that this can be integrated in Global mobile sales app. This wrapper will support only Android as of now.
* For security reasons, the deployment package with “.pxdpz” file will be used as this is in encrypted format. MetLife key is required for the decryption. [Decryption mechanism needs to be checked with Embedded Calculator HP team]
* Suppose if the pxdpz file format does not work out, then the security can be applied by hiding the Embedded calculator pxdp file from the app sandbox. This can be easily achieved by added a dot in front of the filename.

### Sample Web service creation

* Below are the steps tried out to create a web service which returns the results based on the JSON file.
* The sample webservice reads the static JSON file and returns the JSON object to the called adapter.
* Sample code snippet explains the server side implementation.

JSONParser parser = **new** JSONParser();

**try** {

Object obj = parser.parse(**new** FileReader("C:/test/productlist.json"));

JSONObject jsonObject = (JSONObject) obj;

response.setContentType("application/json");

response.getWriter().print(jsonObject.toString());

System.*out*.println(jsonObject.toString());

}

**catch** (Exception e)

{

e.printStackTrace();

}

### Sample Web service invocation through Worklight adapter

* Worklight adapter is used to invoke the sample web services which returns the static JSON data.
* Steps to create worklight adapter:
* Click on the Worklight icon  cid:8e8df0f2-ab48-4538-9602-d61d026ec74f pulldown from the toolbar and select Worklight Adapter (or)

File => New Worklight Adapter from the menu bar

* + - Enter the project name
    - Choose Adapter type like HTTP Adapter, SQL Adapter
    - Enter name of the adapter
    - Click Finish
* Created adapter will have .xml and –impl.js files wherein the web service and its associated methods need to be configured.
* Below are the adapter files for ProductList sample web service.
* PRODUCT\_LIST.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!--

Licensed Materials - Property of IBM

5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or

disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

-->

<wl:adapter name=*"PRODUCT\_LIST"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:wl=*"http://www.worklight.com/integration"*

xmlns:http=*"http://www.worklight.com/integration/http"*>

<displayName>PRODUCT\_LIST</displayName>

<description>PRODUCT\_LIST</description>

<connectivity>

<connectionPolicy xsi:type=*"http:HTTPConnectionPolicyType"*>

<protocol>http</protocol>

<domain>10.207.52.16</domain>

<port>8080</port>

<!-- Following properties used by adapter's key manager for choosing specific certificate from key store

<sslCertificateAlias></sslCertificateAlias>

<sslCertificatePassword></sslCertificatePassword>

-->

</connectionPolicy>

<loadConstraints maxConcurrentConnectionsPerNode=*"2"* />

</connectivity>

<procedure name=*"getProductDeploymentPackageList"*/>

</wl:adapter>

* PRODUCT\_LIST-impl.js

**function** getProductDeploymentPackageList()

{

path = getPath();

**var** input = {

method : 'get',

returnedContentType : 'json',

path : path

};

**return** WL.Server.invokeHttp(input);

}

**function** getPath(interest)

{

**return** '/test/Testservice';

**}**

* Once the adapter is created, it can be invoked from JavaScript code.
* Sample code snippet to invoke the above sample product list adapter

**var** input = {

adapter : 'PRODUCT\_LIST',

procedure : '*getProductDeploymentPackageList*',

parameters : []

};

WL.Client.invokeProcedure(input, {

onSuccess : loadSQLQueerySuccess,

onFailure : loadSQLQueeryFailure

});

* We will get the response json object from web service through adapter on success callback of the previous code.
* Below is the sample json response returned for *getProductDeploymentPackageList* adapter method
* If any additional wrapper class is required, please provide the details to create the wrapper class.

{

"isSuccessful": true,

"responseHeaders": {

"Content-Length": "2324",

"Content-Type": "application\/json",

"Date": "Wed, 11 Feb 2015 13:23:14 GMT",

"Server": "Apache-Coyote\/1.1"

},

"responseTime": 31,

"statusCode": 200,

"statusReason": "OK",

"totalTime": 31,

"transaction": {

"header": {

"guid": "HEX0000000001111",

"locale": "en\_US",

"responseStatus": "OK",

"sourceType": "mobile",

"tenantCode": "DC.HKG.SALES.<Tanent Name>"

},

"parameters": {

"products": [

{

"productCode": "UL",

"productName": "Universal Life",

"productURI": "http:\/\/www.example.org\/UL",

"productVersion": "0.8",

"subProducts": [

{

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageDeploymentDate": "02-04-2014",

"subDeploymentPackageName": "AmLife Lifestyle\_0\_33\_14\_0",

"subDeploymentPackageVersion": "1.9",

"subMarketingName": "AmLifestyle",

"subProductCode": " ULLA5 ",

"subProductName": "AmMetLife Lifestyle"

},

{

" subDeploymentPackageDeploymentDate ": "02-04-2011",

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageName": "AmLife Link\_0\_31\_15\_0",

"subDeploymentPackageVersion": "0.61",

"subMarketingName": "AmLink",

"subProductCode": "ULRP6",

"subProductName": "AmMetLife Link"

}

]

},

{

"productCode": "EN",

"productName": "Endowment Product",

"productURI": "http:\/\/www.example.org\/TP",

"productVersion": "0.8",

"subProducts": [

{

" subDeploymentPackageDeploymentDate ": "02-04-2015",

"DeploymentPackageType": "pxdpz",

"plans": [

{

"marketingName": "Plan1",

"planCode": "ALSB1",

"planName": "AmMetLife SecureBuilder "

},

{

"marketingName": "Plan2",

"planCode": "ALSB2",

"planName": " AmMetLife SecureBuilder "

},

{

"marketingName": "Plan3",

"planCode": "ALSB3",

"planName": " AmMetLife SecureBuilder "

},

{

"marketingName": "Plan4",

"planCode": "ALSB4",

"planName": " AmMetLife SecureBuilder "

}

],

"subDeploymentPackageName": "AmLife SecureBuilder\_0\_44\_22\_0.pxdpz",

"subDeploymentPackageVersion": "0.81",

"subMarketingName": "AmMetLife SecureBuilder",

"subProductCode": "ALSB",

"subProductName": "AmMetLife SecureBuilder "

},

{

" subDeploymentPackageDeploymentDate ": "08-04-2015",

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageName": "AmLife SecureGuard Plus\_0\_43\_25\_0",

"subDeploymentPackageVersion": "0.85",

"subMarketingName": "AmMetLife SecureGuard Plus",

"subProductCode": "BTASGP1EPA",

"subProductName": "AmMetLife SecureGuard Plus"

},

{

" subDeploymentPackageDeploymentDate ": "04-04-2015",

"DeploymentPackageType": "pxdpz",

"plans": null,

"subDeploymentPackageName": "AmLife Secure Wealth\_0\_32\_15\_0",

"subDeploymentPackageVersion": "0.1",

"subMarketingName": "Secure Wealth",

"subProductCode": "BTAS3G1WPA",

"subProductName": "Secure Wealth"

}

]

}

],

"userId": "admin"

},

"type": "getProductList"

}

}

### JSON storage library

* JSONStore will be used as a local storage medium.
* Data fetched from server will be pushed into JSONStore so that it is accessible offline.
* Enable JSONStore feature before using it in project.
* To enable the JSON Store database in your project  structure> open the apps / <your\_app> folder and double click  the  application-descriptor.xml file to open it. Highlight “Optional Features” under the Overview section->Click Add->select JSONStore feature => Click OK
* Once the JSONStore is enabled, we can add collections (tables) by the following snippet.

// JSONStore init

**var** collectionName = 'users';

// Object that defines all the collections.

**var** collections = {

// Object that defines the 'people' collection.

users : {

// Object that defines the Search Fields for the 'people'

// collection.

searchFields : {

name : 'string',

age : 'integer',

address : 'string',

}

}

};

WL.JSONStore.init(collections).then(**function**() {

alert("JSONStore Init success");

}).fail(

**function**(errorObject) {

alert("JSONStore Init fail " + "errcode:" + errorObject.err

+ " msg:" + errorObject.msg);

});

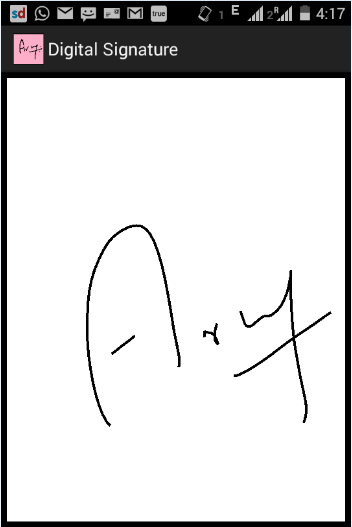
* This collection initialization should be done every time before using it for adding/querying data.
  + JSON storage will be developed as a global library to which the collection object will be passed as a parameter.
* Under Technical Research: How to query data from 2 collection classes using joins like linking 2 tables via SQLs?
  + Below are some of the cases where JSON storage will be used.
    - Product Details
    - Language Translation Text
    - Message Translation Text
    - UI Configurator
    - PDF Configurator
    - User Details for Offline support
    - Customer Details
    - Customer Fact Find Data
    - Financial Need Analysis Data
    - Illustration Data
    - Recently used customers etc
* JSON storage Encryption will be applied for sensitive data.[App Sensitive data needs to be defined]. If not, all the data stored in JSON storage will be encrypted.
* JSON storage needs to be used as much as possible for Offline storage (instead of SQLite database) mainly to have the Sync support from IBM Worklight framework.

### Capture Signature

* Signature Capture will be developed as a library wherein user can draw his signature.
* HTML5 Canvas Drawing app will be used as a reference to implement the Signature concept.

<https://github.com/krisrak/html5-canvas-drawing-app>

* Canvas Drawing feature will be implemented as a common/reusable angular directive with the above reference.



### UI configurator

* There are different scenarios where the Global Mobile Sales app need the UI configurator. This feature is mainly required to make the app features flexible so that it can support different UI for different countries.
* Scenarios:
  + 1. Module and Sub Module configurator
    2. In few places, the columns in a table need to be configured. For ex., The number of columns / data types of columns might vary depends on the product plan chosen. This needs to be defined at run time.
    3. UI needs to be adjusted based on dynamic data. For example, the number of products could be 2 or 10 in future. The number of rows in rider could be 5 or 10 or 100.
* Module and Submodule configurator needs to be defined in JSON format as follows.

Module Name: Customer Fact Find

Visibility: Y

Sub Module Name: Customer Advice Choice and declaration

Visibility: Y

Sub Module Name: Disclosure of Agent status

Visibility: Y

Sub Module Name: My financial goals process

Visibility: Y

Sub Module Name: Risk profiling process

Visibility: Y

Sub Module Name: Existing policy Details process

Visibility: N

Sub Module Name: Family details process

Visibility: N

Module Name: Financial Needs Analysis

Visibility: Y

Sub Module Name: Income Protection Process

Visibility: Y

Sub Module Name: Education Investment Need Process

Visibility: N

Module Name: Sales Illustration Process

Visibility: Y

etc

* + This JSON object will be downloaded via adapter after the user successful login.This configuration will be used to decide the visibility of the module or the sub module as well as the app navigations.
  + Sample code for this feature will be added later.
* Sample for a dynamic UI based on the JSON object:
* Sample JSON UI configurator:

[

{

"title": "TEXT",

"type": "text",

"value": "ANB",

"placeholder": "Enter User Name",

"dvalue": "rajeshraja",

"status": "readonly"

},

{

"title": "PASSWORD",

"type": "password",

"value": "ANB",

"placeholder": "password",

"dvalue": "test"

},

{

"title": "EMAIL",

"type": "email",

"value": "ANB",

"placeholder": "email"

},

{

"title": "URL",

"type": "url",

"value": "ANB",

"placeholder": "url"

},

{

"title": "DATE",

"type": "date",

"value": "ANB",

"placeholder": "date"

},

{

"title": "RANGE",

"type": "range",

"value": "ANB",

"placeholder": "range"

},

{

"title": "TIME",

"type": "time",

"value": "ANB",

"placeholder": "time"

},

{

"title": "COLOR",

"type": "color",

"value": "ANB",

"placeholder": "color"

},

{

"title": "NUMBER",

"type": "number",

"value": "ANB",

"placeholder": "number"

},

{

"title": "WEEK",

"type": "week",

"value": "ANB",

"placeholder": "week"

},

{

"title": "TEXT",

"type": "text",

"value": "ANB",

"placeholder": "username"

},

{

"title": "PASSWORD",

"type": "password",

"value": "ANB",

"placeholder": "password"

},

{

"title": "EMAIL",

"type": "email",

"value": "ANB",

"placeholder": "email"

},

{

"title": "URL",

"type": "url",

"value": "ANB",

"placeholder": "url"

},

{

"title": "DATE",

"type": "date",

"value": "ANB",

"placeholder": "date"

},

{

"title": "RANGE",

"type": "range",

"value": "ANB",

"placeholder": "range"

}

]

* In the above Json configurator, all html input box types like date, time, number, range, color etc can be configured using the field “type”.
* Value field should be unique because this will be used as a variable name. If any function is required then this name will be used.
* Place holder is for user reference. For ex., for the Username input type, the placeholder will be displayed as “Enter user name”.
* If any default values need to be given for any field, “dvalue” can be used.
* If any field needs to be set as Read Only, Status can be used.
* Using the above JSON configurator, new fields can be added at any time. Or existing fields can be removed at any time. The UI design will be generated based on the JSON object.
* Positioning of the controls need to follow certain pattern. For ex., 2 fields per row. With this approach, the system knows where to place the new field.
* Below are the steps to load a dynamic form control in Angular js.
* Sample Controller.js to create dynamic fields

Using **$http.get('json/formControls.json')** function, the above JSON object can be stored in 1 variable as follows:

**scope.field = response.data**

* The input fields will be created as follows with the above JSON data

if(scope.field.dvalue != undefined && scope.field.status != undefined)

{

element.html(scope.field.title + ': <input type='+scope.field.type+' ng-model='+scope.field.value+' placeholder='+scope.field.placeholder+' value='+scope.field.dvalue+' '+scope.field.status+' />');

}

else if(scope.field.dvalue != undefined && scope.field.status == undefined)

{

element.html(scope.field.title + ': <input type='+scope.field.type+' ng-model='+scope.field.value+' placeholder='+scope.field.placeholder+ ' value='+scope.field.dvalue+' />');

}

else if(scope.field.dvalue != undefined && scope.field.status == undefined)

{

element.html(scope.field.title + ': <input type='+scope.field.type+' ng-model='+scope.field.value+' placeholder='+scope.field.placeholder+ ' '+scope.field.status+ ' />');

}

else

{

element.html(scope.field.title + ': <input type='+scope.field.type+' ng-model='+scope.field.value+' placeholder='+scope.field.placeholder+' />');

}

* Sample HTML to create dynamic fields

Using the following tag angular js will create the input field in HTML code.

<ul class="create\_new\_left">

<li ng-form-field class="form-row" ng-repeat="field in fields"></li>

</ul>

* **ng-repeat** will load all input tag for the fields already created by angularjs.
* Sample CSS to create dynamic fields

.create\_new\_left

{

list-style-type: none;

width: 100%;

float:left;

}

.create\_new\_left li

{

float: left;

padding: 10px 2%;

width: 46%;

}

* Screen shot with dynamic fields based on the JSON object
* Samples will be added in the next version which includes the Rider sample.

### Multi-language support

* One JSON translation file will be created for each language.
* Very first time, when the app is opened, translation file for English will be downloaded.
* The downloaded data will be stored in JSON storage for Offline use.
* When the user changes the language, that respective translation file will be downloaded again and this data will be overridden in the JSON storage.

**Option 1: (using ng-translate from Angular.js)**

* Angular ng-Translate plug-in will be used to implement the multi-language support for Global Mobile sales app.
* Reference URL for ng-Translate:

<https://github.com/StephanHoyer/ng-translate>

* Steps to use ng-translate plug-in
  + - 1. Register the ng-translate library
      2. Configure the provider as below:

app.config(['$translateProvider', function ($translateProvider)

{

// Simply register translation table as object hash

$translateProvider.translations('en\_EN',

{

'Login': 'Login-EN',

'Language':'Language-EN',

'English' :'English-EN'

}); $translateProvider.translations('de\_DE',

{

'Login': 'Login-DE',

'Language':'Language-DE',

'English' :'English-DE'

});

$translateProvider.uses('en\_EN');

}]);

* + - 1. Add the json provider reference to the main controller.

app.controller('MainCtrl', function($translate)

* + - 1. To apply translation for any control, the translate function to be called.

{{‘Login’ | translate }}

Example:

<button class="homepagefrmctrls" type="submit" ng-click="submit()">{{ 'Login' | translate }}</button>

**Option 2: (using Angular.js expressions)**

* Assign all static data like Labels, Place holders etc. using angular expressions with reference to the downloaded language json file.
* By this approach all the static data can be translated to the desired language throughout the application dynamically.
* Below is the illustration code snippet.

<body ng-controller="mainController">

<label>{{login.name}}</label><input type="text"><br>

<label>{{login.age}}</label><input type="text"><br>

<label>{{login.address}}</label><input type="text"><br>

<input type="button" ng-Click="changeEN()" value="English"/>

<input type="button" ng-Click="changeFR()" value="French"/>

<script>

function mainController($scope){

//json data for English

var lang\_EN={name:"name ",age:"age ",address:"address "};

//json data for another language

var lang\_FR={name:"eman ",age:"ega ",address:"sserdda "};

$scope.login=lang\_EN;

$scope.changeEN= function(){

$scope.login=lang\_EN;

};

$scope.changeFR= function(){

$scope.login=lang\_FR;

};

}

</script>

</body>

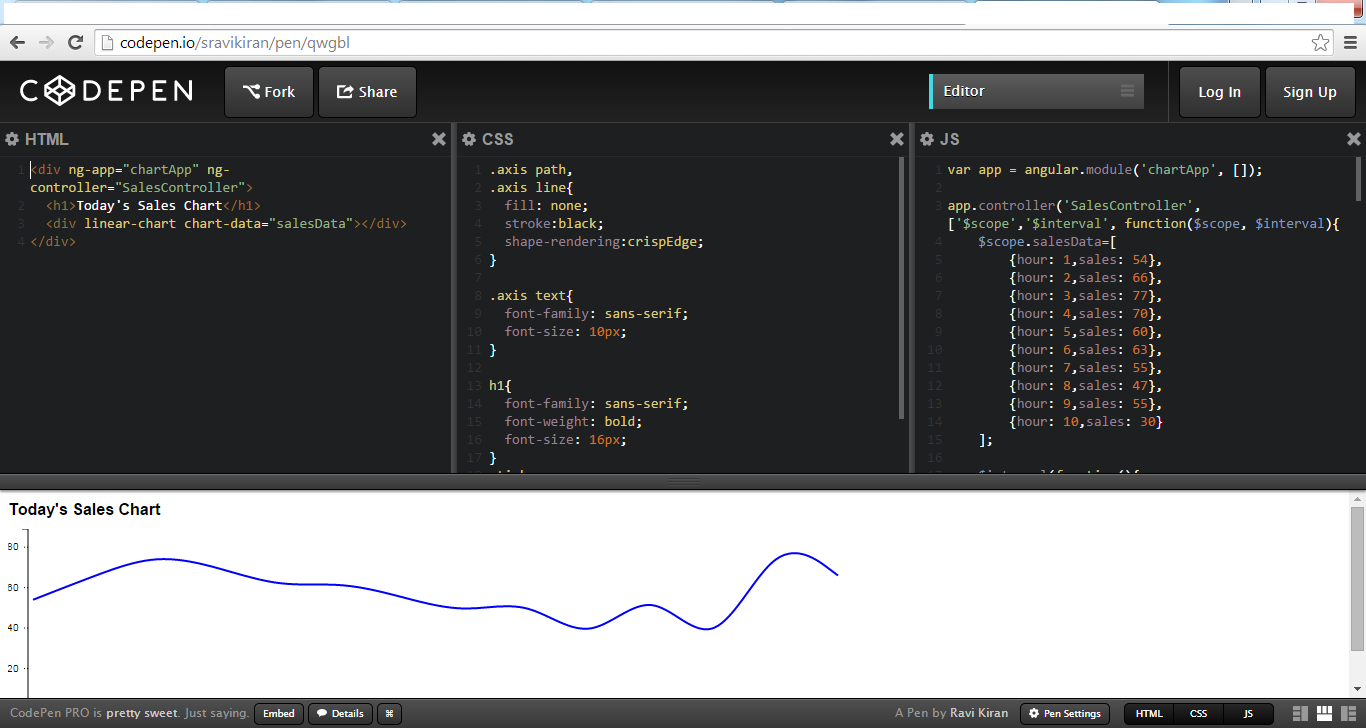
### Graph Control

* D3 javascript library will be used along with Angular JS to create graphs.
* Reference URL:

<https://github.com/mbostock/d3/wiki/Gallery>

<http://www.sitepoint.com/creating-charting-directives-using-angularjs-d3-js/>

* Sample code to implement a simple graph using D3.js and Angular js.



### PDF generator

* Implementation details will be added in the next version.

## User Login

### User Authentication

Following flowchart describes the Login Flow:



# Glossary and Acronyms

| Term or Acronym | Definition |
| --- | --- |
| API | Application Programming Interface |
|  |  |