

System Design Description

MetLife [Global Mobile Sales Platform]

|  |  |
| --- | --- |
| **Version Number:** | 1.3 |
| **Draft/Final as of:** | 25/02/2015 |
| **Printed on:** | [Date] |
| **Author** | Chella,  Mobile Solution Architect  Wipro Technologies, Bangalore  chella.pandeeswari@wipro.com |
| **Owner** | MetLife |

|  |  |
| --- | --- |
| **Approval Signatures** | |
| Name | Title |
| Komal | MetLife Mobility Applications Lead |
|  |  |

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

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Reviewed By** | **Change Description** |
| 10/02/2015 | 1.1 | Chella | Komal | Initial version |
| 21/02/2015 | 1.2 | Chella | Komal, Kishan | Updated based on V 1.1 feedback + added additional components |
| 04/03/2015 | 1.3 | Ravinder | Kishan |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

1 Scope 5

1.1 System Design Definition Overview 5

1.2 Intended Audience 5

1.3 Identification 5

1.4 References 5

2 System Architectural Design 6

2.1 System Architecture 6

2.2 Application Logical Architecture 7

2.3 Environment (Dev, QAT, UAT, Production) 7

2.3.1 System Hardware Environment and Dependencies 7

2.3.2 System Software Environment and Dependencies 7

2.4 System Integration 8

2.5 Third Party Software 9

3 Interface Definition 9

3.1 Interface Standards 9

3.1.1 Standards for Global Mobile Sales App Webservice calls 9

3.1.2 Request Header Information 10

3.1.3 Response Error Information 11

4 Processes 12

4.1 Global Components 12

4.1.1 File Transfer 12

4.1.2 Embedded Calculator Integration 14

4.1.3 Sample Web service creation 16

4.1.4 Sample Web service invocation through Worklight adapter 16

4.1.5 JSON storage library 22

4.1.6 JSON Data Model (Work In Progress) 24

4.1.7 Capture Signature 29

4.1.8 UI configurator 38

4.1.9 Multi-language support 46

4.1.10 Graph Control 47

4.1.11 PDF Viewer 50

4.1.12 Video Player 51

4.1.13 Idle Time check 52

4.1.14 PDF generator 52

4.1.15 App Authenticity 60

4.1.16 FastClick.js 61

4.1.17 Wrapper Classes 61

4.2 User Login 64

4.2.1 User Authentication 64

4.2.2 App Flow 66

4.2.3 Products Download 67

4.2.4 FNA and Illustration flow 69

5 Glossary and Acronyms 69

# 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

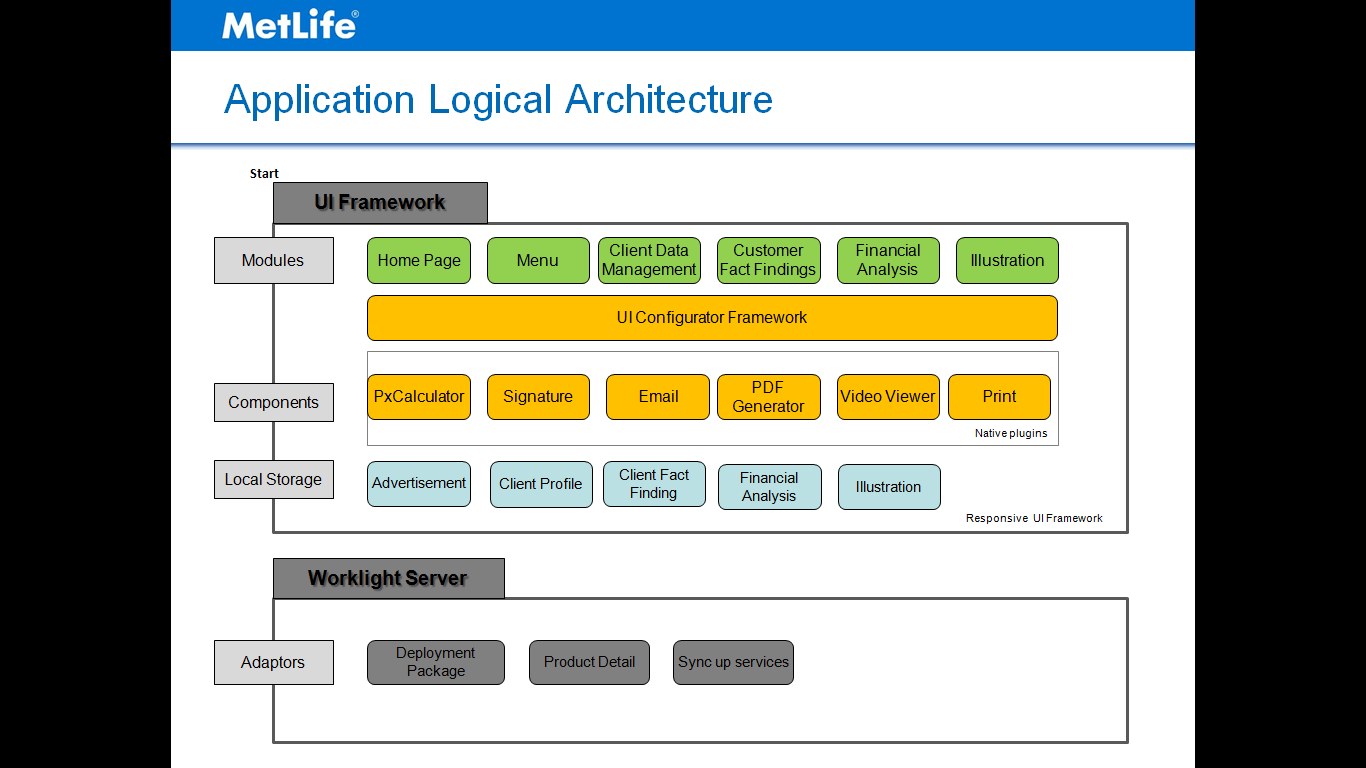
|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | **Author** | **Version** | **Date** |
| MetLife- GSP SRD | Mahadevan | Refer latest version |  |
| PX\_Calculator\_Specification.docx | HP |  |  |
| ProductExpress Deployment Pkg.zip | HP |  |  |
| MetLife Interface Specification Document | Komal | Refer latest version |  |

# 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

### System Software Environment and Dependencies

Softwares for Dev:

1. Eclipse-juno classic
2. Android OS-Android 4.2 and above
3. XCode 6.1 and above
4. iOS- 8 and above (Deployment Target: IOS 7 onwards)
5. Cordova-4.2.0 with IBM Worklight
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. IOS 7 and above
3. Software to capture the bugs
4. Worklight Server access from 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

* Cordova plug in for File Transfer

Reference URL:

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

* Cordova FileSystem API along with crypto-js javascript secure algorithm for File encryption and decryption.  Advanced Encryption Standard (AES-128) crypto algorithm will be followed wherever security is required.

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

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

* 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/>

* JSPDF library will be used to generate the PDF files from HTML view.
* Reference URL for implementation logic

<http://www.c-sharpcorner.com/UploadFile/b629e0/creating-pdf-from-html-through-jspdf/>

# 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

* **Assumption:** No service based calculator will be used for Global Sales mobile app.
* **Note -** PX DP calculator is for android native version only. Currently support is not available for IOS, so calculate is not part of IOS app.
* 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]

Note:

* 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.
* **Encryption/Decryption detail for pxdpz file**

Still the class information would be required to call these methods.

===== API Details =====

  /\*\*  
     Import key for deployment package decryption.  
       
     @param filename The filename (Including path) file containing the encryption key.  
   \*/  
  public void importKey(String filename)  
    throws PxException;  
  
  /\*\*  
     Remove key for deployment package decryption.  
       
     @param keyLabel The label of the encryption key to remove.  
   \*/  
  public void removeKey(String keyLabel)  
    throws PxException;  
      
  /\*\*  
     Relabel key for deployment package decryption.  
       
     @param oldLabel The label of the encryption key to relabel.  
     @param newLabel The new label of the encryption key.  
   \*/  
  public void relabelKey(String oldLabel, String newLabel)  
    throws PxException;  
      
  /\*\*  
     List of imported encryption key labels.  
       
     @return List containing the labels of all imported encryption keys.  
   \*/  
  public ArrayList<String> keyList()  
    throws PxException;

### 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

{

"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 with encryption feature with options.password.
* 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.
  + 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 will be used for Offline data storage (instead of SQLite database), mainly to have the Sync support from IBM WorkLight framework.

### JSON Data Model

Below table provides the JSON Objects fields to store in JSON Store Collections.

|  |  |  |  |
| --- | --- | --- | --- |
| **Collection Name** | **Sub Object** | **Key** | **Remarks** |
| ProductList | - | productCategory |  |
|  |  | productCategoryImage |  |
|  | Product Object | productCode |  |
|  |  | productName |  |
|  |  | productVersion |  |
|  |  | productImage |  |
|  | SubProduct object |  | Objects repeated in Category like Wealth, Protection |
|  |  | subProductCode |  |
|  |  | subProductName |  |
|  |  | subproductDownloadStatus |  |
|  |  | subPercentComplete |  |
|  |  | subProductDetailsVersion |  |
|  |  | subProductDetailsUpdateddate |  |
|  |  | subProductDeploymentPkgVersion |  |
|  |  | subProductDeploymentPkgName |  |
|  |  | subProductDeploymentPkgPath |  |
|  |  | subProductDeploymentPkgUpdateDate |  |
|  |  |  |  |
|  |  |  |  |
| Customer |  | dob | Unique identifier for record/document |
|  |  | id |
|  |  | salutation |  |
|  |  | customerName |  |
|  |  | idType |  |
|  |  | occupation |  |
|  |  | occupationCategory |  |
|  |  | occupationClass |  |
|  |  | email |  |
|  |  | smokingHabit |  |
|  |  | gender |  |
|  |  | maritalStatus |  |
|  |  | contacttype |  |
|  |  | contact |  |
|  |  | alternateContactType |  |
|  |  | alternateContact |  |
|  |  | estAnnualIncome |  |
|  |  | customerImageBigPath |  |
|  |  | customerImageSmallPath |  |
|  |  | updatedDate |  |
|  |  | createdDate |  |
|  |  |  |  |
| Settings |  | agentCode |  |
|  |  | agentName |  |
|  |  | email |  |
|  |  | contactNo |  |
|  |  | selectedLang |  |
|  |  | currencyType |  |
|  |  | currencyImagePath |  |
|  |  | imagePath |  |
|  |  |  |  |
| CFF |  | dob |  |
|  |  | id | Unique identifier for record/document |
|  |  | profileName |
|  |  | cffStatus |  |
|  |  | updatedDate |  |
|  |  | createdDate |  |
|  |  | customerAdviceChoice |  |
|  |  | agentDeclaration |  |
|  |  | noOfexistingPolicies |  |
|  |  | totalscore |  |
|  |  | preference |  |
|  |  | noOfFamilyMembers |  |
|  |  | noOfRecommendations |  |
|  | riskprofile |  |  |
|  |  | investForDuration |  |
|  |  | iCanAcceptRisk |  |
|  |  | tolerance |  |
|  | financialneeds object |  |  |
|  |  | financialNeed |  |
|  |  | alreadyPlanned |  |
|  |  | toDiscuss |  |
|  |  | priority |  |
|  |  | remarks |  |
|  |  | comment |  |
|  | existingPolicies object |  |  |
|  |  | policyOwner |  |
|  |  | lifeAssured |  |
|  |  | company |  |
|  |  | planType |  |
|  |  | deathBenefits |  |
|  |  | disabilityBenefits |  |
|  |  | criIllRider |  |
|  |  | otherBenefit |  |
|  |  | annualPremium |  |
|  |  | premiumType |  |
|  |  | frequency |  |
|  |  | startDate |  |
|  |  | maturityDate |  |
|  |  | projectedSumMaturity |  |
|  |  | affordability |  |
|  |  | protection |  |
|  |  | retirement |  |
|  |  | education |  |
|  |  | savings |  |
|  |  | investment |  |
|  |  | otherIncome |  |
|  |  | otherIncomeComment |  |
|  | FamilyMember object |  |  |
|  |  | memberName |  |
|  |  | relationship |  |
|  |  | gender |  |
|  |  | dob |  |
|  |  | occupation |  |
|  |  | occupationCategory |  |
|  |  | occupationClass |  |
|  |  | contactType |  |
|  |  | contact |  |
|  |  |  |  |
|  | RecordOfAdvice | selectedIllustrations |  |
|  |  | planType |  |
|  |  | sumCovered |  |
|  |  | policyOwnerName |  |
|  |  | premium |  |
|  |  | recommendingReason |  |
|  |  | otherRecommendReasons |  |
|  |  | term |  |
|  |  | additionalCoverage |  |
|  |  | frequency |  |
|  |  | bought |  |
|  |  | actionDescription |  |
|  | DeclarationAndAck |  |  |
|  |  | customerDeclaration |  |
|  |  | customerSignImage |  |
|  |  | agentStatus |  |
|  |  | agentSignImage |  |
|  |  |  |  |
| FNA |  | dob | Unique identifier for record/document |
|  |  | id |
|  |  | designedFor |  |
|  |  | createDate |  |
|  |  | lastUpdateDate |  |
|  | incomeRA object |  |  |
|  |  | annualIncome |  |
|  |  | incomePercentage |  |
|  |  | desiredIncome |  |
|  |  | inflationRate |  |
|  |  | projectionRate |  |
|  |  | yearsProtection |  |
|  |  | presentValue |  |
|  |  | capitalRequired |  |
|  |  | existingFunds |  |
|  |  | additionalCapital |  |
|  | childEducation object |  |  |
|  |  | childName |  |
|  |  | projectedInflation |  |
|  |  | currentAge |  |
|  |  | uniEntryAge |  |
|  |  | yearsLeftToUni |  |
|  |  | year1 |  |
|  |  | year2 |  |
|  |  | year3 |  |
|  |  | year4 |  |
|  |  | total |  |
|  |  | existingSavings |  |
|  |  | existingSavingsYield |  |
|  |  | futureYearlySavings |  |
|  |  | futureYearlySavingsYield |  |
|  |  | savingsProjection |  |
|  |  | addlFundsRequired |  |
|  |  | invstReturnProjection |  |
|  |  | optionAYearly |  |
|  |  | optionBMonthly |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Illustration |  | dob | Unique identifier for record/document |
|  |  | id |
|  |  | createdDate |  |
|  |  | lastUpdatedDate |  |
|  | lifeAssured object |  |  |
|  |  | salutation |  |
|  |  | lifeAssuredName |  |
|  |  | dob |  |
|  |  | gender |  |
|  |  | occupation |  |
|  |  | occupationClass |  |
|  |  | smokingHabit |  |
|  |  | isPlicyOwner |  |
|  | policyOwner object |  |  |
|  |  | salutation |  |
|  |  | policyOwnerName |  |
|  |  | relationship |  |
|  |  | dob |  |
|  |  | gender |  |
|  |  | occupation |  |
|  |  | occupationClass |  |
|  |  | smokingHabit |  |
|  | selectedProduct object |  |  |
|  |  | productCategory |  |
|  |  | productName |  |
|  |  | productCode |  |
|  | plan object |  |  |
|  |  | replica of product details object for user input data |  |
|  |  |  |  |
| Credentials |  | Email |  |
|  |  | Password |  |
|  |  |  |  |
| MyContent |  | docName |  |
|  |  | docMarketingName |  |
|  |  | docType |  |
|  |  | docVersion |  |
|  |  | docDeploymentDate |  |
|  |  | categoryName |  |
|  |  | productCategory |  |
|  |  |  |  |
| Pdf | pdfID | customerId |  |
|  |  | dob |  |
|  |  | pdfSavePath |  |
|  |  | pdfDesc |  |

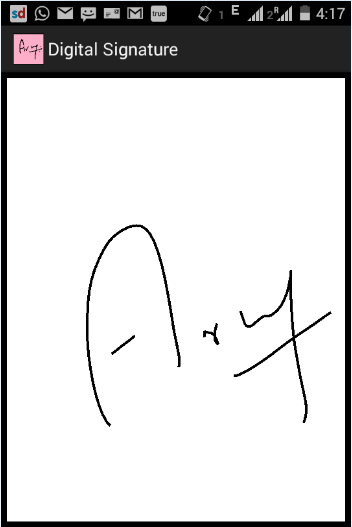
Reference – Please refer to [MetLife Interface Specification Document](#_References) for product list and product details JSON object.

### 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 base reference to implement the Signature concept.

[https://github.com/krisrak/html5-canvas-drawing-app](https://github.com/krisrak/html5-canvas-drawing-app" \t "_blank)

* Canvas Drawing feature is implemented as a common/reusable angular directive with the above reference. **Refer below for more information on this.**

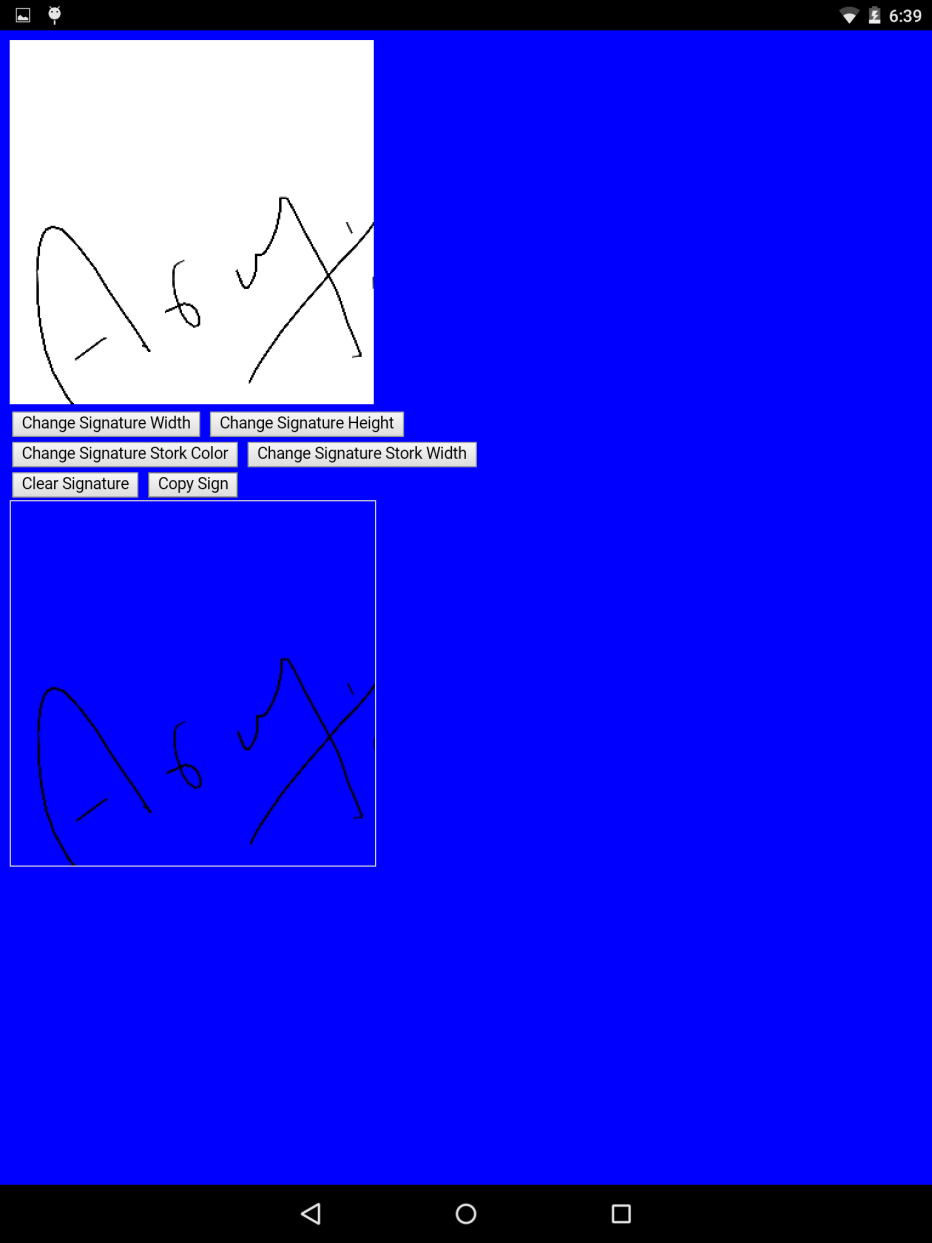


Signature is created as a global component using AngularJS directive.

**Advantages of this component:**

* Signature control width and height can be set based on the app need.
* If no width and height specified, it takes 200 pixels by default.
* The drawn signature can be fetched and shown in another canvas with transparency background. Means only the signature image will be captured and saved in a file if required.

**Output:**



Below are the steps to use signature library in any Angular mobile app.

**Steps to use Signature library**

1. Add AngularJs library files
2. Create AngularJs application module.
3. Attach signature directive to the created application module.
4. Add signature tag in html will create a component to sign over.
   * <signature SignatureWidth="300" SignatureHeight="300" signatureScope="xyz"></signature>
5. Below are the custom attributes which the signature element accepts.
   1. signatrueWidth: To set the width of the signature component (width set to 200px if signatureWidth is not available).
   2. signatureHeight: To set the height of the signature component (width set to 200px if signatureHeight is not available).
   3. signatureBgColor: To set the background color of the signature component (background color set to white if signatureBgColor is not available).
   4. signatureStrokeWidth: To set the stroke width of the signature component (stroke width set to 1px if signatureStrokeWidth is not available).
   5. signatureStrokeColor: To set the stroke color of the signature component (stroke color set to black if signatureStrokeColor is not available).
   6. signatureScope: To create a scope for signature through which the signature methods can be invoked.
6. Below are the methods provided via signatureScope object.
   1. getSignatureData(callback): returns the drawn image data of the signature to the callback function.
   * $scope.xyz.getSignatureData(function(imageData) { //xyz is the signature scope object.
   * ctx.putImageData(imageData, 0, 0); //context of temporary canvas to display the image received from getSignatureData
   * });
   1. clearSignature(): resets signature component.
   * $scope.xyz.clearSignature(); //xyz is the signature scope object.
   1. Setter and Getter methods for all the signature attributes (signatureWidth, signatureHeight, signatureBgColor, signatureStrokeWidth, signatureStrokeColor) are available via signature scope object.
   * $scope.xyz.setSignatureWidth(200); //xyz is the signature scope object.
   * $scope.xyz.getSignatureWidth();

NOTE:

1. Signature scope object should be created to invoke all the above mentioned methods of signature library.
2. Created signature scope object is available inside controller’s $scope object.

### 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

* 1. 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 to show or hide sub modules under Customer Fact Find:**

**Note:**

* “visibilitypage” field contains the tabpage name. In this case, it is set as tab1, tab2 etc.
* id field in the below JSON is used for internal logic.
* The tabpage name should be part of the id field.
* “Visibility” defines whether this tabpage should be created or not.
* The tabpages can be set as visible: yes or no based on the country. It is flexible to change this settings in the below json at any point of time. The app will behave as per that.
* The tabpage navigations also will change as per the below JSON settings.

**Sample JSON data (to set the visibility of sub modules)**

Note: Modules and Sub modules are not added in the below JSON data.

This is just to show how the app handles the visibility of the tabs and the navigations between the tabs as well identifying whether the tab is clicked once.

**There are totally 6 tabs in CFF, out of which tab3 is set as invisible.**

[

{

"Visibility": "yes",

"visibilitypage": "customeradvice",

"id": "footer\_tab1"

},

{

"Visibility": "yes",

"visibilitypage": "agentdisclosure",

"id": "footer\_tab2"

},

{

**"Visibility": "no",**

"visibilitypage": "financialgoals",

"id": "footer\_tab3"

},

{

"Visibility": "yes",

"visibilitypage": "riskprofiling",

"id": "footer\_tab4"

},

{

"Visibility": "yes",

"visibilitypage": "existingpolicy",

"id": "footer\_tab5"

},

{

"Visibility": "yes",

"visibilitypage": "familydetails",

"id": "footer\_tab6"

},

]

UI Configurator is made as a separate component with AngularJS module. Below are the steps to use UI Configurator.

**Steps to use UI** **Configurator:**

1. Add UI Configurator library file.
2. Add UI Configurator to the application module.
   * var app=angular.module("MetLife",['uiConfigurator']);
3. Add ui tag in html wherever the dynamic elements to be placed.
   * <ui uiScope="xyz"></ui> //xyz is the UI configurator scope object.
4. Every <ui> tag should contain separate uiScope object as an attribute.
5. This uiScope object will be available under controllers scope object ($scope).
6. createUi() is the method available in uiScope which will get respective JSON object (ui configurator object) as parameter by which the configurator renders elements.
   * $scope.xyz.createUi(object);
7. Elements created through ui configurator object will get the following key parameters as its descriptor.
   * **Mandatory**:
   1. tagName: Tag name of the element like div, input and button will be specified here.
   * **Optional:**
   1. Properties: All the attribute of the element like class, ng-bind, and ng-click will be added here.
      1. All AngularJS related attribute values should be prepended with $parent.
   * Example: ng-bind=”name” should be written as ng-bind=”$parent.name”.
   1. hide: it will Boolean value. Element will be hidden if it is true. Default is false.
   2. child: all the child elements will be specified as JSON Array.

{

"sales\_illustration": {

"basic\_plan\_SecureBuilder": [

{

"tagName": "ul",

"child": [

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.planCode"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.planName"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.planOption"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.planTerm"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.premiumTerm"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.sumAssured"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.yearlyPremium"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

},

{

"tagName": "li",

"child": [

{

"tagName": "label",

"properties": {

"ng-bind": "$parent.$Language.sales\_illustration.paymentFrequency"

}

},

{

"tagName": "input",

"properties": {

"type": "text"

}

}

]

}

]

}

]

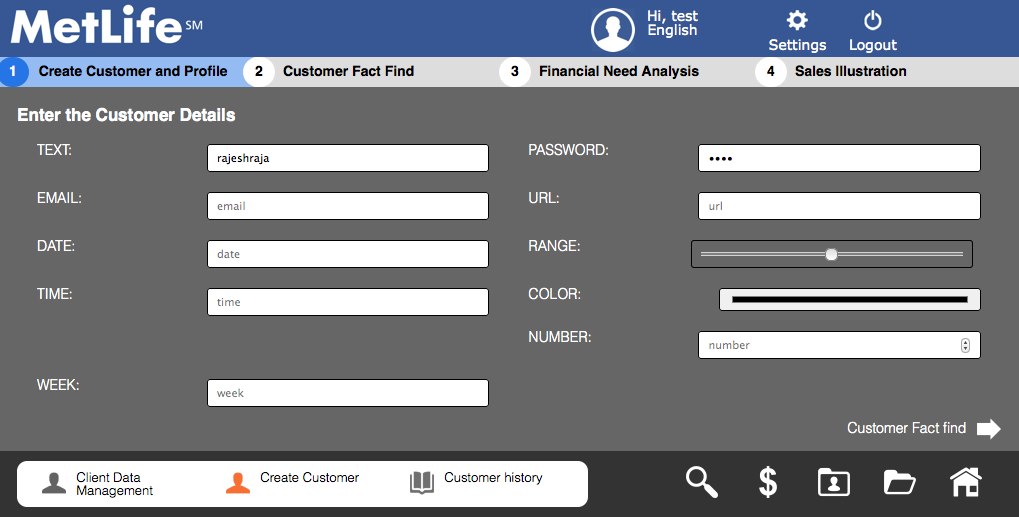
}

}



1. UI tag will get replaced once the respective dynamic elements are rendered.

* Screen shot with dynamic fields based on the JSON object



### 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 data will be stored in JSON storage for Offline use.
* When the user changes the language, that respective translation file will be loaded in to JSON storage and current data will be overridden.

Language Localization is made as a separate component with AngularJS service. Below are the steps to use Language library.

* 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.

**Steps to use Language library.**

1. Add Language service to the application module.

app.service('$Language',function($rootScope,$http){

this.changeLanguage=function(language){

$http.get('json/'+language+'.json').then(function(values) //as of now language object is stored as .json file

{

$rootScope.$Language = values.data;

});

};

});

1. setLanguage() is the method provided by $Language service which takes language option as a parameter.
2. Upon the specified language option setLanguage() method will load the respective language object to the application $rootScope so that it will be accessible throughout the application.
3. Default language should be set by calling setLanguage() in the run method of the application module.
   * var app=angular.module("MetLife",[]);
   * app.run(function($rootScope,$Language) {
   * $Language.setLanguage("Language\_EN");
   * });
4. Application Language can be changed dynamically by calling setLanguage() method.
5. Language object is created in the form of module->component.
   * {
   * "sales\_illustration": {
   * "planCode": "Plan Code",
   * "planName": "Plan Name",
   * "planOption": "Plan Option",
   * "planTerm": "Plan Term",
   * "premiumTerm": "Premium Term",
   * "sumAssured": "Sum Assured(RM)",
   * "yearlyPremium": "Yearly Premium(RM)",
   * "paymentFrequency": "Payment Frequency",
   * "plan":"Plan",
   * "totalPremium":"Total Premium",
   * "yearly":"Yearly",
   * "halfYearly":"Half Yearly",
   * "quarterly":"Quarterly",
   * "monthly":"Monthly",
   * "calculate":"CALCULATE",
   * "basicPlan":"Basic Plan",
   * "basicPremium":"Basic Premium",
   * "riders":"Riders",
   * "fundType":"Fund Type",
   * "topUps":"Top Ups",
   * "preferences":"Preferences"
   * }
   * }
6. These language objects are available under $Language object throughout the application.
   * <label ng-bind="$Language.sales\_illustration.basicPlan"></label>

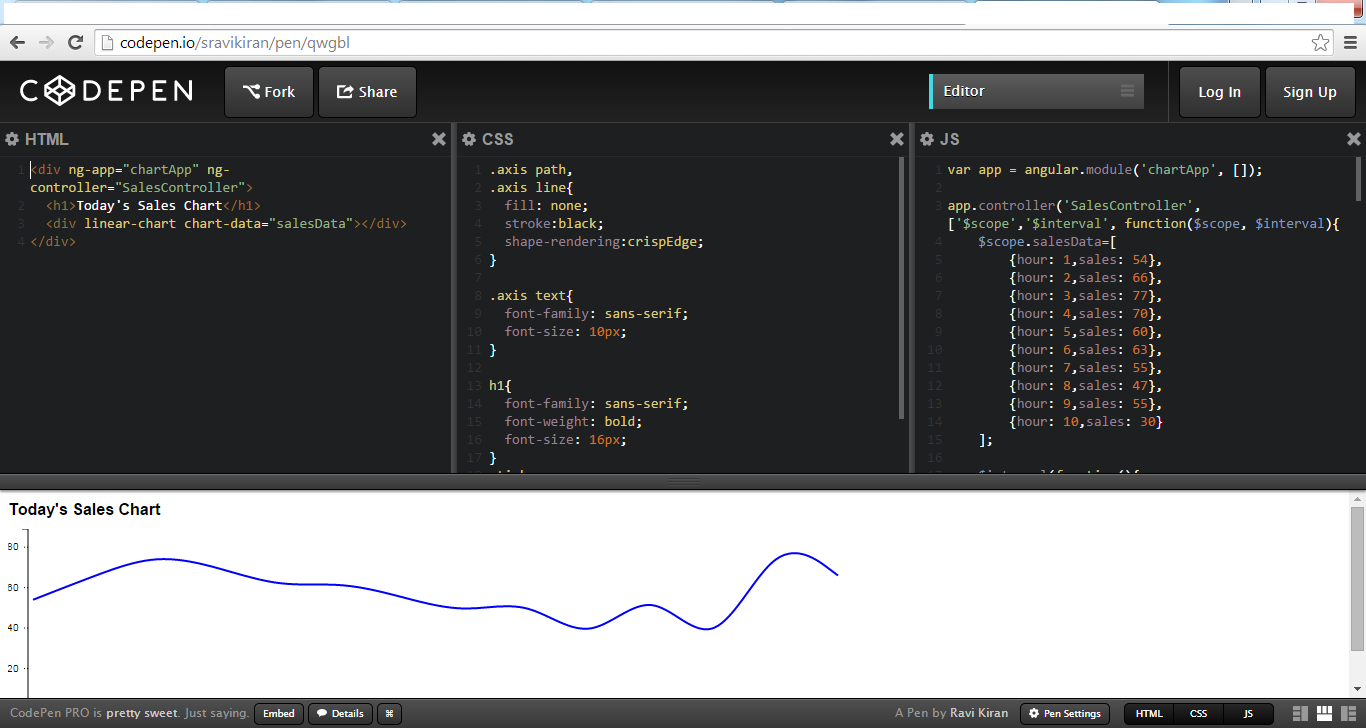
### Graph Control

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

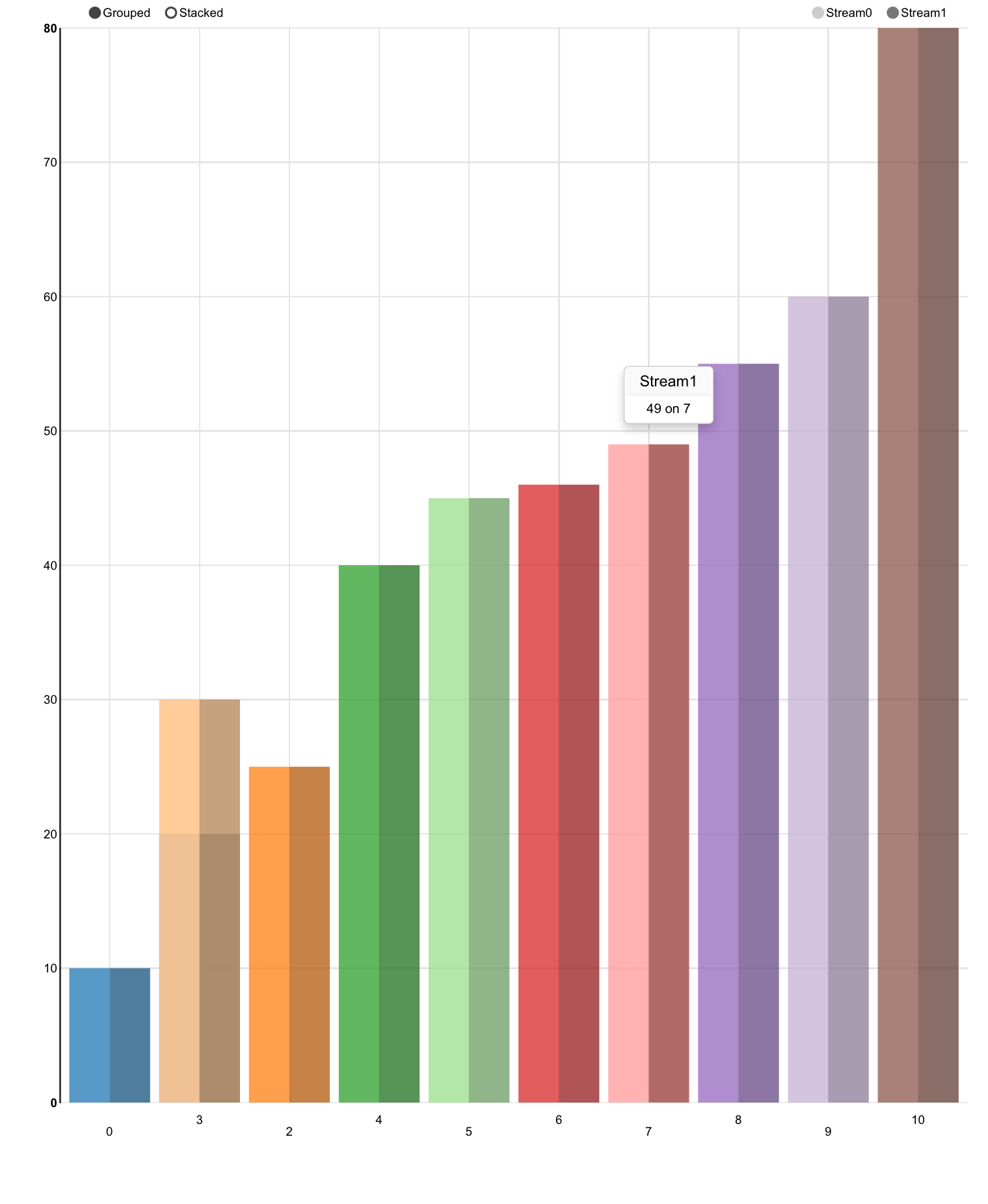
<https://github.com/mbostock/d3/wiki/Gallery>

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

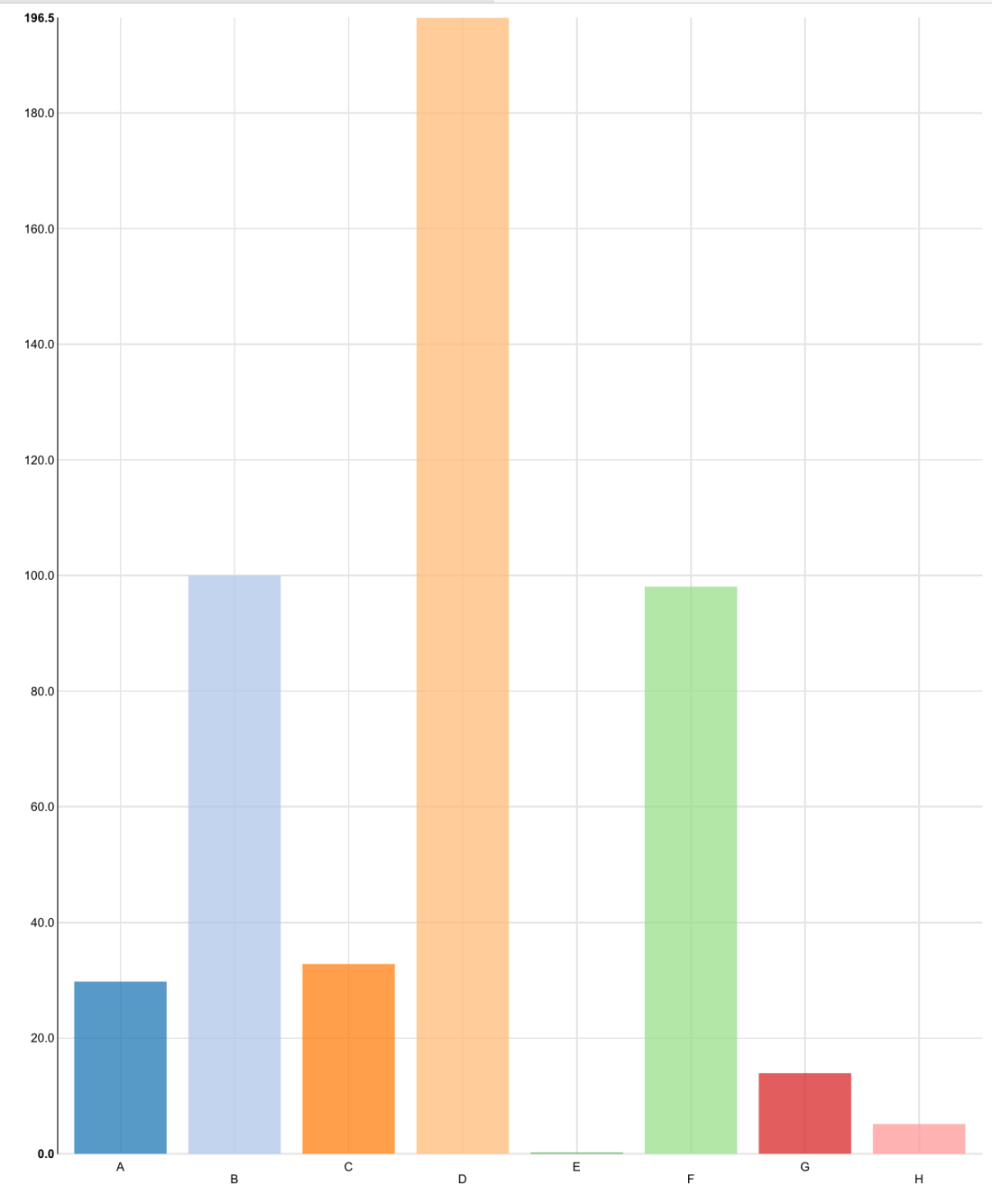
* Sample code to implement a simple graph using D3.js and Angular js.



* **Sample graph-1 created using D3.js and Angular js.**



* **Sample graph-2**



### PDF Viewer

* AngularJS PDF viewer directive will be used with pdf.js to display the PDF in angular app.
* Sample code of ng-pdfviewer

**AngularJS PDF viewer directive using pdf.js.**

<button ng-click="prevPage()">&lt;</button>

<button ng-click="nextPage()">&gt;</button>

<br>

<span>{{currentPage}}/{{totalPages}}</span>

<br>

**<pdfviewer src="test.pdf" on-page-load='pageLoaded(page,total)' id="viewer"></pdfviewer>**

and in your AngularJS code:

var app = angular.module('testApp', [ 'ngPDFViewer' ]);

app.controller('TestCtrl', [ '$scope', 'PDFViewerService', function($scope, pdf) {

$scope.viewer = pdf.Instance("viewer");

$scope.nextPage = function() {

$scope.viewer.nextPage();

};

$scope.prevPage = function() {

$scope.viewer.prevPage();

};

$scope.pageLoaded = function(curPage, totalPages) {

$scope.currentPage = curPage;

$scope.totalPages = totalPages;

};

}]);

* Reference:

<https://github.com/akrennmair/ng-pdfviewer>

### Video Player

* Ng-src will be used to play the video.
* Play Video will be created as an angular directive to play videos anywhere on the app.
* Height and Width of the video control can be set based on the screen design.
* Sample code to create Play Video directive:

**Angular Custom directive:**

app.directive('playvideo', function () {

        return {

            restrict: 'EA',

            replace: true,

            transclude: true,

            scope: {

                video: '@'

            },

            template: '<div class="playvideo">' +

**'<iframe ng-src="{{ video }}"></iframe>' +**

                      '</div>',

            link: function (scope, element, attrs) {

                var ratio = (attrs.height / attrs.width) \* 100;

            }

        };

    });

**HTML code using Playvideo directive**

<playvideo video="http://vimeo.com/112713789" height="50"width="50"></playvideo>

<playvideo video="http://www.youtube.com/embed/JMl8cQjBfqk"width="50" height="50"></playvideo>​

### Idle Time check

* ng-idle angularjs module will be used to detect and respond to app idle time.
* App idle time can be configured in AppConfig JSON object. Default idle time is 15 min.

### PDF generator

* JSPDF library will be used to generate the PDF files from HTML view.
* **Implementation Logic:**

1. For each product plan, a HTML view will be generated.
2. For each HTML view, there should be a JSON object to supply the data.
3. To apply translation, there should be a JSON object to supply the translated text for each static text in the HTML view for a specified language.
4. JSON data is applied to the HTML view including the translation.
5. The final HTML view is passed as an input to jsPDF to generate the PDF document.

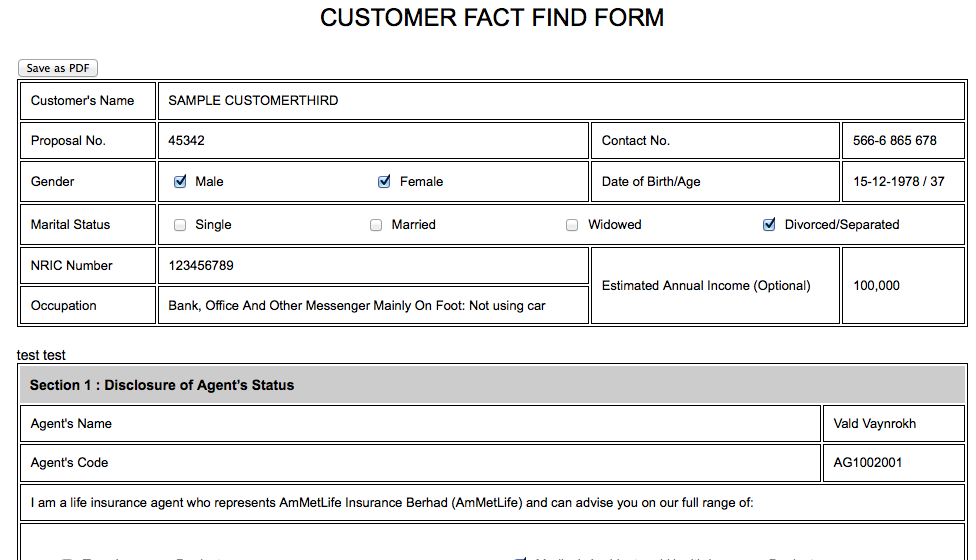
**Note:**

If there are similar layouts for different products / product plans, the HMTL view can be combined together. This call will be taken during the development.

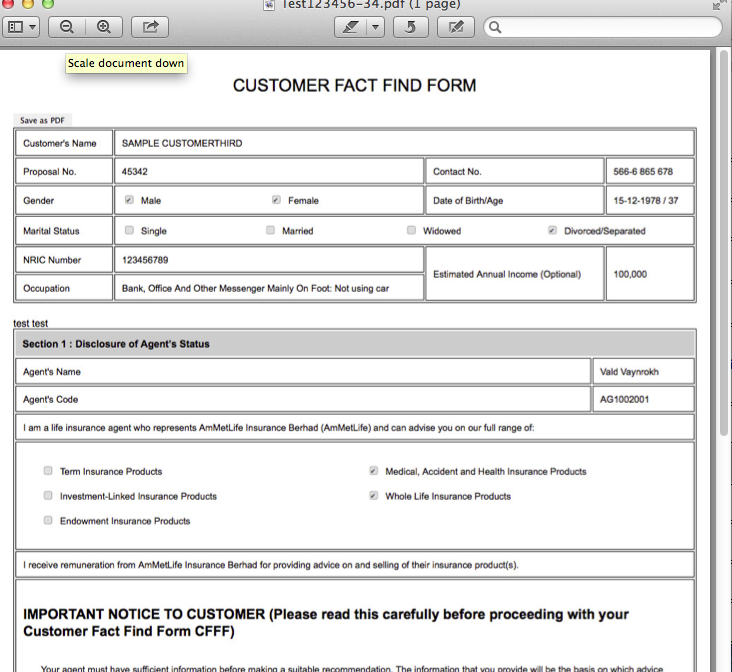
**About jsPDF library**

* JSPDF is an open-source library for generating PDF documents using nothing but JavaScript.
* It uses various functions to create various elements of PDF pages.  
    
  For example:
* **doc.text(x, y, 'string');**  
  Will print the string given in single quotes starting from the position given as point (x,y).  
    
  Instead of using a string, we can select a tag from a HTML page using JavaScript or jQuery.
* **doc.save('filename.pdf');**  
  Will save the document with the name "filename".
* **doc.addPage();**  
  Gets an extra page in the PDF file.
* **doc.setFontType('stylename');**  
  Changes the style of the font such as to italic or bold.
* **doc.setFont('fontfaceName');**  
  Provides the font face, like Times New Roman, Comic, Arial and so on.

**Sample HTML View:**



**Sample PDF View:**



**Angular JS code to save the PDF file using jsPDF:**

function saveAsPdf(){

var pdf = new jsPDF('p','pt','a4');

pdf.addHTML(document.body,function() {

pdf.save('Sample Customer Fact Find.pdf');

});

Note:

[**jsPDF**](https://mrrio.github.io/jsPDF/doc/symbols/jsPDF.html#constructor)(orientation, unit, format)

Creates new jsPDF document object instance

**jsPDF**(orientation, unit, format)

Creates new jsPDF document object instance

**Parameters:**

**orientation**

One of "portrait" or "landscape" (or shortcuts "p" (Default), "l")

**unit**

Measurement unit to be used when coordinates are specified. One of "pt" (points), "mm" (Default), "cm", "in"

**format**

One of 'a3', 'a4' (Default),'a5' ,'letter' ,'legal'

**Returns:**

*{*[*jsPDF*](https://mrrio.github.io/jsPDF/doc/symbols/jsPDF.html)*}*

**Sample JSON Data for the PDF view:**

title : "CUSTOMER FACT FIND FORM",

table1 : {

title : "",

data : {

rows : [

[

{

dataType : "simple",

label : "Customer's Name",

value : "SAMPLE CUSTOMERTHIRD"

}

],

[

{

dataType : "simple",

label : "Proposal No.",

value : "45342"

},

{

dataType : "simple",

label : "Contact No.",

value : "566-6 865 678"

}

],

[

{

dataType : "options",

label : "Gender",

options : [

{

optionValue : "Male",

isChecked : "1"

},

{

optionValue : "Female",

isChecked : "1"

}

]

},

{

dataType : "simple",

label : "Date of Birth/Age",

value : "15-12-1978 / 37"

}

],

[

{

dataType : "options",

label : "Marital Status",

options : [

{

optionValue : "Single",

isChecked : "0"

},

{

optionValue : "Married",

isChecked : "0"

},

{

optionValue : "Widowed",

isChecked : "0"

},

{

optionValue : "Divorced/Separated",

isChecked : "1"

}

]

}

],

[

{

dataType : "simple",

label : "NRIC Number",

value : "123456789"

},

{

dataType : "simple",

label : "Estimated Annual Income (Optional)",

value : "100,000"

}

],

[

{

dataType : "simple",

label : "Occupation",

value : "Bank, Office And Other Messenger Mainly On Foot: Not using car"

}

]

]

}

},

table2 : {

title : "Section 1 : Disclosure of Agent’s Status",

data : {

rows : [

[

{

dataType : "simple",

label : "Agent's Name",

value : "Vald Vaynrokh"

}

],

[

{

dataType : "simple",

label : "Agent's Code",

value : "AG1002001"

}

],

[

{

dataType : "label",

label : "I am a life insurance agent who represents AmMetLife Insurance Berhad (AmMetLife) and can advise you on our full range of:"

}

],

[

{

dataType : "options",

label : "",

options : [

{

optionValue : "Term Insurance Products",

isChecked : "0"

},

{

optionValue : "Investment-Linked Insurance Products",

isChecked : "0"

},

{

optionValue : "Endowment Insurance Products",

isChecked : "0"

},

{

optionValue : "Medical, Accident and Health Insurance Products",

isChecked : "1"

},

{

optionValue : "Whole Life Insurance Products",

isChecked : "1"

}

]

}

],

[

{

dataType : "label",

label : "I receive remuneration from AmMetLife Insurance Berhad for providing advice on and selling of their insurance product(s)."

}

],

[

{

dataType : "list",

label : "IMPORTANT NOTICE TO CUSTOMER (Please read this carefully before proceeding with your Customer Fact Find Form CFFF)",

listItems : [

"Your agent must have sufficient information before making a suitable recommendation. The information that you provide will be the basis on which advice is given.",

"If you choose not to provide all the relevant information requested, your agent may not be able to provide you with suitable advice and as a result, you may risk making a financial commitment to a life insurance policy inappropriate to your needs.",

"Your agent is required to preserve the confidentiality of information disclosed by you and restrict the use of such information only for the purpose of recommending life insurance product(s).",

"You must ensure that important information regarding the policy is disclosed to you and that you understand that information. Where something is not clear, you should seek an explanation from the agent or AmMetLife.",

"Before you make a decision to purchase any life insurance policy, you must satisfy yourself that the product(s) meets your insurance needs and resources."

]

}

]

]

}

},

}

### App Authenticity

* **Add securityTest="MetLifeCustomAppAuthenticitySecurity"**in application-descriptor.xml
* Note that for App Authenticity to **really work**, you **must use** the Consumer edition of IBM Worklight.
* **Steps to enable App Authenticity:**

1. Configure a security test that has the wl\_authenticityRealm realm:

<customSecurityTest name="customTests">

<test realm="wl\_antiXSRFRealm" step="1"/>

<test realm="wl\_authenticityRealm" step="1"/>

<test realm="wl\_remoteDisableRealm" step="1"/>

<test realm="wl\_anonymousUserRealm" isInternalUserID="true" step="1"/>

<test realm="wl\_deviceNoProvisioningRealm" isInternalDeviceID="true" step="2"/>

</customSecurityTest>

1. Place this securityTest on the environment(s) in application-descriptor.xml, for example:

<android version="1.0" securityTest="customTests">

1. For Android, generate the public signing key:

<android version="1.0" securityTest="customTests">

<worklightSettings include="true"/>

<security>

<encryptWebResources enabled="false"/>

<testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>

<publicSigningKey>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBdfdsfdsfiQKBgQCPCbaCfAfnAqQ12/S5LLfA4cBz/3INyLRPhFGSVFztdWNzPhrna1xDc8/3V1sUIW2odfdfddfdfffdR2n3dAnNMVydfdfdfv68gmU5qVCN4LxSKKRAj7VVbhBxBIEt5MbY+c0o7NZ2Pgu/moJee8Wgu8veZ1TJntYn+cWCYuI/JSnA9nsskwhSdPHK32edsfsdfdfQIDAQAB</publicSigningKey>

</security>

</android>

1. For iOS, place the app bundleId:

<iphone bundleId="com.worklight.test" version="1.0" securityTest="customTests">

1. Run As > Run on Worklight Development Server

Now the App Authenticity dropdown should be enabled in Worklight Console.

### FastClick.js

* Reference URL: **[https://docs.angularjs.org/api/ngTouch/directive/ngClick](https://docs.angularjs.org/api/ngTouch/directive/ngClick" \t "_blank)**
* The above URL says it is incorporated with ngTouch/ngclick but if there is any concern during the development then fastclick.js will be considered.

### Wrapper Classes

1. All IBM Worklight adapters need to be accessed via wrapper classes. For example, Worklight adapter needs to be called through a wrapper. This needs to be followed throughout the implementation mainly to decouple worklight objects from Metlife Global Sales app.
2. Adapter wrapper is an separate angular module which needs to be included in application module for usage
   * angular.module('adapterWrapper', []).service('$adapterUtility', function(){
   * this.invoke= function(p1,p2,p3,p4,p5,p6){
   * var options={},invocationData={}; **//Adapter wrapper class**
   * if(typeof p6 !== "undefined") { //p1,p2,p3,p4,p5,p6 are formal parameters
   * invocationData = {
   * adapter : p1,
   * procedure : p2,
   * parameters : [p3],
   * compressResponse : p4
   * };
   * options = {
   * onSuccess : p5,
   * onFailure : p6
   * };
   * }
   * else if(typeof p5 !== "undefined") {
   * invocationData = {
   * adapter : p1,
   * procedure : p2,
   * parameters : [p3]
   * };
   * options = {
   * onSuccess : p4,
   * onFailure : p5
   * };
   * }
   * else if(typeof p4 !== "undefined") {
   * invocationData = {
   * adapter : p1,
   * procedure : p2,
   * parameters : []
   * };
   * options = {
   * onSuccess : p3,
   * onFailure : p4
   * };
   * }
   * WL.Client.invokeProcedure(invocationData, options);
   * };
   * });
   * **//Attaching Adapter Wrapper class to application module.**
   * **var** app = angular.module('MetLife', ['common','adapterWrapper']);
3. $adapterUtility is the service provided by Adapter wrapper class which holds the method invoke() for invoking any adapter calls.
4. Invoke() method can take maximum of 6 parameters and minimum of 4 parameters.
5. The four mandatory parameters for invoke() method are adapterName, procedureName, successCallback and failureCallback.
6. If number of parameter is 6:

Following are the parameters that need to be included and in the following Sequence:

adapterName: Name of the adapter.

procedureName: Name of the procedure inside that adapter.

parameter: Parameters that need to be passed for POST request. Parameter should be in string format. If it is in json format, it need to be converted into string.

compressResponse: It takes boolean value. Whether to compress the response.

successCallback: If the result is success, successCallback function is called.

failureCallback: If the result is error, successCallback function is called.

$adapterUtility.invoke(adapterName,procedureName,parameter,compressResponse,successCallback,failureCallback);

}

function successCallback(result)

{

//result object will hold the responseData from the webservice

}

function failureCallback(Error)

{

//error object will hold error code and error message.

}

1. If number of parameter is 5:
   * Following are the parameters that need to be included and in the following Sequence:

adapterName: Name of the adapter.

procedureName: Name of the procedure inside that adapter.

parameter: Parameters that need to be passed for POST request. Parameter should be in string format. If it is in json format, it need to be converted into string.

successCallback: If the result is success, successCallback function is called.

failureCallback: If the result is error, successCallback function is called.

$adapterUtility.invoke(adapterName,procedureName,parameter,successCallback,failureCallback);

}

function successCallback(result)

{

//result object will hold the responseData from the webservice

}

function failureCallback(Error)

{

//error object will hold error code and error message.

}

1. If number of parameter is 4 (for GET request):

Following are the parameters that need to be included and in the following Sequence:

adapterName: Name of the adapter.

procedureName: Name of the procedure inside that adapter.

successCallback: If the result is success, successCallback function is called.

failureCallback: If the result is error, successCallback function is called.

$adapterUtility.invoke(adapterName,procedureName,successCallback,failureCallback);

}

function successCallback(result)

{

//result object will hold the responseData from the webservice

}

function failureCallback(Error)

{

//error object will hold error code and error message.

}

## User Login

### User Authentication

Following flowchart describes the basic Login Flow:



**Processing and Implementation Logic for Login screen:**

* Very first time when the user logs into the app, nothing will be present in JSONstore.
* He should be online to login to the app the very first time. If he is offline and tries to login, he will get the error saying that “Network connectivity is required to login to the app for the very first usage”
* Once he enters his username and password and clicks on Login button, LoginAuthentication webservice is called which validates the user credentials.
* LoginAuthentication webservice returns the JSON object with only 1 user details as [admin@gsp.com](mailto:admin@gsp.com) as the username and “admin1234#” as the password.
* After the successful authentication of the user, the username and password gets stored in JSON storage with encrypted format. This will be used for offline authentication.
* If the user has not used the app for 30 mins for example, the app will do auto logout. This 30 mins is configurable in AppConfig JSON object.
* Next time when the user logs in, first the system checks whether the network connectivity is there. If so, it verifies the user credentials against the server. If valid, it takes the user to Home screen. If invalid, it asks the user to enter the right credentials and clears the user credentials from JSON storage. If there is no network connectivity, then the system verifies the user credentials with the local JSON storage.
* If the user minimizes the app and revisits the app once again, ie, when the app come to foreground stage and the user is ONLINE, the app verifies the user credentials at the background without popping up the login screen. Means it verifies the user credentials stored in the local JSON store against the server. If valid, it allows the user to continue working on the app where he is. If invalid, it displays the message to the user saying that his credentials are invalid and he needs to login to the app again. It shows the Login screen to the user to enter his credentials again.

### App Flow

Following flow chart describes the app flow for different module.

Main points:

* 1. User can start the flow from customer create, customer fact find form, FNA and Sales Illustration.
  2. FNA is optional to go Sales Illustration.
  3. User can directly start the FNA flow without selecting customer by selecting FNA icon in the bottom footer.
  4. User can directly start the Sales Illustration flow without selecting customer by selecting Illustration icon in the bottom footer.
  5. User can do Sales Illustration after FNA without selecting customer in direct flow.
  6. After completing CFF or FNA or Illustration, user will be directed to profile status screen.



### Products Download

Following flowchart provides the detailed flow for downloading of common data, product deployment package, product list, product details and content.



### FNA and Illustration flow

Following flowchart provides the detailed flow for financial need analysis and sales illustration.



### Screen Navigation Flow

Following flowchart describes the detailed screen navigation flow.



# Glossary and Acronyms

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