

Mobile Banking for Banco de Chile

Technical Design Document

Version 1.9

16 October 2013

Prepared for:

http://upload.wikimedia.org/wikipedia/en/thumb/a/aa/Banco_de_Chile_Logo.png/200px-Banco_de_Chile_Logo.png

SAP Professional Services

1337 East 750 North

Orem, Utah

Contents

[2 Change Log 4](#_Toc358210285)

[3 Introduction 5](#_Toc358210286)

[3.1 Overview 5](#_Toc358210287)

[3.2 Scope of Project 5](#_Toc358210288)

[3.3 Out of scope 5](#_Toc358210289)

[3.4 Copyright 5](#_Toc358210290)

[3.5 Related Documentation 6](#_Toc358210291)

[3.6 Glossary 6](#_Toc358210292)

[4 Technical Approach 6](#_Toc358210293)

[4.1 Development Phases 6](#_Toc358210294)

[4.2 Assumptions 6](#_Toc358210295)

[4.3 Dependencies 7](#_Toc358210296)

[4.4 Constraints 7](#_Toc358210297)

[4.5 Risks 7](#_Toc358210298)

[5 Architecture 8](#_Toc358210299)

[5.1 Component Overview 9](#_Toc358210300)

[5.1.1 mBanking Gateway 10](#_Toc358210301)

[5.1.2 Hibernate Persistence 11](#_Toc358210302)

[5.1.3 Aspects 11](#_Toc358210303)

[5.1.4 OSGi Preferences 12](#_Toc358210304)

[5.1.5 Logging 13](#_Toc358210305)

[5.2 Bundling Structure 14](#_Toc358210306)

[5.3 Deployment Overview 14](#_Toc358210307)

[5.3.1 Environments 14](#_Toc358210308)

[5.3.2 Access to Environments 14](#_Toc358210309)

[5.3.3 Test Data 15](#_Toc358210310)

[5.3.4 Security Tokens 15](#_Toc358210311)

[5.4 Settings 15](#_Toc358210312)

[5.4.1 JDBC Connection Pool 15](#_Toc358210313)

[5.4.2 JVM Settings 16](#_Toc358210314)

[5.5 Exception Handling and Error Code 16](#_Toc358210315)

[5.5.1 Errors from Bank Web Services 16](#_Toc358210316)

[5.5.2 Error Handling in Mobiliser mBanking 16](#_Toc358210317)

[5.5.3 Code Implementation 17](#_Toc358210318)

[6 User Interface 17](#_Toc358210319)

[6.1 Multi-Language 18](#_Toc358210320)

[6.2 Multi-Bank 18](#_Toc358210321)

[6.3 Customer Segments 18](#_Toc358210322)

[7 Web Services for Banking 18](#_Toc358210323)

[8 Communication Design 19](#_Toc358210324)

[8.1 SSL Encrypted Communication 19](#_Toc358210325)

[8.2 Interfaces 19](#_Toc358210326)

[9 Customer Messaging 19](#_Toc358210327)

[10 Security 20](#_Toc358210328)

[11 Customizations to Database 20](#_Toc358210329)

[11.1 Customizations for Banco de Chile 20](#_Toc358210330)

[11.2 Best Practices 21](#_Toc358210331)

[11.3 Persistence Layer 22](#_Toc358210332)

[12 Drop-down lists 22](#_Toc358210333)

[13 Customer Identification 22](#_Toc358210334)

[13.1 RUT number 22](#_Toc358210335)

[13.2 Device ID 24](#_Toc358210336)

[14 Customer enrollment 25](#_Toc358210337)

[15 Customer login 26](#_Toc358210338)

[16 Auditing 26](#_Toc358210339)

[17 Session Management 27](#_Toc358210340)

[18 Privileges 28](#_Toc358210341)

[19 Appendix A – Guidelines for Web Services 28](#_Toc358210342)

# Change Log

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Contact | Version | Change Description |
| 5/24/2013 | Brian Allred | 1.0 | Initial Revision. |
| 6/4/2013 | Brian Allred | 1.1 | Added information about RUT numbers |
| 6/5/2013 | Brian Allred | 1.2 | Added clarifications |
| 5/7/2013 | Brian Allred | 1.3 | Additional clarifications |
| 6/19/2013 | Brian Allred | 1.4 | Included new decisions resulting from recent conversations. |
| 6/26/2013 | Brian Allred | 1.5 | More definition of message text in the database. |
| 6/28/2013 | Brian Allred | 1.6 | More details about error numbers and session management. |
| 7/19/2013 | Brian Allred | 1.7 | More details about enrollment, preferences, and auditing. |
| 7/24/2013 | Brian Allred | 1.8 | Clarification added for deliverables. |
| 10/16/2013 | Brian Allred | 1.9 | Changes to audit logging (Section 16) requested by the bank. |

# Introduction

## Overview

The purpose of this document is to provide technical design information describing the Mobile Banking solution:

* Banco de Chile components
* SAP components
* Integration points
* Required customizations

This document describes some of the Banco de Chile systems, but this document is not intended to offer detailed information on the architecture of the Banco de Chile systems.

## Scope of Project

SAP Mobile Banking will provide mobile banking for customers of Banco de Chile. The first phase of the project will provide similar functionality to the existing mobile banking application at the bank, plus a few additional features, namely the RedGiro feature. More features will be added in subsequent phases.

The features to be included in the first phase are documented in separate ‘Use Case’ documents.

SAP will deliver installable apps and a mobile web (WAP) interface. The installable apps will be provided only for Android and iOS. See section 6 below for more details about the deliverables.

## Out of scope

None of the following items will be done in this project:

* Multi-language
* Multi-country
* Data migration
* Data cleaning

## Copyright

This document is the property of SAP. It may not be copied in whole or in part, or presented to other parties outside of Banco de Chile without the prior written consent of SAP.

## Related Documentation

|  |  |
| --- | --- |
| Document Name | Date |
| Use Case Document | 05/24/2013 |
| Project Schedule | 05/24/2013 |
| Mobiliser Installation Guide 5.1 |  |

## Glossary

|  |  |  |
| --- | --- | --- |
| Term | Description | Domain |
| mBanking | The name for SAP’s Mobile Banking application. It is part of SAP’s Mobiliser product. | SAP |
| Mobiliser | SAP’s mobile financial framework. Version 5.1 is the latest version, and version 5.1 will be used for this project. | SAP |
| Smartphone Mobiliser | The development tools inside Mobiliser to build installable applications for smart phones. It is based on Apache Cordova. |  |

# Technical Approach

## Development Phases

Preliminary builds of the application will be delivered during development, with a final build to be delivered when the application is complete. The delivery schedule will be documented in a separate project schedule document.

## Assumptions

* Bank will provide Web Services to access all banking functionality.
* Bank will provide documentation for how to use its web services.
* The application will be build using SAP mBanking Mobiliser version 5.1.
* The application includes a database of its own, but for this application the goal is to store the least amount of data possible in the mBanking database. A minimal customer profile and ‘audit’ information will be stored. See the ‘Customizations to Database’ section below.

## Dependencies

* The project depends on web services developed by the bank. Development depends on these services being documented. Development also depends on these web services being accessible and available in a ‘development’ environment.
* The bank needs to provide access to a ‘development’ environment and a ‘system test’ environment where web services are running and responding.
* The bank agrees to provide useful test data in the development environments.
* The bank agrees to provide VPN access to the development environments.
* The bank agrees to provide a document showing web service error numbers and the error message text to display for each error number.
* The bank and SAP need to agree on a tool to tracking defects, which will be accessible by the bank and SAP.

## Constraints

* None identified at this time

## Risks

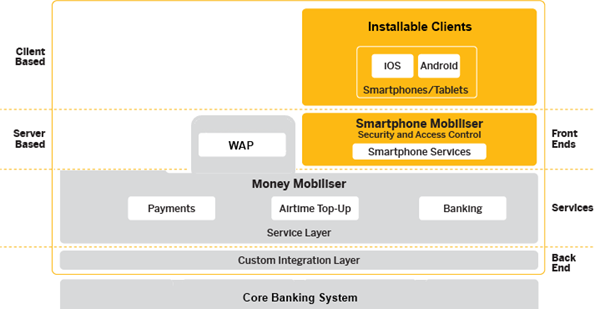
* Time and resources
* Web design being completed
* Development and QA environments being available and accessible
* Bank web services being available and accessible
* Bank web services sufficiently documented
* VPN access by developers to development environments
* Any additional changes to business requirements

# Architecture

The mBanking platform conforms to the standard three-tier architecture for enterprise applications.

|  |  |  |
| --- | --- | --- |
| Presentation Layer | The user interface | HTML and CSS visible to customer. |
| Logic Layer | The business logic of the application | Executes in Java on the application server |
| Data Layer | The banking and customer data | The bank’s back-end system |

This image shows the major parts of the architecture:



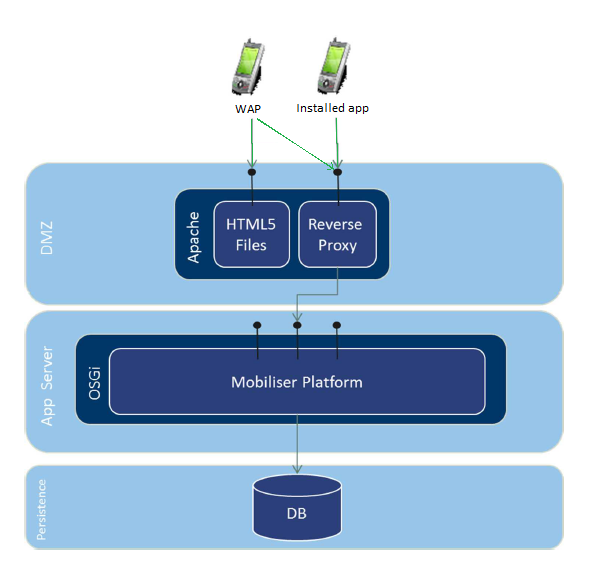
Here is another image showing the component servers:



The core SAP Mobile Banking application runs on an app server, and is composed of bundles running on Apache Felix, which is an OSGi container. The OSGi framework provides better encapsulation of actions and allows for dynamic reloading of code. The web application makes the service calls into the core banking system at Banco de Chile by calling web services.

The Mobile Banking WAP interface will be deployed as a WAR file, which will be deployed to Apache so it can serve the HTML, CSS, and image files used by the application.

In addition to functioning as a web server, Apache will be used as a reverse proxy. This is required because of the “Same origin policy” (http://en.wikipedia.org/wiki/Same\_origin\_policy), which requires that the HTML files and the AJAX services be provided by the same server (hostname + port). The reverse proxy can also be used for the SSL termination. See the following illustration.



The Mobile Banking installable applications can be downloaded from app stores for Apple and Android devices.

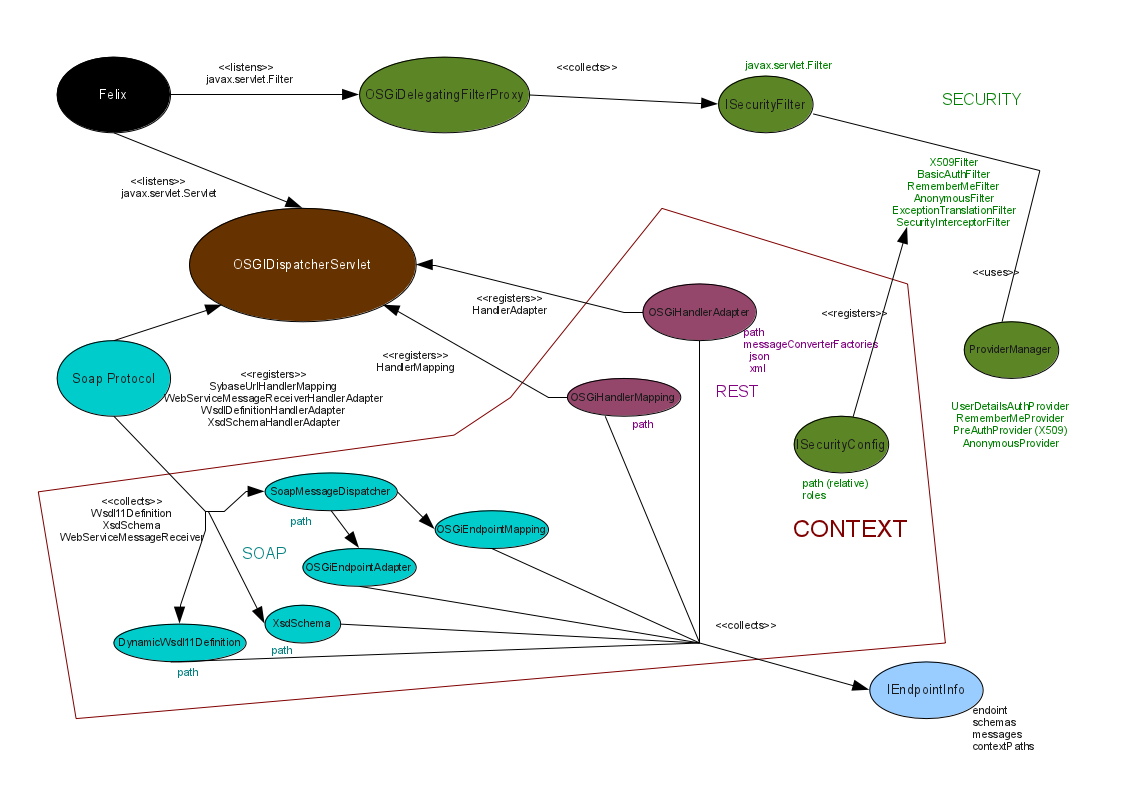
## Component Overview

The following section describes the components that make up the mBanking platform. The first section has some terms associated with the solution, and the sections below are broken down into separate components in the solution

|  |  |
| --- | --- |
| Term | Description |
| Apache Felix | Base OSGI Container |
| Apache Cordova | A framework for building installable applications from HTML, Javascript, and CSS. This is also known as ‘PhoneGap’. |
| Hibernate | Used to persist data to database |
| Maven | Tool used for buid |
| Spring Framework | Framework used for services |
| Spring-WS | Contract first web service framework which interfaces with Spring |
| SLF4J | Logging framework |

### mBanking Gateway

The mBanking gateway exposes a central registry for banking services and provides a common security layer and an audit layer. These services are deployed as an OSGi bundle and are deployed in the Felix container that is shipped with the core application. The overview of the architecture looks like:



These OSGi services are either SOAP or Restful/JSON/XML services. The SOAP services leverage Spring WS and the Rest services leverage core Spring framework. Authorization and authentication is provided leveraging Spring security.

The contract defines the request and response for a given operation. These inputs and outputs are defined by the web services provided by the bank.

### Hibernate Persistence

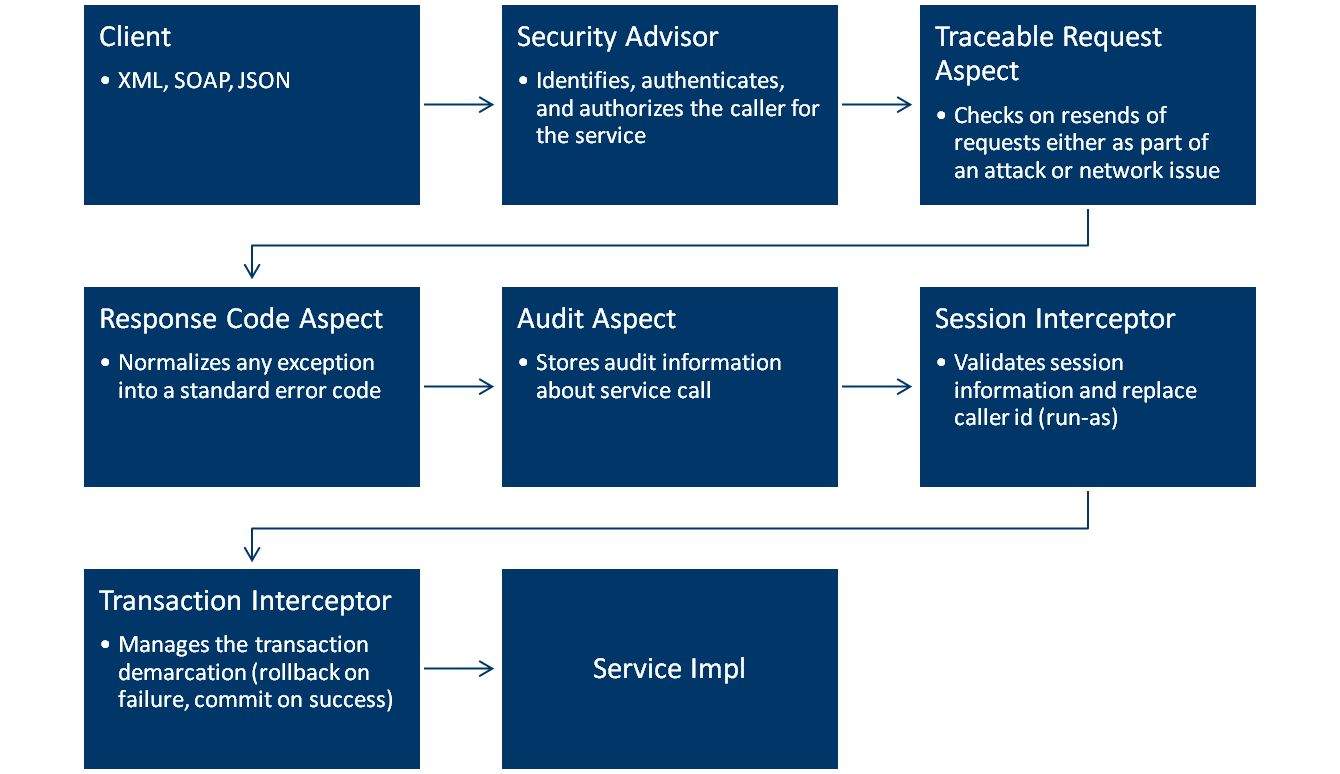
Hibernate is the persistence framework used by mBanking to store data in the mBanking database. The Hibernate persistence bundles provide the Hibernate SessionFactory and Spring PlatformTransactionManager consistently as OSGi services, such that implementations can rely on them. The actual Hibernate beans are discovered dynamically through the OSGi service registry and are defined by implementations of PersistenceServiceProvider.

### Aspects

The core Mobiliser platform provides aspects that provide functionality across modules. The core aspects include:

* SessionManagementInterceptor - This interceptor watches for MobiliserRequestTypes with a session id set. It then checks the validity of the session and updates the last used timestamp.
* SecurityAdvisor – Identifies, authenticates and authorizes the caller for the service
* TraceableRequestAspect - This method interceptor handles TraceableRequestType checking. If the exact same call has already been processed, it returns the old response found in the database. Otherwise the response is stored to the database for future processing. Usually, any request generating data should be of that type. Through this mechanism we can make sure that these requests are not processed twice, accidentally. Double-sending a request could happen because of issues on the network layer, but also due to misbehaving clients. Traceable requests must always provide a trace number (we recommend to use a UUID for that). This trace number must be unique for the last X hours (which is configurable, default 72h). If the trace number is not unique, an error is returned. Additionally, this mechanism can also be used to query for a request result a second time without actually executing the request. To achieve that, the caller must send the same request with the exact same trace number, but explicitly mark the request as a repeat request. In this case, if the caller is identical, mBanking will return the stored response from the previous call.
* ResponseCodeAdvice - This method interceptor ensures that the Status field of outgoing responses is always filled and copies the conversation id from the request into the response. It also catches any exception thrown and creates a response object and fills it with an error code.
* AuditAspect - This method interceptor stores audit information about incoming web service calls by sending relevant information to a configured instance of IAuditManager.
* SessionInterceptor - All MobiliserRequestTypes have a field sessionId which clients can use to mark the request as belonging to a previously initiated session. Endpoints must check the validity of this session value and the SessionInterceptor does just that. It also marks the last updated value of the session to now.
* TransactionInterceptor – Manages the transaction demarcation

When a request to the service comes in, this is the flow the request would take through the filter framework:



### OSGi Preferences

mBanking has a preferences interface that is similar to the core Java preferences interface, but allows for greater flexibility. There are other utility bundles that provide functionality like ‘preferences encryption’ and ‘refresh’.

### Logging

**General Logging**

SL4J (Simple Log for Java) is used to log tracing information to a log file. SL4J provides simple, fast, reliable, and OSGi-enabled logging.

The location of the logs will be in the file system in the /logs directory of the deployed Felix server. Some logging will output to mobiliser.custom.log file. The base log file is called mobiliser.log. Soap messages will be logged to a rawsoap.log file.

Tools such as splunk can monitor the files. Or you can use other ‘log tailer’ tools to watch the log files.

You can dynamically change the logging level in OSGI container by modifying the file conf/cfgbackup/rg.ops4j.pax.logging.properties and copy it to conf/cfgload folder.

In order to keep a reasonable history of logs, and in order to avoid running out of disk space, we recommend a policy of rolling and archiving log files, as determined best by the bank. An example policy might be to roll the logs once per day and delete them after 10 days.

**Masked Logging**

Masked Logging is currently available for web service request and response objects. Masking is achieved through a new custom class called MobiliserFilteringLayout. The changes to mask logging must be made in the org.ops4j.pax.logging.properties file found in the conf/cfgbackup directory.

Sensitive field includes:

* Password
* Pin
* OTP
* Security Answers
* DOB
* Consumer Bank Account Number
* Receiver Bank Account Number
* Credit Card Number
* Receiver UID

Log details can be masked at component level by making the modifications to this file.

## Bundling Structure

Refer to the Mobiliser\_R5\_Customisation guide section 2.3 for standard structure of OSGi bundles. All custom bundles must follow this pattern for consistency.

## Deployment Overview

mBanking will be deployed in the following infrastructure in production:

* OS – Redhat Linux
* Database - Oracle 11g
* Web Server and Reverse Proxy – Apache
* App Server - The mBanking core will be deployed within Felix OSGi container
* Multiple instances of the mBanking core can run simultaneously
* Multiple instances of the web server can run simultaneously
* Firewalls will be deployed as the bank determines

### Environments

The bank has talked about having the following environments, and SAP agrees that this is an ideal set of environments.

* Development – The developers will need to access the web services here, but will not deploy builds here.
* System Test – Developers will need to be able to upload and deploy builds here for system tests. The web services should function here.
* QA – This is used by the bank for their testing.
* Staging – This should be an exact replica of the production environment. Builds can be deployed here by the bank to test how they will perform in production before putting them into production.
* Production – The environment accessible by customers.

All the environments will model the production environment, but the bank may deploy single instances of each component, and the different components may all run on a single server.

### Access to Environments

The bank has agreed to provide VPN access for the SAP developers to the ‘development’ and ‘system test’ environments.

The ‘development’ and ‘system test’ environments must be accessible 24 hours per day, because some of the development will be done in India.

### Test Data

The bank has agreed to provide sample customers with sample accounts in the ‘development’ and ‘system test’ environments. Many different combinations should be included of customer types, customer banks, account types, number of accounts, etc. It should also include customers with accounts in multiple banks.

### Security Tokens

The bank has agreed to provide DigiCards and DigiPass for the developers to use for the ‘development’ and ‘system test’ environments. If this is not possible, the environments will be configured to accept hard-coded DigiCard or DigiPass numbers.

See the ‘Security’ section in this document for more information about these tokens.

## Settings

### JDBC Connection Pool

All database access will be done using the application server connection pools. These are JDBC connection pools that are configured through a properties file located on the application server file system.

The JDBC connection settings are set in the com.sybase365.mobiliser.framework.persistence.jdbc.bonecp.pool.properties configuration file.

The connection settings include:

driverClass=oracle.jdbc.OracleDriver

jdbcUrl=jdbc:oracle:thin:@todd-2b2ab9fbd0:1521:orcl

username=mobr5

password=paybox

Secure connection using SSL can be configured by changing the jdbcURL property and specifying Protocol tcps. The configuration of the connection to database will be done by Banco de Chile.

- bonecp connection pool: For best performance use 3 or 4 partitions. The minimum number of connections should be the same value as the maximum number of connections to avoid issues with idle thread eviction policies. Sizing information:

- small to medium systems (8-32 logical cores): 128 connections per partition

- large systems (>32 logical cores): 256-512 connections per partition

Ex.

-> jdbcUrl=jdbc:oracle:thin:@perf01-xor.resdev.lab:1521:orcl

-> maxConnectionsPerPartition=512

-> minConnectionsPerPartition=512

-> partitionCount=3

### JVM Settings

- JVM args are defined in the {MOBILISER\_HOME}/bin/setenv.sh script. The following settings have worked well in the past.

Heap size and management for high performance installations:

- MOBILISER\_OPTS="-Xms32G -Xmx32G"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:PermSize=256M -XX:MaxPermSize=256M"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:ReservedCodeCacheSize=128M"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:+UseLargePages -XX:LargePageSizeInBytes=2M"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:+UseParNewGC -XX:+UseConcMarkSweepGC"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:CMSInitiatingOccupancyFraction=70 -

XX:+UseCMSInitiatingOccupancyOnly"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:ParallelCMSThreads=20"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:ParallelGCThreads=40"

- MOBILISER\_OPTS="$MOBILISER\_OPTS -XX:+ExplicitGCInvokesConcurrent"27 Money Mobiliser – Installation and Setup

For more information see Mobiliser\_Platform\_Installation\_and\_Configuration\_5.1.pdf.

## Exception Handling and Error Code

### Errors from Bank Web Services

When an error happens in one of the bank’s web services, it will return an error code (or error number). The bank agrees to provide a document that lists the possible error codes and the error message text to display to users when we receive that error code.

Banco de Chile agreed to provide us with a list of error numbers and the corresponding error message (in Spanish) to display when we receive the error number from calling a web service function. However, the bank has not yet provided this for us. Instead, they have agreed that we will display a generic error message for all errors we receive from their web services. This error is:

“Servicio no disponible en este momento. Intente de nuevo más tarde.”

It is very possible that we will later find an error number for which the bank wants a specific error message. Therefore, we will include the architecture to allow it.

The architecture to support this is described below in section 9 of this document. We should use a fixed prefix key of “bankServiceError\_” followed by an error number in the MOB\_PREFERENCES database table. If no specific match is found for the error number we received from the web service, then we will use the generic error key of “bankServiceError”.

For example, the following records (plus others in the future) could be added:

|  |  |  |
| --- | --- | --- |
| **KEY** | **VALUE** | **<description>** |
| bankServiceError | Servicio no disponible en este momento. Intente de nuevo más tarde. | This is the generic error message |
| bankServiceError\_235 | La contraseña ingresada es incorrecta. | This is only an example of a message specific to return code 234 |

### Error Handling in Mobiliser mBanking

When error or failure occurs, Mobiliser mBanking will throw a specific exception to indicate the exact cause. The mBanking core will embed the deepest error code and message within its own error message. The exception is unchecked runtime exception and can be automatically handled by the ResponseCodeAspect for Mobiliser services. The error code defined by this exception will be populated into the {@link MobiliserResponseType}. In addition, the exception may carry a number of parameters providing more detail to the exception; these parameter maps will also be copied into the response object.

### Code Implementation

All custom error codes shall be defined in StatusCode.java in the package of com.sybase365.mobiliser.custom.project.businesslogic.exceptions. The starting number is 10000. Any numbers below 9999 currently are reserved for framework errors. Therefore, any error number above 10000 will indicate an error specific to the Banco de Chile project.

The mapping between error code and descriptions are defined in the MOB\_ERROR\_CODE table in the database. Here is sample SQL to add a new error:

INSERT INTO MOB\_ERROR\_CODES ( ID\_ERROR\_CODE , ID\_ERROR\_LEVEL , STR\_INFORMATION) VALUES (12000, 'INFO' , ‘General Error Code for UCD’);

Each error code in StatuCodes.java should have a corresponding entry in the MOB\_ERROR\_CODES.

# User Interface

The targeted devices for the first phase of the Banco de Chile project are:

|  |  |
| --- | --- |
| * iPhone | Installable app |
| * iPad | Installable app |
| * Android phones | Installable app |
| * Other mobile devices | Using a web browser (WAP) |

SAP will deliver installable apps for the devices listed above. The bank will submit them to the appropriate app stores.

The WAP interface is accessible from any device using a web browser. Its pages will be designed for the small screens of mobile devices.

The user interface, and the underlying application code, will be shared by all of the devices listed above. This will simplify development and maintenance. Some modifications will need to be made to make the installed applications look like native apps.

The user interface for the installed apps will be built using Apache Cordova, which is also known as PhoneGap. This is an open-source multi-platform mobile application framework which allows an installable application to be built using HTML 5, CSS 3, and Javascript. The application will run inside a web browser running on the native operating system of the device. PhoneGap provides a wrapper layer implemented using the native code for each specific supported mobile platform to provide the hooks into the mobile device features like geo-location, camera, and contact list.

The HTML, CSS, and Javascript files for the WAP user interface will be stored on Apache web servers at Banco de Chile.

## Multi-Language

The application will only be available in Spanish.

## Multi-Bank

Banco de Chile is actually composed of three different banks: Banco de Chile, CrediChile, and EdwardsCiti.

SAP will deliver separate apps for each of the three banks. The code and functionality for each of the three banks will be almost identical, but the colors and logos will change. The bank will need to send each of these different apps to the respective app store.

## Customer Segments

The bank has different groups of customers, which are called ‘Segments’. The two segments of interest to this application are VIP customers and standard customers. VIP customers have different options available to them. The application will receive the customer’s segment ID in the web service for login. The application will keep this Segment ID in the session to use to hide or show the relevant options.

## Deliverables

SAP will deliver a single ZIP file to Banco de Chile. In addition to containing the Mobiliser core, it will contain the 9 different versions of the application, which are as follows:

* An installable app for Android, styled for Banco de Chile, as an APK file.
* An installable app for Android, styled for CrediChile, as an APK file.
* An installable app for Android, styled for Citi Edwards, as an APK file.
* An installable app for iPhone and iPad, styled for Banco de Chile, as an IPA file.
* An installable app for iPhone and iPad, styled for CrediChile, as an IPA file.
* An installable app for iPhone and iPad, styled for Citi Edwards, as an IPA file.
* A WAR file containing a WAP application styled for Banco de Chile.
* A WAR file containing a WAP application styled for CrediChile.
* A WAR file containing a WAP application styled for Citi Edwards.

# Web Services for Banking

All of the banking data resides in the Banco de Chile back end.

All of the banking transactions are performed by the Banco de Chile back end.

The bank will provide web services that can be called by mBanking to query customer data and to perform banking transactions. Banco de Chile will provide documents for these web services, including the necessary WSDL files and descriptions of the parameters they require.

In order for the web services to be accessible from the different environments, the web services will specify a DNS name in their URL. Developers will need to modify their ‘hosts’ file to specify the correct IP address for these DNS names.

# Communication Design

## SSL Encrypted Communication

Any communications passing over the internet is required to be secured and encrypted. Such communication paths will use HTTPS / SSL to pass information between systems.

## Interfaces

Mobiliser provides the following interfaces:

* SOAP
* JSON/XML
* Internal Service Call

# Customer Messaging

There are points in the application flow were SMS messages or email messages need to be sent to customers. These messages will be sent by the bank by the application by calling web service functions provided by the bank.

In the development environments, the bank should provide a way to test that SMS and email messages are sent, and a way to see the contents of these messages.

SAP will store templates in the mBanking database for the different types of messages to be sent to customers. These templates will be stored in the database in order to allow the bank to change them without requiring an update to the application.

Here is an example of a message template in the database:

“Estimado %1, su debito de %2 a la cuenta %3 fue existoso.”

In this example, %1 represents the customer name, %2 represents an amount, and %3 represents an account number.

The same is true for any ‘terms and conditions’ text to be displayed to customers. The text must be stored in the database.

The same is also true for ‘help’ text.

There are cases where the ‘terms and conditions’ text is stored in a PDF file. In this case, the URL for the file containing the text will be stored in the MOB\_PREFERENCES database table.

Records will be added to the MOB\_PREFERENCES table for each case where we (or the bank) determines that there is a message that may change in the future. There is no need to store every string that we display to customers. The application will have to hard code the KEY value for the text it wants to look up in the database. The KEY values can be anything, as long as they are unique in the table.

If the application wants to use a string as a template containing ‘variables’, the application needs to know the variables and their meanings.

Continuing the example above, a record could be added as follows:

KEY: “debitSuccessful”

VALUE: “Estimado %1, su debito de %2 a la cuenta %3 fue existoso.”

Help text is done in the same way. For example, the following records could be added:

|  |  |
| --- | --- |
| **KEY** | **VALUE** |
| bankPrimaryPhone | +56 2 26373737 |
| bankEmergencyPhone | +56 9 123 45 67 |

Terms and conditions text will be stored in the same way. There will be other cases where we will call a web service function where the bank will generate a PDF file for terms and conditions. In these cases, the web service will return the URL to the PDF file. In other caces, the terms and conditions text may be in a PDF file and we will store the URL to the file. For example:

|  |  |
| --- | --- |
| **KEY** | **VALUE** |
| loanTerms | Lorem ipsum stuff you must accept . . . |
| mortgageTerms | http://192.168.23.12/mortgageterms.pdf |

# Security

The installable mBanking apps will not store any personal or bank data on the mobile device. The customer’s RUT number will be stored in our database, but not on the mobile device. The only information that we will store on the customer’s local device is:

* The device ID that we generate (See section 13.2)
* The customer’s first name

All communications between the device and the application server will be done through SSL.

Some banking operations, typically any operation where money is moved, require the customer to enter a second factor of authentication. mBanking will support the DigiCard (bingo card) or DigiPass (token device). The application must call a web service function to validate the value entered by the customer. A separate web service function is used to query the DigiCard coordinates that the customer must enter.

The application server and web server at Banco de Chile should be protected by firewalls.

# Customizations to Database

## Customizations for Banco de Chile

For the Banco de Chile project, we want to store as little information as possible in the mBanking database. The primary data repository is the bank’s own databases. Only the mBanking tables listed below will be used.

The mBanking ‘audit’ tables will be used extensively. See the ‘Auditing’ section below.

The MOB\_PREFERENCES table will be used for messages that will be displayed to customers. See section 9 above.

A new ‘enrollment’ table will be added, where we will store information used during the customer enrollment process which is described below.

* Customer RUT
* Date of enrollment
* Device Id
* Device Operating System
* Device Model
* Device Name (Entered by user) (Can be null)
* Device Phone Number (Can be null)
* Bank Id

Some other minor tables may be used, such as to store information displayed in drop-down lists for such things as currency codes, city names, country names, etc. Not for customer information.

## Best Practices

In order to provide consistency and simplicity, there are some best practices when making database modifications or additions.

Restrict the possible field types to:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| XSD-Type | ASE | DB2 | Oracle | PostgreSQL |
| idShort | numeric(5) |  | NUMERIC(5) | NUMERIC(5) |
| idLong | numeric(18) |  | NUMERIC(18) | NUMERIC(18) |
|  | number(38,19) | NUMERIC(29,9) | NUMERIC(32,19) | NUMERIC(32,19) |
|  | char(1) | CHAR(1) | CHAR(1 BYTE) | CHAR(1) |
|  | char(2) | CHAR(2) | CHAR(2 BYTE) | CHAR(2) |
|  | char(3) | CHAR(3) | CHAR(3 BYTE) | CHAR(3) |
| strShort | varchar(6) | VARCHAR(6) | VARCHAR2(6 CHAR) | VARCHAR(6) |
| strMedium | varchar(80) | VARCHAR(80) | VARCHAR2(80 CHAR) | VARCHAR(80) |
| strLong | varchar(200) | VARCHAR(200) | VARCHAR2(200 CHAR) | VARCHAR(200) |
| strHuge | varchat(2048) | VARCHAR(2048) | VARCHAR2(2048 CHAR) | VARCHAR(2048) |
| dateTime | datetime | TIMESTAMP | TIMESTAMP | TIMESTAMP |

* The table names for a customization project should all start with a fix 3 character prefix plus an underscore character to identify them more easily. The following ones are already reserved for internal usage: AMS\_, MOB\_, EVT\_, MSG\_, QTZ\_. E.g. if we are adding additional customer data, CUS\_MOB\_CUSTOMER. The CUS\_ identifies this as a custom extension
* We do not drop any columns from existing tables, and we avoid making any modifications to existing tables. This provides for a smoother upgrade path.
* Column names must start with
  + ID\_ for NUMBER, CHAR(1 BYTE), CHAR(2 BYTE) and CHAR(3 BYTE) types
  + BOL\_ for CHAR(1 BYTE) type
  + STR\_ for VARCHAR2 types
  + DAT\_ for TIMESTAMP types
* Add comment to explain the purpose of the field.
* Each table must include the columns DAT\_CREATION, ID\_CUSTOMER\_CREATION. Each table that also gets updated (not extending DbEntry or NoneUpdatableGeneratedIdEntry) needs to include the additional columns DAT\_LAST\_UPDATE and ID\_CUSTOMER\_LAST\_UPDATE.
* All scripts file names must be ordered (001, 002, ….)
* The scripts must not contain a commit
* The scripts must never be changed (once executed successfully)
* Custom scripts will be kept in known location in source control

## Persistence Layer

When adding persistence code, the following steps should be performed:

* Create an entity class in dao-model package using only JPA compatible annotations. The reason why we want to keep it JPA compatible and not use Hibernate specific annotations is if we decide to change the implementation in the future.
* Create a new DAO interface in dao-api extending a sub-interface of BaseDAO.
* Add a new DAO interface getter and setter to the DAOFactory interface. Make sure to define the encapsulated entity class and its primary key class as generic arguments.
* Create new DAO hibernate implementation in dao-hibernate of interface which extends a sub-class of BaseDAOHbnImpl.
* Enhance HibernateDaoFactoryImpl and add a new class variable for the DAO interface as well as a set of getter and setter methods.
* Add to Spring context
* Configure DAO Factory in Spring context to inject new instance of implementation.

# Drop-down lists

The core Mobiliser system has a mechanism for storing data used to populate UI elements and other functions. This data is typically non-sensitive data such as country, state and currency lists. This service call maps the entity name in the request to the MOB\_LOOKUP\_QUERIES.ID\_QUERY field. The code then runs the MOB\_LOOKUP\_QUERIES.STR\_QUERY and returns the ID, Name values as look up entities.

# Customer Identification

## RUT number

The primary customer identifier is called the customer ‘RUT’. The RUT is the tax-payer ID number used by the government of Chile. Every customer of Banco de Chile must have a RUT number, so it is used as the primary ID for customers.

RUT numbers in Chile follow some basic formatting rules, and must match a defined checksum. Here is a sample Javascript function to validate a RUT. This function needs improvement, but is provided here only as an example.

function CheckRut(Object)

{

var sTemp = "";

var sRutAux = "";

var sDV; //string for checksum digit

var iLength = Object.value;

if (iLength.length == 0)

return false;

sRut = Object.value

iRutLength = sRut.length;

if (iRutLength < 2) //at least 2 characters length

{

alert('Rut invalid');

Object.focus();

return false;

}

for ( i=0; i < sRut.length ; i++ )

{

if (sRut.charAt(i) != ' ' && sRut.charAt(i) != '.'

&& sRut.charAt(i) != '-')

{

sTemp = sTemp + sRut.charAt(i);

}

}

sRutAux = sTemp;

sRut=sTemp;

iRutLength = sRut.length;

if ( iRutLength > 2 )

sRutAux = sRut.substring(0, iRutLength - 1);

else

sRutAux = sRut.charAt(0);

sDV = sRut.charAt(iRutLength-1);

if ( sRutAux == null || sDV == null )

return 0;

var dvr = '0';

suma = 0; //var to sum

mul = 2; // checksum factor

for (i= sRutAux.length-1 ; i >= 0; i–)

{

suma = suma + sRutAux.charAt(i) \* mul;

if (mul == 7)

mul = 2;

else

mul++;

}

res = suma % 11; //mod 11 formula

if (res==1)

{

dvr = 'k';

}

else if (res==0)

{

dvr = '0';

}

else

{

dvi = 11-res;

dvr = dvi + "";

}

if (dvr != dv.toLowerCase())

{

alert('RUT invalid');

Object.focus();

return false;

}

alert(‘RUT is OK!’);

Object.focus();

return true;

}

There is also a JQuery plugin available, if it makes sense to use it, at this URL: <http://joaquinnunez.cl/jQueryRutPlugin/>

## Device ID

During the enrollment process (described below), we need to store the device’s unique ID to the database. There are three purposes for using this ID:

1. To ensure that a single device is only used by a single customer.
2. To allow us to look up a customer’s RUT number based on the device being used.
3. To know if a device is already enrolled.

We have learned that Apple rejects apps that attempt to retrieve and use the Unique   
Device ID for devices, so we will not be using that as our device ID. For our purposes, we only need an ID that is unique to each device, so any Universally Unique Identifier (UUID) will suffice. Therefore, during the enrollment process, we will generate a unique ID and store that in the device’s local storage.

We will use the ‘UUID’ Java class to generate this device ID.

Whenever a customer starts the application on their device, we will look in the device local storage to see if a device ID is stored.  If a device ID is present, we assume the device has been enrolled (There are exceptions to this.  See below).  If no device ID is present, we know it is not enrolled, so we prompt the customer to enroll.

During enrollment, we generate a new globally unique device ID.  This ID is stored in the device local storage and also added to the ‘enrollment’ table in our database.

It is possible that during app start-up, a device ID is in local storage, but the device has been un-enrolled from another device.  In this case, we will assume that the device is enrolled and prompt the user to log in.  When the user attempts to log in, the application back-end will attempt to look up the customer’s username using the device ID sent from the device, and no match will be found, and a ‘not found’ error will need to be returned to the front-end.  We need to add code to the application front end to prompt to customer to re-enroll at that time.

A new device ID is generated every time a device is enrolled.  We don’t try to detect if a device was previously enrolled and un-enrolled.

Here is an example:

Let’s use the case where I have 2 iPads: iPad1 and iPad2.  And my user id is “JRojas”.

* I enroll iPad1, and let’s say that we generate a device ID of “AAA” at that time.
* Then I enroll iPad2, and we generate device ID “BBB”.
* In the Mobiliser database, we now have two enrollment entries: [JRojas, AAA] and [JRojas, BBB].
* Now I delete the app from iPad1.  It doesn’t matter if I un-enroll the device or not.
* Now I re-install the app and launch it.  Since the app was previously uninstalled, no device ID is stored on the device, so it says I need to enroll the device.
* So I enroll iPad1, and I get a new device ID of “CCC”.
* On the application back end, we store a new record [JRojas, CCC].
* If the customer previously un-enrolled before un-installing the application, we would have removed the record with [JRojas, AAA], but if not, then the [JRojas, AAA] record is orphaned.  But it is no big deal to leave it there.
* The next time we launch the app on iPad2, for example, we will pass the device ID of “BBB” that is stored on the device, and the application backend will look up the record for “BBB” and find the username of “JRojas”.

If a customer uninstalls our app on a device without un-enrolling the device first, a record for that device will remain in our ‘enrollment’ database. Customers can later clean out these old records by using the ‘configuration’ page in the application.

In order to help customers identify which devices have been enrolled, when they are using the ‘configuration’ page, we will store the device information in the ‘enrollment’ database table. We will also ask customers to enter a ‘device name’ during enrollment.

We will also store the device’s phone number in our enrollment table. We understand that some devices, like iPads, do not have a phone number, but storing a phone number is helpful for those devices that do have a phone number. If a customer enrolls a device, and we find a device in our database with the same phone number, we will overwrite the existing enrollment with that phone number.  It is not allowed to enroll two devices with the same phone number.

When customers visit the ‘configuration’ page of the application, they should be able to see all of the following information about their enrolled devices:

* Device Name
* Enrollment Date
* Phone Number
* Device Type

It is not a real problem if a customer mistakenly un-enrolls a device that is still being used.  As described above, we will prompt them to re-enroll in this case.

# Customer enrollment

Banco de Chile requires that a customer enroll his/her device before being able to use it for banking transactions. The enrollment process helps to ensure that the device being used belongs to the customer.

During enrollment, data will need to be stored in a new ‘enrollment’ table in the mBanking database, as described in the ‘Customizations to Database’ section of this document.

A customer can be enrolled with multiple devices, so the primary unique key is the combination of the RUT and deviceId fields. As described above, the device ID will be generated by our application at the time of enrollment.

A customer can enroll on a device that was previously enrolled for a different customer, to handle the case where a customer failed to un-enroll before selling the device to a different customer. If a customer enrolls on the same deviceId as an existing enrollment with a different RUT, the existing enrollment should be overwritten.

In order to make installable applications startup quickly, the app should not need to communicate with the app server to determine if a customer has already enrolled on the device. A simple flag will be stored in the device local storage which will indicate that a customer is enrolled. During app startup, if the local flag is set, it will show the ‘login’ page. Otherwise it will show the ‘enrollment’ page.

Since there will be a separate app in the app store for each of the three component banks, the enrollment process must not allow a customer to enroll using an app that does not correspond to his/her bank. For example, if the customer downloads the app for CrediChile, the app will call the ‘validate password’ web service function after the customer enters his/her password. That function will return customer information, including the ‘primary’ bank of the customer (because some customers have accounts in more than one of the component banks). If that information indicates that the customer’s ‘primary’ bank is not CrediChile, an error message should be displayed indicating that the customer should install a different app.

Each customer enrollment should be written to the ‘audit’ database.

The customer is also allowed to un-enroll by selecting a menu option. In this case, the existing enrollment record will be deleted, and an audit entry will be saved.

Once enrollment is successful, we will store the customer’s first name to the device local storage. This will allow us to display the customer’s first name in the user interface before the customer logs in.

There is no enrollment process for the WAP client. When launching the WAP client, the ‘login’ page will always be the initial page displayed to the customer.

As specified in the enrollment use case, customers must have a cell phone registered with the bank before they can enroll. They must have that cell phone with them at the time of enrollment even if they are enrolling a device that has no phone number, like an iPad.

# Customer login

When a customer logs in through the user interface, the app will call a web service function that will return information about the customer and the customer’s accounts. This information will be saved in the session, but not stored to the mBanking database.

The user interface for login will be different for the WAP client than for the installed apps.

The installed apps will not prompt users to enter their RUT number—only their password. In this case, after the user enters the password and submits the page, the back-end application will receive the password and the device ID. It will then look up the customer’s RUT number from the database and use that to call the required web service function.

The WAP interface, since there is no enrollment process for WAP, will prompt customers to enter their RUT number and their password.

When calling the ‘login’ web service, the password must be encrypted using a custom encryption method used by the bank. The bank will provide Java code to perform this encryption.

Based on the information returned from the login web service, different menu options will be available to the customer. Other menu options may appear later as the result of calling other web service functions. This requires that the main menu page use AJAX to call web service functions and display new options without refreshing the page.

The following customer information will be stored in the customer session for use through the session:

* Segment ID
* List of Accounts
* Email address
* Cell phone number

Each customer login event should be written to the ‘audit’ database.

# Auditing

Banco de Chile requires that many customer actions be logged to audit table(s) in the database.

The database table in the Mobiliser mBanking database is named MOB\_AUDIT\_LOGS. No modification to this table will be required.

The bank requests that the following data be stored for each audit entry. These are the only columns of the table that we will use:

|  |  |
| --- | --- |
| **Data** | **Database Column** |
| Mobiliser Customer ID | ID\_CUSTOMER |
| Date & Time | DAT\_CREATION |
| Action | STR\_ACTION |
| Device ID (if not using WAP) | STR\_DEVICE\_ID |
| Result Code (success or failure) | STR\_ACTION\_RESULT\_CODE |
| Customer ID (RUT) | STR\_PARAMETER\_1 |
| Source account number | STR\_PARAMETER\_2 |
| Destination account number | STR\_PARAMETER\_3 |
| Amount | STR\_PARAMETER\_4 |
| Additional context | STR\_PARAMETER\_5 |
| Transaction ID | STR\_PARAMETER\_6 |
| Type of authentication device used | STR\_PARAMETER\_7 |

Note that ID\_CUSTOMER, is the key of the customer in the MOB\_CUSTOMERS table in the Mobiliser database.

Note that in the case where customers are using the WAP channel, no Device ID will be available. Leave STR\_DEVICE\_ID null in that case as an indicator that the customer was using the WAP channel.

Note that we need to audit all of the required transactions, whether they succeed or fail. In the cases where the transaction succeeds, we will save zeros to STR\_ACTION\_RESULT\_CODE. In the cases where the transaction fails, we will save the non-zero result code received from the bank’s web service. Also, in the event that the transaction fails, no Transaction ID will be available, so null will be stored to STR\_PARAMETER\_6.

For STR\_PARAMETER\_7, we need to store some consistent value that indicates the type of authentication device used for the transactions where an authentication device is required. For example, “digiCard”, “digiPass”, “Emergency Key”.

Below is a list of the activities that we will log, and a description of the audit data we will store.

**Enrollment**

|  |  |
| --- | --- |
| STR\_ACTION | “ENROLL” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | (not used) |
| STR\_PARAMETER\_7 | (not used) |

Be sure to populate STR\_DEVICE\_ID with the ID of the newly created entry in the BDC\_ENROLLMENTS table.

**Un-enrollment**

|  |  |
| --- | --- |
| STR\_ACTION | “UNENROLL” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | (not used) |
| STR\_PARAMETER\_7 | (not used) |

Be sure to populate STR\_DEVICE\_ID with the ID of the device being un-enrolled.

**Login**

|  |  |
| --- | --- |
| STR\_ACTION | “LOGIN” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |
| STR\_SESSION\_ID | If a Mobiliser session ID is available for the session being started. |

**Logout**

|  |  |
| --- | --- |
| STR\_ACTION | “LOGOUT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | (not used) |
| STR\_PARAMETER\_7 | (not used) |
| STR\_SESSION\_ID | If a Mobiliser session ID is available for the session being ended. |

**Terms and Conditions Accepted (This is not used for this phase of the project)**

|  |  |
| --- | --- |
| STR\_ACTION | “TC\_ACCEPT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Some explanation of the context. For example, “Terms for a new loan” or “Terms for alerts”. |
| STR\_PARAMETER\_6 | (not used) |
| STR\_PARAMETER\_7 | (not used) |

**Transfer Funds between accounts**

|  |  |
| --- | --- |
| STR\_ACTION | “TRANSFER\_BETWEEN\_ACCOUNTS” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Destination account number |
| STR\_PARAMETER\_4 | Amount transferred |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Transfer Funds to third party**

|  |  |
| --- | --- |
| STR\_ACTION | “TRANSFER\_THIRD\_PARTY” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Destination account number |
| STR\_PARAMETER\_4 | Amount transferred |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Credit Card Advance**

|  |  |
| --- | --- |
| STR\_ACTION | “CREDIT\_CARD\_ADVANCE” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source card number |
| STR\_PARAMETER\_3 | Destination account number |
| STR\_PARAMETER\_4 | Amount advanced |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Add Recipient**

|  |  |
| --- | --- |
| STR\_ACTION | “ADD\_RECIPIENT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | Recipient name or account number |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Recipient email address |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Payment Made to Credit Line**

|  |  |
| --- | --- |
| STR\_ACTION | “PAYMENT\_CREDIT\_LINE” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Payee identifier |
| STR\_PARAMETER\_4 | Amount paid |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Payment Made to International credit card**

|  |  |
| --- | --- |
| STR\_ACTION | “PAYMENT\_CREDIT\_CARD\_INTERNATIONAL” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Payee identifier |
| STR\_PARAMETER\_4 | Amount paid |
| STR\_PARAMETER\_5 | ‘billed amount’, ‘minimum amount’, or ‘other amount’ |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Payment Made to national credit card**

|  |  |
| --- | --- |
| STR\_ACTION | “PAYMENT\_CREDIT\_CARD\_NATIONAL” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Payee identifier |
| STR\_PARAMETER\_4 | Amount paid |
| STR\_PARAMETER\_5 | ‘billed amount’, ‘minimum amount’, or ‘other amount’ |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Payment Made to Service Account**

|  |  |
| --- | --- |
| STR\_ACTION | “PAYMENT\_ACCOUNT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Payee identifier |
| STR\_PARAMETER\_4 | Amount paid |
| STR\_PARAMETER\_5 | Business or service name |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**RedGiro (send money to ATM)**

|  |  |
| --- | --- |
| STR\_ACTION | “REDGIRO” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | Amount sent |
| STR\_PARAMETER\_5 | Recipient cell phone number |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Invest in Time Deposit**

|  |  |
| --- | --- |
| STR\_ACTION | “INVEST\_TIME\_DEPOSIT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Time deposit account number |
| STR\_PARAMETER\_4 | Amount deposited |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Balance Inquiry (for account)**

|  |  |
| --- | --- |
| STR\_ACTION | “BALANCE\_INQUIRY\_ACCOUNT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | account number |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**Balance Inquiry (for credit card)**

|  |  |
| --- | --- |
| STR\_ACTION | “BALANCE\_INQUIRY\_CARD” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | credit card number |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | (not used) |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**Balance Inquiry (for investments)**

|  |  |
| --- | --- |
| STR\_ACTION | “BALANCE\_INQUIRY\_INVESTMENT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | account number |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Number of items in response |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**Transaction History Request (for accounts)**

|  |  |
| --- | --- |
| STR\_ACTION | “TRANSACTION\_HISTORY\_ACCOUNT” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | account number |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Number of transactions in response |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**Transaction History Request (for credit cards)**

|  |  |
| --- | --- |
| STR\_ACTION | “TRANSACTION\_HISTORY\_CARD” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | credit card number |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Number of transactions in response |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**Cell Phone Top-Up**

|  |  |
| --- | --- |
| STR\_ACTION | “TOP\_UP” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Source account number |
| STR\_PARAMETER\_3 | Telephone number |
| STR\_PARAMETER\_4 | Amount |
| STR\_PARAMETER\_5 | Carrier ID |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**Take Loan**

|  |  |
| --- | --- |
| STR\_ACTION | “TAKE\_LOAN” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | Loan account number (if available) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | Amount |
| STR\_PARAMETER\_5 | Pre-approved amount |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | Type of authentication device |

**View Received Transfers**

|  |  |
| --- | --- |
| STR\_ACTION | “RECEIVED\_TRANSFERS” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Number of transfers in response |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**View Sent Transfers**

|  |  |
| --- | --- |
| STR\_ACTION | “SENT\_TRANSFERS” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Number of transfers in response |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**Request List of Alerts**

|  |  |
| --- | --- |
| STR\_ACTION | “QUERY\_ALERTS” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | Number of alerts in response |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

**‘Contact’ request submitted (using the Contacto page)**

|  |  |
| --- | --- |
| STR\_ACTION | “CONTACT\_SUBMITTED” |
| STR\_PARAMETER\_1 | Customer RUT number |
| STR\_PARAMETER\_2 | (not used) |
| STR\_PARAMETER\_3 | (not used) |
| STR\_PARAMETER\_4 | (not used) |
| STR\_PARAMETER\_5 | First 80 characters of the message |
| STR\_PARAMETER\_6 | Transaction ID |
| STR\_PARAMETER\_7 | (not used) |

SAP will not provide a user interface to view the audit information stored. They will write their own queries or reports to view this data. SAP will provide some sample SQL to query the database.

# Session Management

The core Mobiliser system holds no state, but the framework supports the concept of sessions for channels that do not hold state.

Once a user is authenticated, the calling services should pass in the mBanking session with requests. The [SessionManagementInterceptor](#sessioninter) will validate the session then update the last active timestamp.

The core CustomerLogic Business Logic class includes a method named checkSessionForExpiration.

For this project, session management for installed apps and the WAP interface is done in Javascript. The file named app.js includes this logic.

It appears that Mobiliser requires that a ‘customer’ object exist in the database in order to maintain a session for that customer. Therefore, we will create minimal customer records in the Mobiliser database as necessary when customers log in. We will use the customer’s RUT number as the customer’s name in the STR\_DISPLAY\_NAME field of the table.

We had discussed using the device ID as the customer’s primary identifier, but a customer can have multiple devices, so it is better to use the RUT number, which is unique to each customer.

Here are the columns we will use in the MOB\_CUSTOMERS table. Note that all of the columns except for ID\_CUSTOMER and STR\_DISPLAY\_NAME) will have the same hard-coded value. All other columns not listed here will be set to NULL.

* ID\_CUSTOMER (auto-generated index number)
* BOL\_IS\_ACTIVE (set to “Y”)
* BOL\_IS\_TEST (set to “N”)
* ID\_ORGUNIT (set to “0000”)
* ID\_BLACKLISTREASON (set to “0”)
* STR\_DISPLAY\_NAME (Set to the RUT number of the customer)
* ID\_LANGUAGE (Set to “es”)
* ID\_CUSTOMER\_TYPE (set to “2”)
* ID\_CANCELLATION\_REASON (set to “0”)
* ID\_NOTIFICATION\_MODE (set to “0”)
* ID\_CUSTOMER\_CREATION (set to “0)

Add an index to MOB\_CUSTOMERS on the STR\_INDEX field to speed look-ups.

# For example, when a customer logs in, we will look for an existing record in the MOB\_CUSTOMERS table where STR\_DISPLAY\_NAME matches the RUT number matching the RUT number of the customer logging in. If there is no match found, we will create a new record in the table and use that record in the customer’s session. If we find a match, we will use that existing customer record for the session.Privileges

For the Banco de Chile project, all privileges are controlled by the bank back end system. If a customer attempts to perform a transaction that the customer is not allowed to do, the bank web services will return an error code.

Some privilege checking will be done by the mBanking application, based on customer attributes. For example, a customer with only one account is not allowed to perform a transfer. Also, some additional features are only accessible by VIP customers, which are determined by the customers’ ‘segment’. These additional checks are documented in the ‘use case’ document.

# Appendix A – Guidelines for Web Services

**Guideline for Integrating to Third-party Web Services and Exposing Web Services**

1. **Integrate to third-party web services**
   1. Create a new package com.sybase365.mobiliser.custom.project.touchpoinst.NEWNAME
   2. Copy all files from com.sybase365.mobiliser.custom.project.touchpoinst.weather into the new package except the WeatherTargetSource.java
   3. Add your WSDL file inside the src/main/resources folder
   4. Change “weather” to “new name” where-ever it is necessary
   5. Modify pom.xml: points to your own WSDL file
   6. In your new package, run “mvn clean install”. This will generated all stub java files from WSDL
   7. Copy over WeatherTargetSource and rename it to XXXTargetSource.java:
      1. change it to reference your own service interface (The interface name usually is the same as the name specified inside <wsdl: portType name=”THE\_INTERFACE\_NAME” /> tag in the WSDL file
      2. Specify jaxWsProxyFacotry’s properties suas as jaxWsProxyFactory.setServiceName(), etc.
      3. Specify the endpoint address using preference node:

node.get("KEY", “WSDL\_URL”), where

KEY is the value specified in str\_name column in the MOB\_PREFERENCES table, please use the convention like “service.NEWNAME.url”; An example: *“service.customer.regis.url”*

WSDL\_URL is the default value in case it is not defined in the MOB\_PREFERENCES table for the key; An example: <http://172.31.8.221:8180/CustomerRegistration/services/CustomerRegistration_v1_0>

* + 1. Created the db script for the service url. (You need to save the credential to access the url in the preference table too)
    2. Save your database script and add to dbMaintain scripts
  1. Recompile your package using “mvn clean install”
  2. Add Spring bean for your touch point in bundle-context.xml:

<!-- Create preference node -->

<prefs:node id="prefNode"

class="com.sybase365.mobiliser.custom.project.touchpoints.ucd.UcdTargetSource"

refreshable-beans="ucdTargetSource" auto-decrypt="true" />

<!-- Setup a target source to generate the instance based on the prefs -->

<bean id="ucdTargetSource"

class="com.sybase365.mobiliser.custom.project.touchpoints.ucd.UcdTargetSource">

<constructor-arg ref="prefNode" />

</bean>

<!-- Create a proxy which will be handed out to client code which knows nothing of preferences -->

<bean id="ucdTouchpint" class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="targetSource" ref="ucdTargetSource" />

<property name="interfaces">

<list>

<value>wu.eh.dis.ucd.wsdl.activatedeactivate\_v1.ActivateDeactivateV10</value>

</list>

</property>

</bean>

* 1. Export your touch point as OSGI service in bundle-context-osgi.xml:

<osgi:service ref="ucdTouchpoint" interface="your\_path.your\_interface"

context-class-loader="service-provider">

</osgi:service>

* 1. Import the touch point in other package such as business logic to use it:

<osgi:reference id=*"ucdTouchpoint"*  interface=*"your\_path.your\_interface"* />

* 1. In pom.xml, make sure the stub classes are exported:

<Export-Package>YOUR\_PATH </Export-Package>

1. **Expose web service**
   1. In services.contract package, find the related existing XSD file or create a new XSD file, add 4 new objects: XXXRequestType, XXXResponseType, XXXRequest, XXXResponse. Make XXXRequest extends from XXXRequestType
   2. Create java methods in services.endpoint package. The java file name should be like “somethingEndpoint.java”
   3. Export your endpoint to OSGI server in bundle-context-osgi.xml
   4. Give permission to allow request and response pass through in bundle-context-{specific}.xml (for example, bundle-context-customer.xml or bundle-context-data.xml)
   5. Compile and deploy the services package/bundles, you will see your service exposed in the designated WSDL