NTTコミュニケーションズ APIゲートウェイ連携 サンプルWebアプリ活用ガイド

第1版 2015/06/11

1 目次

2 はじめに

「NTTコミュニケーションズ APIゲートウェイ(以下APIゲートウェイ)」とはNTTCom各サービスの申込みから保守運用までのビジネスプロセスに関する各情報へのアクセスを一元的に提供するAPIプラットフォームである。

本書ではこのAPIゲートウェイを活用するために作成されたデモアプリについて、その構成や環境構築手順ならびにカスタマイズ方法について述べている。このデモアプリをカスタマイズすることで、図2-1のように業務や自社システムにより最適なサービスを選択し導入することが可能である。

なお、デモアプリではRubyOnRailsやNode.jsなどの技術を利用している。環境構築時やカスタマイズ時にはこれらの技術について一定の知識を持っている必要がある。

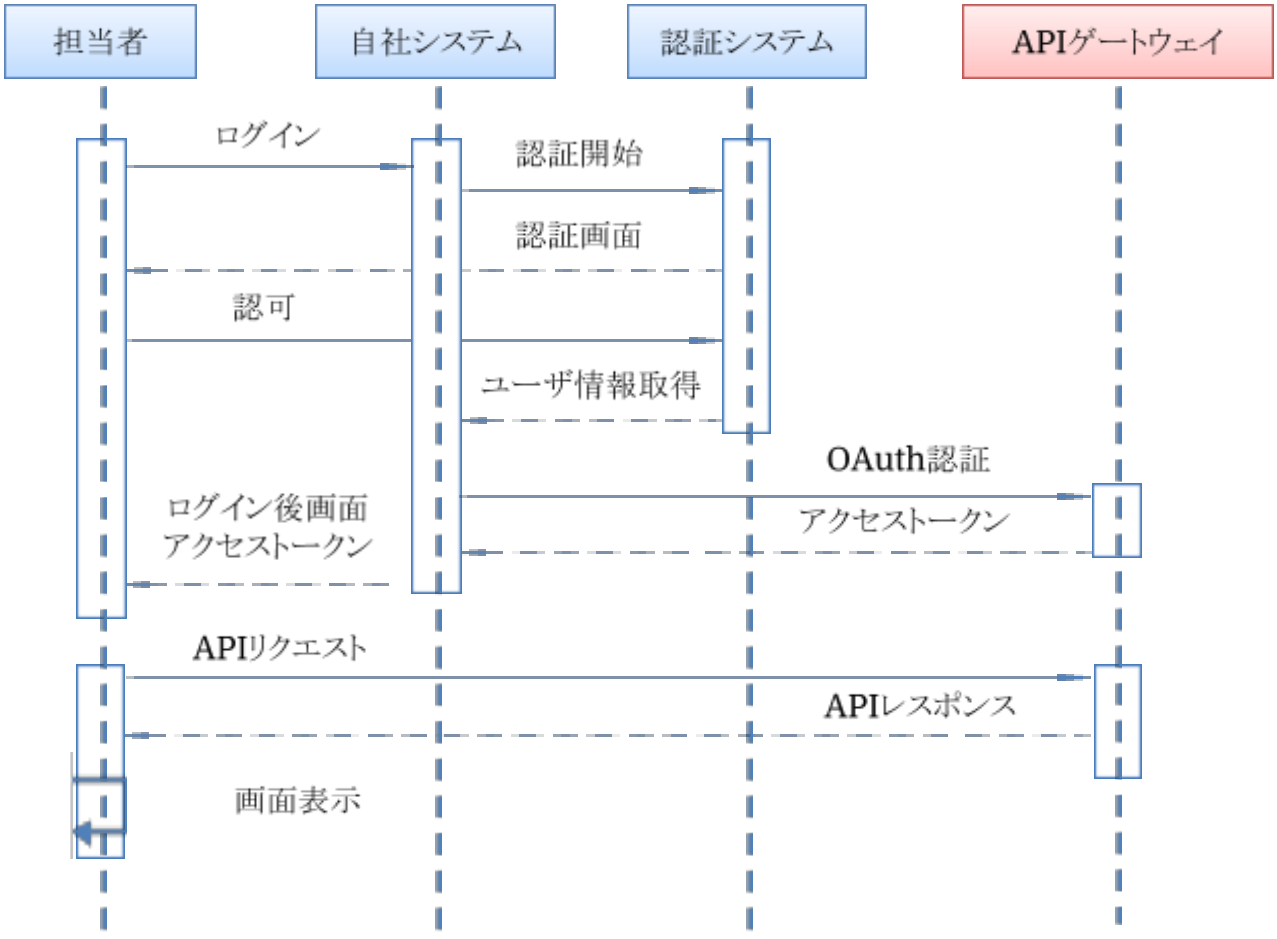


図2-1 APIゲートウェイ利用例

3 デモアプリ概要

3.1 システム構成

APIゲートウェイを利用するデモアプリの概要は以下の通りである。

- サーバアプリ(認証サーバ)

ソーシャルログイン機能を提供し、認証ユーザへAPIゲートウェイから取得したアクセストークンを提供する。ユーザ管理を行わない代わりに、アカウントごとに認証キーを変更可能な仕組みを実装している。

- クライアントアプリ

サーバアプリから提供されたアクセストークンを利用し、JavaScriptの非同期通信によりBusiness ProcessAPIから各種サービスの情報を取得しブラウザ上に表示する。JavaScriptのアプリケーションフレームワークAngularJSにて実装されている。

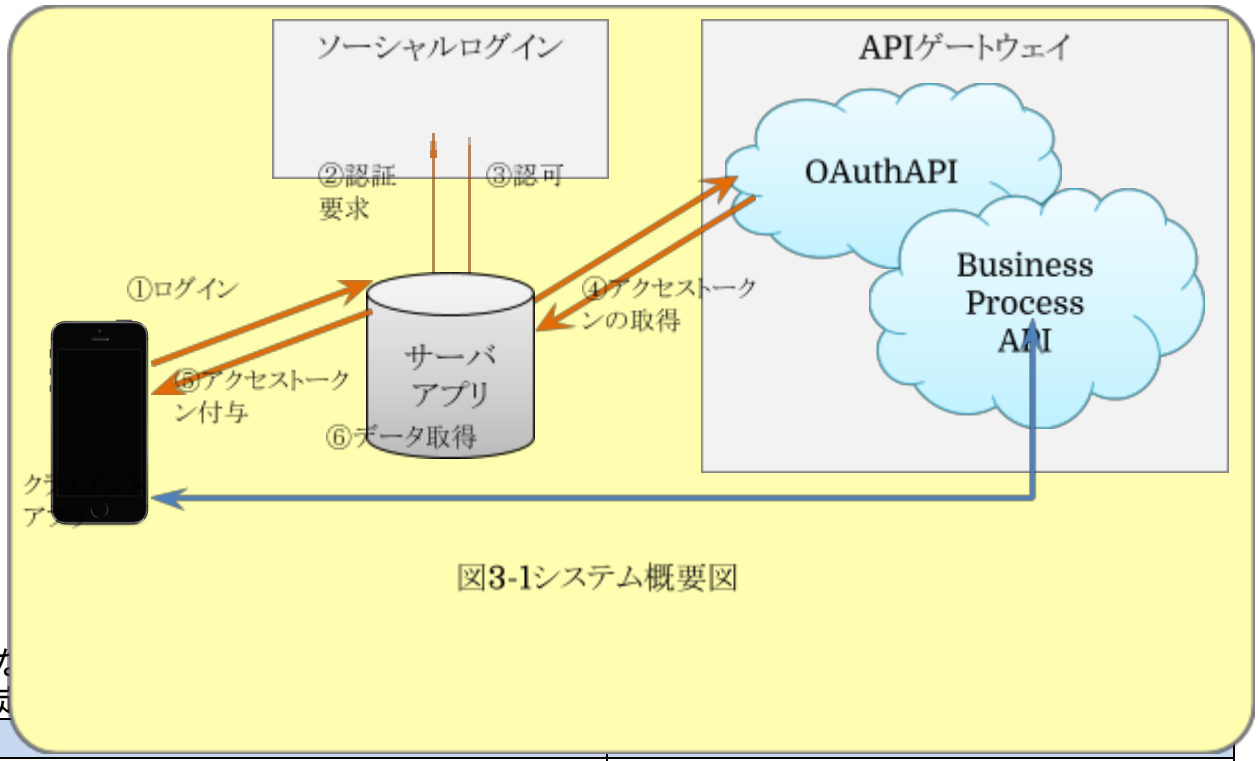


図3-1システム概要図

APIゲートウェイとの連携を組み込んだ自社システム	担当者がサービスの申込みや運用保守情報の参照など業務に利用するアプリ

3.2 クロスドメインリクエストについて

デモアプリではHTML(クライアントアプリ)を認証サーバから読み込んだあと、以降はブラウザがJavaScriptを利用し、Business ProcessAPIへ直接リクエストを行い各種データを取得する仕組みになっている。このような場合、HTMLを生成した認証サーバと、実際に通信を行う先(APIゲートウェイ)が異なるため、クロスサイトスクリプティングを防止するブラウザの仕様により通常では通信が行えないようになっている。

このようなクロスドメインリクエストを実現出来るように、JSONPという手法や

CORS(Cross-Origin Resource Sharing)[1]※1といった仕組みが存在する。今回は後者のCORSを使ってクロスドメインリクエストを実現している。

CORSはクロスドメインリクエストされるサーバ側、つまりAPIゲートウェイ側にアクセス制御のルールを記述する。ブラウザはクロスドメインリクエストを行う場合、最初にプリフライトリクエストと呼ばれるリクエストを行い、実際のリクエストを送信しても安全かどうかの確認を行う仕組みとなっている。なお、このプリフライトリクエストはブラウザが一定のルールに基づき自動で送信しているため、クライアントアプリ側で直接送信を行っているわけではない

[1]※1 Wikipedia：Cross-origin resource sharing
http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
CORS(Cross-Origin Resource Sharing)によるクロスドメイン通信の傾向と対策
http://dev.classmethod.jp/cloud/cors-cross-origin-resource-sharing-cross-domain/

4 サーバアプリ(認証サーバ)

4.1 サーバアプリの構成

サーバアプリのモジュールは下記の構成となっている。

```
ServerApp
  app
  ├ assets
  │ └ stylesheets                  各種スタイルシートを格納する(assets:precompileで
                                   コンパイルされる)
  │   └ **.scss
  ├ controllers
  │ ├ application_controller.rb    アプリ全体のベースコントローラだが特別な処理は
                                   追加していない
  │ ├ auth_controller.rb           Twitter/Facebook認証コールバックコントローラ
  │ └ login_controller.rb          ログイン画面のコントローラ
  ├ helpers
  │ └ application_helper.rb        アプリ全体のヘルパーでアラート表示に関するメ
                                   ソッドが実装されている
  └ views
   ├ layouts
   │ └ application.html.haml       アプリ全体のベースレイアウト
   ├ login
   │ └ index.html.haml             ログイン画面のview
   └ shared
     └ _flash_messages.html.haml   アラート表示テンプレート
  config
  ├ puma.rb                        Puma(Webサーバ)の設定を記述したファイル
  ├ routes.rb                      ルーティングの設定を記述したファイル
  ├ settings                       実行環境別設定ファイルを格納
  │ └ **.yml
  ├ settings.local.yml.sample      サーバ別設定ファイル。環境に合わせAPIゲートウェ
                                   イのキー、Twitterキー、Facebookのキー等を記述す
                                   る
  └ settings.yml                   共通設定ファイル
  lib
  └ apigw_request.rb               APIゲートウェイへのリクエストを行う処理の実装
```

4.2 環境構築手順

4.2.1開発環境について

- Ruby (2.2.0)

➢ Bundler

4.2.2インストール手順

Rubyは2.2が既にインストール済であると想定する。

参考:

Rubyインストールですが、以下ご利用になれます。

RVM導入

https://rvm.io/

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3
```

rvm install

```
$ \curl -sSL https://get.rvm.io | bash -s stable
$ echo source ~/.profile >> ~/.bash_profile
```

Ruby 2.2 insatll

```
$ rvm list known
$ rvm install ruby-2.2
$ which ruby
  ~/.rvm/rubies/ruby-2.2.1/bin/ruby
$ which gem
~/.rvm/rubies/ruby-2.2.1/bin/gem
```

1). Bundlerをインストールする。

```
$ gem install bundler --no-document
```

2). 必要なGemをインストールする。

```
 $ cd /path/to/app
 $ bundle install --without development test --path vendor/bundle
```

3). coffeeスクリプトやscssをJavaScript、CSSファイルとして展開するためアセットプリコンパイルを実施する。

```
$ bundle exec rake assets:precompile
```

4). 設定ファイルを準備する。APIゲートウェイから払い出されたConsumerKey／ConsumerSecretとクライアントアプリ側のURLを設定する。また、5.2.3で作成する暗号化キーの設定も行う。

```
$ cp config/settings.local.yml.sample config/settings.local.yml
$ vim config/settings.local.yml

---
apigw_keys_default:
  consumer_key: 'APIGWのConsumer Key'
  secret_key: 'APIGWのConsumer Secret'
```

```
access_token_encryption_key: '5.2.3で作成されたBase64 keyの値'

webapp:
  url: 'http://xxx.xxx.xxx.xxx/apigw' #最後のスラッシュは不要
```

5). 環境変数を設定する。.env.sampleを参考にForemanを利用してもよい。
※Foremanはバックグラウンドで動くプロセスの管理や環境変数の管理をしてくれるツール

```
$ export SECRET_KEY_BASE=(bundle exec rake secret で生成された文字列を設定する)

# 以下は必要に応じて設定
$ export RAILS_SERVE_STATIC_FILES=true (Railsにアセットをserveさせる。nginx等と連携させる場合は不要)
$ export RAILS_RELATIVE_URL_ROOT=/apigw (実行時サブディレクトリを指定。通常は指定する必要はない)
```

4.2.3 ソーシャルログインの準備

TwitterとFacebookのアプリを各サイトから作成し、各URLの設定を以下のように設定する。

Twitter Apps登録:
https://apps.twitter.com/
【TwitterコールバックURL】

```
http(s)://<本認証サーバーのURL(ホスト名)>/auth/twitter/callback
```

Facebook Apps登録:
https://developers.facebook.com/
【Facebook SiteURL】

```
http(s)://<本認証サーバーのURL(ホスト名)>
```

※適切に設定しないとFacebookのログイン画面へ遷移できないので注意

Twitter、Facebookのアプリ管理画面より取得した認証情報を設定ファイルに記述する。

```
$ vim config/settings.local.yml

---
# Twitterアプリ認証情報
twitter:
  consumer_key: 'Consumer Key (API Key)'
  consumer_secret: 'Consumer Secret (API Secret)'
# Facebookアプリ認証情報
facebook:
  app_id: 'App ID'
```

```
  app_secret: 'App Secret'
```

4.2.4起動・停止手順(Puma単体での起動)

注意: 単純に、商用APIゲートウェイに接続する場合、
本項目をスキップし、4.2.5に進む

ローカル環境などで動作確認する場合は、以下のコマンドを実行利用する。サーバ起動後、http:// <IPアドレス>:3000でアクセスし、サーバを停止する場合は、起動したコンソールからCtrl-cで停止する。

```
$ bundle exec rails server -e development -b 0.0.0.0 -p 3000
```

4.2.5nginxとの連携

ここではnginxと連携させる場合の設定例を記載する。なお、Pumaの設定はconfig/puma.rbを参照。

nginx
http://nginx.org/en/
Puma
http://puma.io/

Pumaをデーモンモードで実行

```
$ bundle exec puma -e production -C config/puma.rb
```

デーモン化したPumaを停止する場合、以下

```
$ kill `cat /path/to/app/tmp/puma.pid`
```

nginxの設定例

アプリのtmp内にあるpuma.socketを参照し、アセット用にpublicへのaliasを設定する必要がある。また、nginxの実行ユーザーが参照できる場所にアプリを配置しないとpublic配下を公開できないので注意が必要。

注意: path/ServerAppは、モジュール配置する環境に応じて変更すること。

```
# 前略
upstream apigw {
  server unix:/path/ServerApp/app/tmp/puma.socket;
}
server {
# 中略
  location ^~ / {
    allow 127.0.0.1;
    allow all;
    alias /path/ServerApp/app/public;
    try_files $uri @apigw;
  }
  location /assets/ {
    alias /path/ServerApp/public/assets/;
```

```
  }
  location @apigw {
    if (-f $request_filename) { break; }
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_pass http://apigw;
  }
 # 後略
}
```

4.3 アクセストークンについて

本デモアプリでは、簡易的ではあるがセキュリティ対策としてNTTCom OAuthAPIにより取得したアクセストークンを暗号化してからクライアントに送出している。暗号化の手順は以下の通りである。

1). Twitter or Facebookでの認証成功後、NTTCom OAuthAPIよりアクセストークンを取得する。
2). IVを生成し、取得したアクセストークンを暗号化キー(※1)とIVを使いAES256で暗号化する。
3). IVと暗号化済アクセストークンをBase64でそれぞれエンコードし、コロン(":")で連結する。
4). 3)で作成した暗号化文字列とOAuthAPIより取得した有効期限をクライアントへ送出する。

※1暗号化キーは各環境ごとに作成する。詳細は4.2.2を参照のこと。

5 クライアントアプリ

5.1 クライアントアプリの構成

クライアントアプリのモジュールは下記の構成となっている。

```
ClientApp
app
├ index.html
├ scripts
│ ├ app.coffee                          アプリの全体設定やルーティングを記述
│ ├ controllers
│ │ ├ auth.coffee                       アクセストークンを受け取るコントローラ
│ │ ├ contracts
│ │ │ ├ contract_detail.coffee          契約詳細を表示するコントローラ
│ │ │ ├ contract_group.coffee           契約ID一覧を表示するコントローラ
│ │ │ └ contract_list.coffee            契約一覧を表示するコントローラ
│ │ ├ root.coffee                       他の全コントローラの親のコントローラ
│ │ ├ service_orders
│ │ │ ├ service_order_detail.coffee     サービスオーダ進捗詳細を表示するコントローラ
│ │ │ └ service_order_list.coffee       サービスオーダ進捗一覧を表示するコントローラ
│ │ └ service_orders
│ │   ├ ticket_detail.coffee            トラブルチケット詳細を表示するコントローラ
│ │   └ ticket_list.coffee              トラブルチケット一覧を表示するコントローラ
│ ├ directives
│ │ ├ ncs_loading.coffee                一覧のローディング表示や0件表示などをするテンプレート
│ │ ├ ncs_root.coffee                   index.html直下のテンプレート
│ │ └ ncs_type.coffee                   故障状況やカテゴリを日本語で表示するテンプレート
│ ├ filters
│ │ ├ forward_match.coffee              前方一致で絞り込みを行うフィルター
│ │ ├ ncs_date.coffee                   日付を表示するフィルター
│ │ ├ pagination.coffee                 ページネーションを行うフィルター
│ │ ├ service.coffee                    UNOの絞り込みを行うフィルター
│ │ ├ service_sort_order.coffee         故障状況によってソートするフィルター
│ │ ├ status.coffee                     故障状況によって絞り込むフィルター
│ │ ├ truncate.coffee                   文字を省略して「...」を表示するフィルター
│ │ └ vpn_group.coffee                  V番号による契約一覧の絞り込みを行うフィルター
```

│└ services	
│ ├ api	
│ │├ access_token_decrypter.coffee	暗号化されたアクセストークンを復号する実装
│ │├ access_token_decryption_key.coffee	アクセストークンの復号キーを定義
│ │├ api_client.coffee	APIへのリクエスト処理の実装
│ │└ api_settings.coffee	APIの設定を定義
│ ├ menu	
│ │├ menu_constant.coffee	左側メニューの項目を定義
│ │├ menu_factory.coffee	menuConstantから左側メニューのモデルオブジェクトを生成する
│ │└ menu_list.coffee	menuFactoryを順番に並べた配列を生成する
│ ├ per_page.coffee	ページネーションの1ページの件数を定義
│ ├ services	
│ │├ service_category_constant.coffee	サービスメニューのグループ(UNO、Cloud･･･)を定義
│ │├ service_category_factory.coffee	serviceCategoryFactoryから右側メニューのカテゴリのモデルオブジェクトを生成する
│ │├ service_constant.coffee	サービスメニュー(UNO、Cloudn、UCaaS等）を定義
│ │└ service_factory.coffee	serviceConstantからサービスメニューのモデルオブジェクトを生成する
│ ├ statuses	
│ │├ contract_region_constant.coffee	UNOの国内(N)・国際(W)を定義
│ │├ order_status_constant.coffee	サービスオーダの進捗状況を定義
│ │├ order_type_constant.coffee	サービスオーダのオーダタイプを定義
│ │├ service_status_constant.coffee	契約一覧の故障状況を定義
│ │└ trouble_status_constant.coffee	トラブルチケットの対応状況を定義
│ └ stores	
│ ├ contract_store.coffee	契約一覧の取得と保持を行うサービス
│ ├ service_order_store.coffee	サービスオーダ進捗一覧の取得と保持を行うサービス
│ └ ticket_store.coffee	トラブルチケット一覧の取得と保持を行うサービス
├ styles	
│├ _navmenu.scss	左右メニューに関するスタイルを定義
│└ main.scss	左右メニュー以外の独自スタイルを定義
└ views	コントローラとフィルターに結びつくViewを格納Hamlで記述すると、ビルド時にhtmlに変換される

5.2 環境構築手順

5.2.1開発環境について

- Node.js (0.12.x)
  - ➢ CoffeeScript
  - ➢ Yo
  - ➢ Bower
  - ➢ Grunt
- Ruby (2.1.x)
  - ➢ Compass
  - ➢ Haml

5.2.2インストール手順

Rubyは2.1以降が既にインストール済とする。

参考:

Rubyインストールですが、以下ご利用になれます。

RVM導入

https://rvm.io/

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3
```

rvm install

```
$ \curl -sSL https://get.rvm.io | bash -s stable
$ echo source ~/.profile >> ~/.bash_profile
```

Ruby 2.2 insatll

```
$ rvm list known
$ rvm install ruby-2.2
$ which ruby
  ~/.rvm/rubies/ruby-2.2.1/bin/ruby
$ which gem
~/.rvm/rubies/ruby-2.2.1/bin/gem
```

1). Node.jsをインストールする。

```
$ brew install nodebrew
$ nodebrew install-binary stable

※Homebrewで直接 brew install node でも問題なし
```

2). 必要なnpmパッケージをインストールする。

```
$ npm install -g coffee-script yo bower grunt-cli
```

3). 必要なGemをインストールする。

```
$ gem install compass haml
```

4). クライアントアプリのソースコードを展開し、クライアントアプリのルートディレクトリに移動後、プロジェクト依存パッケージをインストールする。

```
$ git clone https://github.com/nttcom/apigw-sample-clientapp.git
$ cd /path/to/ClientApp
$ npm install
$ bower install
```

bower install時に以下のようなメッセージが表示された場合3を選択する
注意: by nttcom-api-gateway-webappを選択してください。

```
Unable to find a suitable version for angular, please choose one:
    1) angular#>=1 <1.3.0 which resolved to 1.2.28 and is required by angular-bootstrap#0.12.1
    2) angular#1.3.14 which resolved to 1.3.14 and is required by angular-animate#1.3.14,
angular-aria#1.3.14, angular-cookies#1.3.14,
angular-messages#1.3.14, angular-mocks#1.3.14, angular-resource#1.3.14,
angular-route#1.3.14, angular-sanitize#1.3.14, angular-touch#1.3.14
    3) angular#^1.3.0 which resolved to 1.3.14 and is required by nttcom-api-gateway-webapp
    4) angular#>=1.0.8 which resolved to 1.3.14 and is required by ngstorage#0.3.0
    5) angular#>=1.0.0 <1.4.0 which resolved to 1.3.14 and is required by
angular-spinner#0.6.1Prefix the choice with ! to persist it to bower.json
```

5.2.3デバイストークン暗号化／復号キーの作成

アクセストークンの暗号化・復号化キーは利用環境ごとに生成したものを設定する。

```
$ node generate_encryption_key.js

bytes key:
[1542051799,1241119239,-1613628043,1525684332,-1864292387,1226057669,1
631506051,1281970661]
Base64 key: W+nX10n5+gef0f11WvAYbJDhJ91JFCfFYT7Og0xpUeU=
```

bytes key:の方を、app/scripts/services/api/access_token_decryption_key.coffee に設定する。

```
angular.module 'nttcomApiGatewayWebappApp'
  .constant 'accessTokenDecryptionKey',
[1542051799,1241119239,-1613628043,1525684332,-1864292387,1226057669,1
631506051,1281970661]
```

Base64 key:はサーバアプリ側に設定する。

5.2.4デプロイ手順

1). ビルドする。

```
$ grunt build
```

※ビルドでエラーもしくはワーニングが発生し、ビルドが止まってしまう場合は、npm install か bower install を実行しライブラリを最新版へアップデートする。

2). ビルド完了後、distディレクトリ内の内容をWebサーバにアップロードすればデプロイは完了。
注: 4.2.2 2)で設定したwebappパスに本distディレクトリ内容をアップすること。

http://xxx.xxx.xxx.xxx/apigw
の場合、distをアップし、apigwにリネームする

5.3 アクセストークンについて

本デモアプリでは、認証成功時にサーバから送られてくるアクセストークンは暗号化されているため、Business ProcessAPIへアクセスする際には暗号化されたアクセストークンを復号化する必要がある。復号化の手順は以下の通りである。

1). 認証成功後、サーバから暗号化文字列(IV+暗号化されたアクセストークン)と有効期限が送られてくるので、それをローカルストレージへ保存する。
2). Business ProcessAPIへアクセスする際にローカルストレージより有効期限を取得し、アクセストークンの有効期限が切れていないかの確認を行う。もし、有効期限が切れていた場合はローカルストレージのデータを削除し、ログイン画面を表示する。
3). 有効期限が切れていない場合は、暗号化文字列をコロン(":")区切りで分割し、それぞれをBase64でデコードする。
4). 分割した暗号化文字列の前者がIV、後者が暗号化されたアクセストークンとなるので復号化キー(※1)を使いアクセストークンを復号化する。

※1復号化キーは各環境ごとに作成する。詳細は5.2.3を参照のこと。

6 デモアプリの使い方

6.1 基本的な使い方

デモアプリは一覧に表示された情報の中から閲覧したい情報を選び、詳細を確認するシンプルな構成となっている。ここでは各画面の簡単な操作方法について説明する。

1). メニュー

メニューは普段は隠れているため、左右のメニューボタンをタップして表示させる。プロセスの変更は左メニューから行い、図6-1のようなメニューが表示されるので照会したいプロセスをタップして選択する。また、照会するサービスを変更する場合は右メニューから行い、サービスメニューの一覧が表示されるのでタップして選択する。

2). ページャー

図6-1メニュー表示

各一覧画面の1ページ毎の表示件数は10件(デフォルト)となっている。データ件数がこの件数を超える場合はページャーを利用する。

図6-2ページャー

6.2 ログイン

1). TwitterまたはFacebookによる認証でログインを行う。サインインボタンを押すとそれぞれの認証画面へ遷移ので、認証情報を入力し認証が成功すると再度アプリ側に戻ってくる。

図6-3ログイン

6.3 契約管理

1). 契約管理一覧画面

APIより取得した契約情報を表示する。一覧のいずれかの契約情報をタップすると図6-4の契約管理詳細へ遷移する。

①に契約番号を入力すると前方一致で一覧の絞り込みを行うことができる。また、②をタップすると図6-5のような選択肢が表示され、状況(故障状況)で絞り込みを行うことができる。

①

②

図6-4契約管理一覧

2). 契約管理詳細画面

詳細画面では一覧で選んだ契約情報の詳細を表示する。

図6-5故障状況絞り込み

①の「チケット情報」ボタンをタップすると、トラブルチケット管理一覧画面へ遷移する。トラブルチケット管理一覧画面では、先ほど閲覧していた契約番号のトラブルチケットのみが表示される。
②のvpnグループ番号(UNOのみ)をタップすると図10の契約ID一覧画面へ遷移する。

図6-7契約ID一覧

6.4 サービスオーダ進捗管理

図6-6契約管理詳細

1). サービスオーダ進捗管理一覧画面

APIより取得したサービス契約オーダ情報を表示する。一覧のいずれかのオーダ情報をタップすると図6-8のサービスオーダ進捗管理詳細へ遷移する。契約管理同様、契約番号・状況(進捗状況)による絞り込みが可能となっている。

2). サービスオーダ進捗管理詳細画面

詳細画面では一覧で選んだサービス契約オーダ情報の詳細を表示する。

6.5 トラブルチケット管理

1). トラブルチケット管理一覧画面

APIより取得したトラブルチケット情報を表示する。一覧のいずれかのチケット情報をタップすると図6-10のトラブルチケット管理詳細へ遷移する。契約管理同様、契約番号・状況(対応状況)による絞り込みが可能となっている

2). トラブルチケット管理詳細画面

詳細画面では一覧で選んだトラブルチケット情報の詳細を表示する。

7 デモアプリのカスタマイズ(認証サーバ編)

7.1 APIゲートウェイの接続先を変更する

デフォルトのAPIゲートウェイの接続先を変更したい場合は以下のファイルを修正する。

```
$ vim config/settings.yml

---
apigw:
  host: ' APIゲートウェイの接続先URL '
```

また、起動モードごとに接続先を変更する場合は、config/settings/(各起動モード).ymlへ上記内容を記述すればよい。

7.2 ConsumerKey／ConsumerSecretを変更する

APIゲートウェイへのアクセスに利用する認証キーを変更する場合は以下のファイルを修正する。

```
$ vim config/settings.local.yml

---
apigw_keys_default:
  consumer_key: 'APIGWのConsumer Key '
  secret_key: 'APIGWのConsumer Secret '
```

また、今回のデモアプリでは簡易的なユーザ管理の仕組みとして、ソーシャルアカウントごとにAPIゲートウェイの認証キーを設定できるようになっている。設定は先ほど同様に以下のファイルを追記する。

```
$ vim config/settings.local.yml

---
apigw_keys:
  -
    twitter_user_names: ['user1', 'user2']
                        facebook_user_mail_addresses:       ['user1@example.com',
'user2@example.com']
    consumer_key: ' APIGWのConsumer Key① '
    secret_key: ' APIGWのConsumer Secret① '
  -
    twitter_user_names: ['user3', 'user4']
                        facebook_user_mail_addresses:       ['user3@example.com',
'user4@example.com']
    consumer_key: ' APIGWのConsumer Key② '
    secret_key: ' APIGWのConsumer Secret② '
```

　twitter_user_namesにはTwitterのユーザー名(@username)を記述し、facebook_user_mail_addressesにはFacebookのメインメールアドレスを指定する。
なお、apigw_keysの設定がない場合や設定されたアカウント以外でログインした場合は、apigw_keys_defaultに設定した認証キーが使われる。

8 デモアプリのカスタマイズ(クライアント編)

8.1 APIゲートウェイと認証サーバの接続先を変更する

APIゲートウェイと認証サーバの接続先を変更するには以下のファイルを修正する。

```
$ vim app/scripts/services/api/api_settings.coffee

----
angular.module 'nttcomApiGatewayWebappApp'
  .constant 'apiSettings', {
    timeout_ms: 10000
    url:
      contracts: 'https://[APIゲートウェイのホスト名]
/v1/business-process/contracts'
      serviceOrders: 'https://[APIゲートウェイのホスト名]
/v1/business-process/service-orders'
      tickets: 'https://[APIゲートウェイのホスト名]/v1/business-process/tickets'
    auth_url: 'http(s)://[認証サーバのURL]'
  }
```

8.2 ビジネスプロセスメニューのカスタマイズ

8.2.1名称を変更する

プロセスのメニュー名を変更するには以下のファイルを修正する。

```
$ vim app/scripts/services/menu/menu_constant.coffee

----
contracts:
  name: "契約管理"
  disabledServiceKeys: []
```

"name"を変更することでメニュー名を変更することが出来る。なお、"disabledServiceKeys"にはそのプロセスで無効とするサービスが記述されているので、このリストを編集することでプロセスごとのサービスの有効・無効を変更することが可能である。

8.2.2表示順を変更する

プロセスの表示順を変更するには以下のファイルを修正する。

```
$ vim app/scripts/services/menu/menu_list.coffee

----
[ 'menuFactory', (menuFactory) ->
  menuList = []
  menuOrder = ['contracts', 'service_orders', 'tickets']
# 中略
  return menuList
```

```
]
```

menuOrderに格納するリストの順番を変更することで表示順を変更することが出来る。

8.3 サービスメニューのカスタマイズ

8.3.1名称を変更する

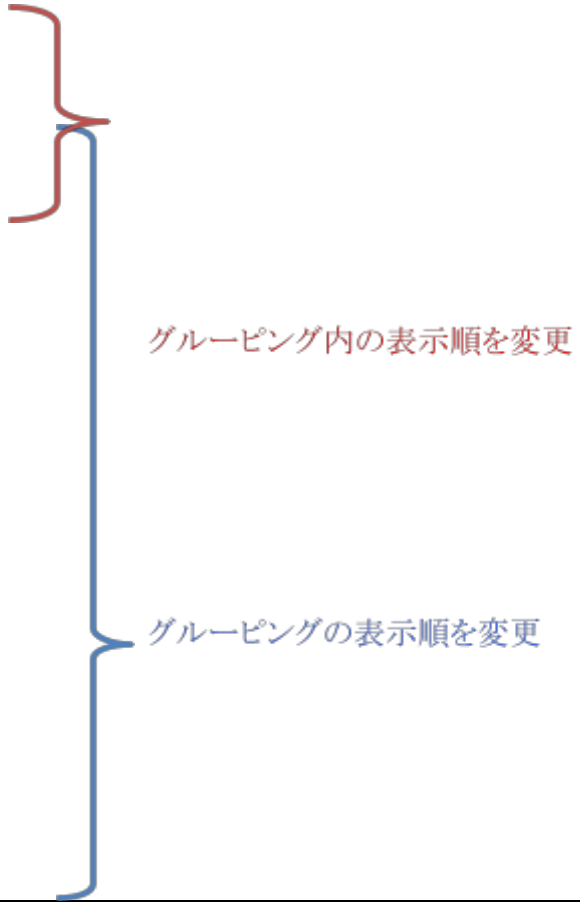
サービスのメニュー名を変更するには以下のファイルを修正する。

```
$ vim app/scripts/services/services/service_constant.coffee

----
uno:
  name: "UNO"
  requestKey: "uno"
  showVpnGroupId: true
# 中略
  troubleStatusKey: "uno"
```

8.3.2表示順を変更する

サービスの表示順を変更するには以下のファイルを修正する。

```
$ vim app/scripts/services/services/service_category_constant.coffee

----
   [{
     name: "UNO"
     serviceKeys: [
       "uno"
       "uno-mobile"
       "global-m2m"
       "uno-virtual"
     ]
   }
   {
     name: "Cloud"
     serviceKeys: [
       "cloudn"
       "bhec"
     ]
   }
```



なお、グルーピング名称を変更するには"name"を変更する。

8.4 1ページ毎の表示件数を変更する

一覧の表示件数(デフォルト10件)を変更するには以下のファイルを修正する。

```
$ vim app/scripts/services/per_page.coffee

----
.constant 'perPage', 10
```

8.5 デザインのカスタマイズ

8.5.1 メニューを変更する

プロセス・サービスメニューのデザインを変更するには以下のファイルを修正する。

```
$ vim app/styles/_navmenu.scss

----
// プロセスメニューの色定義
$navmenu-menu-bg:              #背景色
$navmenu-menu-link-color:      #非選択メニューの文字色
$navmenu-menu-link-active-color: #選択中メニューの文字色
$navmenu-menu-link-active-bg:   #選択中メニューの背景色
$navmenu-menu-brand-color:     #APIゲートウェイアプリタイトル文字色
// 中略
// サービスメニューの色定義
$navmenu-services-bg:              #背景色
$navmenu-services-link-color:      #非選択メニューの文字色
$navmenu-services-link-active-color:   #選択中メニューの文字色
$navmenu-services-link-active-bg:      #選択中メニューの背景色
$navmenu-services-link-disabled-color: #選択不可メニューの文字色
$navmenu-services-brand-color:      #グルーピング名称文字色
```

9 おわりに

今後APIゲートウェイが拡張され、対応サービスの拡充やAPIからオーダーの登録・変更・削除が行えるようになった場合、デモアプリを拡張することで容易に対応することが出来る。ただし、今回のデモアプリはあくまでもサンプルアプリであり、実際の業務で利用する場合は、各社のコンプライアンスやセキュリティポリシーに照らし合わせた上でのカスタマイズが必須である。

10 参考文献

・「OAuth」, <http://oauth.net/>
・「JSONP - Wikipedia」, <http://en.wikipedia.org/wiki/JSONP>
・「CORS(Cross-Origin Resource Sharing)について整理してみた」, <http://dev.classmethod.jp/etc/about-cors/>

問い合わせ先
以下よりお問い合わせください。
https://developer.ntt.com/ja/contact.html