# Windowsからスマホまでマルチデバイスで サクサク外部API呼び出し活用法

第33回 エンバカデロ・デベロッパーキャンプ

エンバカデロ・テクノロジーズ
毛利 春幸

embarcadero
DEVELOPER CAMP

# はじめに

- 現在、多くの企業がより新しい価値のあるサービス構築を目指して、多数の Web API を公開し、第 3 者企業とサービス連携を行っています。
- API 公開によってサービスの応用範囲や収益性が向上する点は、Google、Twitter、Facebookといった企業によって証明されており、すでに多くの企業にとっては API を公開することは当たり前になっています。
- Delphi / C++Builderは RESTを得意とするコンポーネントが揃っています APIをご紹介しながら 各コンポーネントの使い方を、ご説明してまいります。

# アジェンダ

- 各社REST サービスAPIについて
  - Amazon API Gateway
- OAuth 2.0 の仕組みと認証方法
  - TOAuth2Authenticatorコンポーネントを使って簡単OAuth2.0
  - Line, Yahoo, docomo IDなどの認証
- AWS
  - S3
- Kinveyを使ったiOSプッシュ通知
- 駅すぱあと API 接続
  - 位置情報から最寄駅検索

# 各社REST サービスAPIについて

- REST Service API

# REST

- Representational State Transfer (REST) は、ウェブのような分散ハイパーメディアシステムのためのソフトウェアアーキテクチャのスタイルのひとつである。
(インターネットのプロトコルの中心となったHTTP/Web技術がベースです)
- JSON/ XMLなどのフォーマットを用いてデータのやりとりを行います
これからご説明する 「Amazon Web Service」や「LINE」 「Kinvey」「駅すぱあと」はAPIサービスをXML / JSONで提供しています

- その他、今回説明しないRESTサービスとしてはdocomo ID, Yahoo ID, Google, Facebook, spotifyなどが有名です

# 各社REST サービスAPIについて

- OAuth 2.0
  - ID+パスワードで外部サービスを利用する
    Google+, Line, YahoolD, FacebookなどのID認証されたユーザーは
    自社サービスを許可とする、認証代行のようなイメージ
- ストレージサービス
  - AWS S3, Azure ストレージアカウント(BLOBストレージ)などのクラウドストレージ提供
- 独自データ提供
  - 天気予報, 路線情報(駅すぱあと), 番組表, 音楽情報(spotify)

- Amazon API Gateway

# Amazon API Gateway

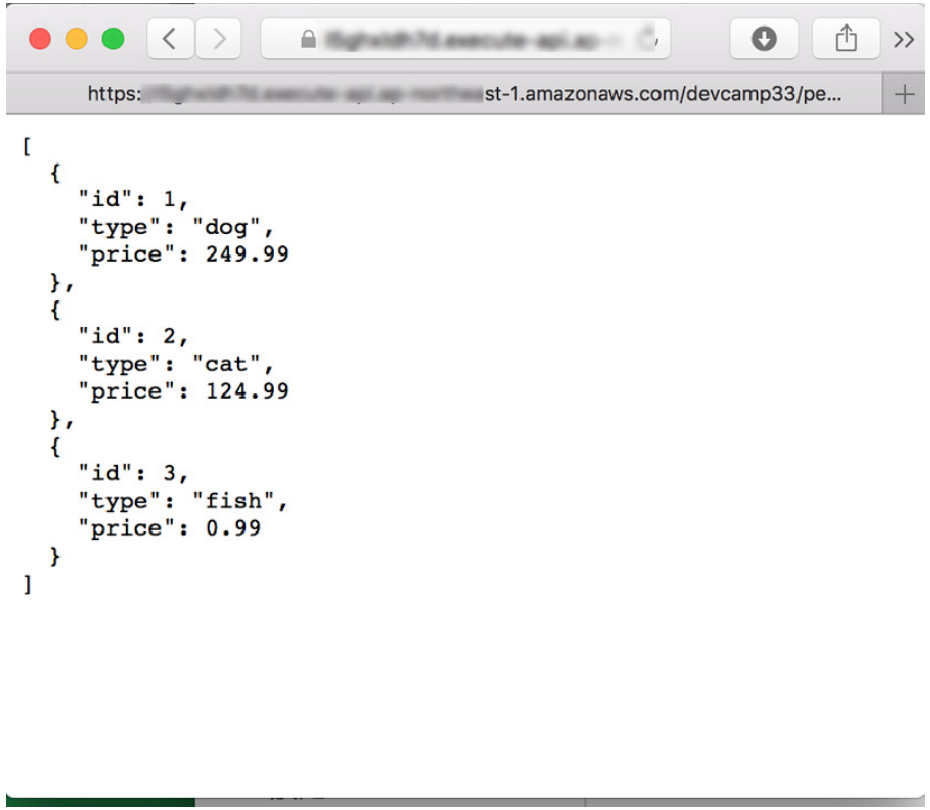
- RESTful なアプリケーションプログラミングインターフェイス (API) の作成、デプロイ、および管理によるバックエンド HTTP エンドポイント、AWS Lambda 関数、またはその他の AWS サービスの公開をサポートし、フロントエンド HTTP エンドポイントを介して、
公開された API メソッドを呼び出す AWS サービス。

# Amazon API Gateway

- API Gateway のサンプルAPIを設定
  - サンプル「PetStore」API作成

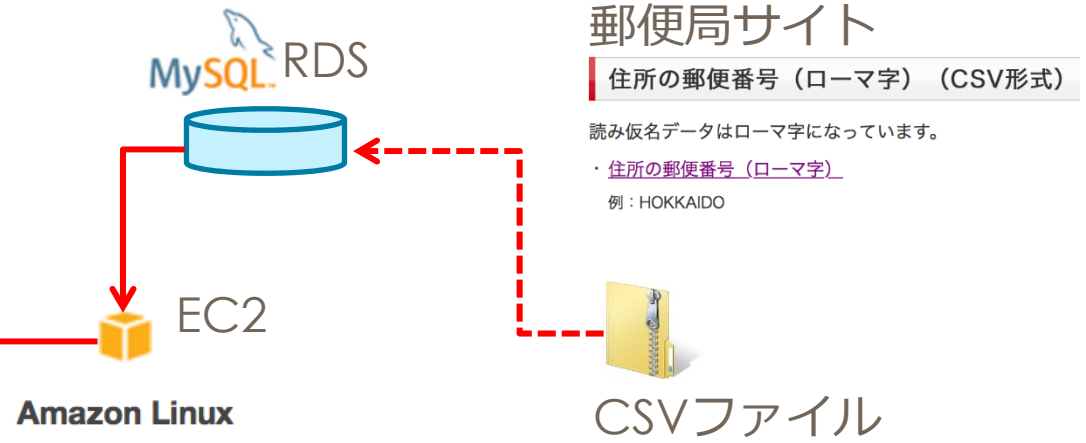
# Amazon API Gateway

- API Gateway 「 PetStore (サンプル)」ブラウザで参照
  - https：//******execute-api.ap-northeast-1.amazonaws.com/devcamp33/pets




# Amazon API Gateway

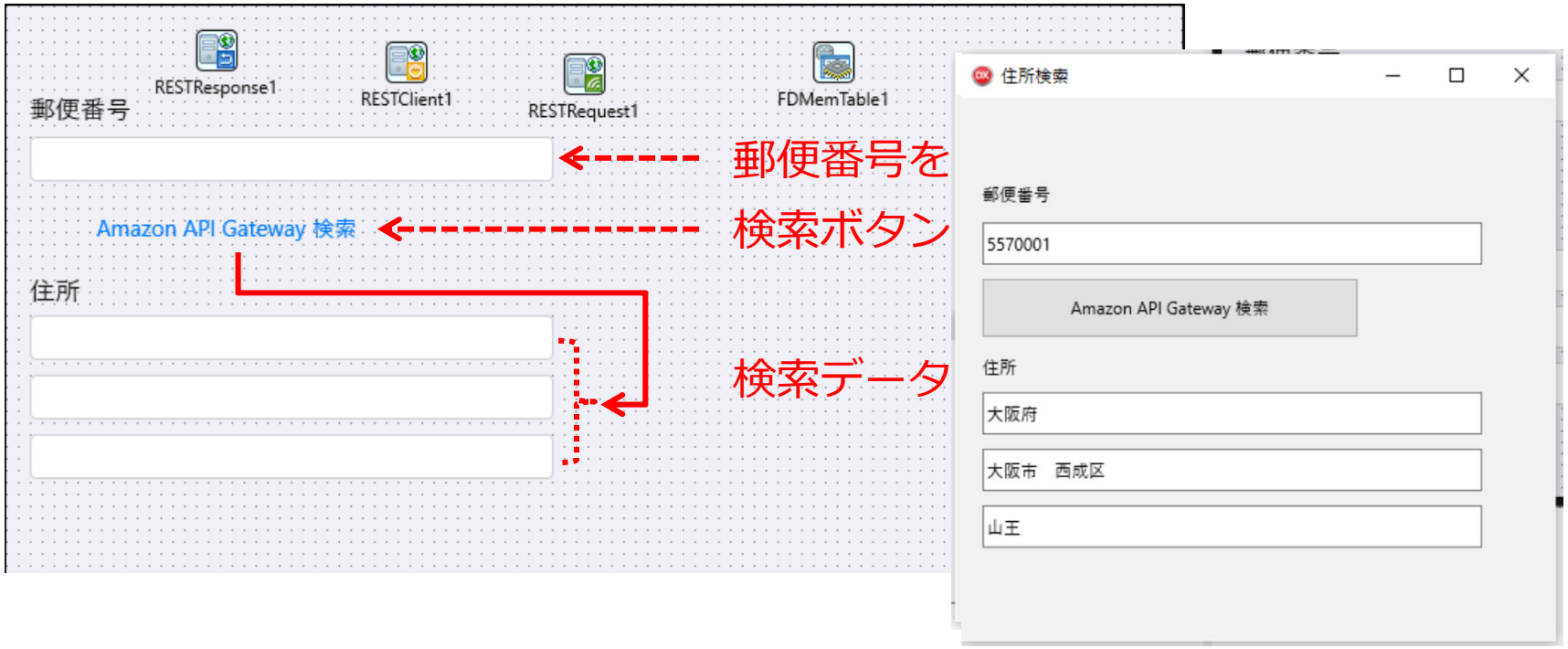
- API Gatewayを少しカスタマイズ


http://www.post.japanpost.jp/zipcode/download.html

# Amazon API Gateway

- API Gateway 「 v1/yubin/ 」Delphi/C++Builderから接続
  - https：//*.execute-api.ap-northeast-1.amazonaws.com/devcamp33/v1/yubin/



# Amazon API Gateway

- API Gateway 「 v1/yubin/ 」 Delphiから接続
  - devcamp33/v1/yubin/ を取得する

```
procedure TForm1.Button1Click(Sender: TObject);                    //検索ボタンイベント
var
  p1: TRESTRequestParameter;
begin
  RESTClient1.BaseURL := 'https://*execute-api.ap-northeast-1.amazonaws.com/devcamp33';
  RESTRequest1.Resource := 'v1/yubin/';                            //リソース
  RESTRequest1.Params.Clear;
  p1  := RESTRequest1.Params.AddItem;
  p1.name   := 'zip_num';                                          //郵便番号パラメータ
  p1.Value  := Edit1.Text;
  RESTRequest1.Execute;                                            //REST実行

  MemTableSet(RESTResponse1.JSONValue as TJSONArray);              //TJSONValueが取得できる
end;
```

# Amazon API Gateway

- API Gateway 「 v1/yubin/ 」 C++Builderから接続
  - devcamp33/v1/yubin/ を取得する

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        RESTClient1->BaseURL      = “https : //* .execute-api.ap-northeast-1.amazonaws.com/devcamp33";
        RESTRequest1->Resource    = "v1/yubin/";
        RESTRequest1->Params->Clear();
        auto p1           = RESTRequest1->Params->AddItem();
        p1->name          = "zip_num";
        p1->Value         = Edit1->Text;
        RESTRequest1->Execute();

}       MemTableSet(static_cast<TJSONArray*>( RESTResponse1->JSONValue ) );
```

# Amazon API Gateway

- API Gateway 「 v1/yubin/ 」 Delphiから接続
  - devcamp33/v1/yubin/ で**JSON**取得後 **TFDMemTable**に登録

```
const field_names: array[0..3] of string = ('zip_num','pref','city','address1');

procedure TForm1.MemTableSet(const ajson: TJSONArray);
var
  i: Integer; field_name: String; a_item: TListViewItem; a_line: String;
begin
  FDMemTable1.Open;                                          //FDMemTableの初期設定は済前提
  for i := 0 to Pred(ajson.Count) do                         //TJSONArrayデータをコピー
  begin
    FDMemTable1.Append;
    FDMemTable1.FieldByName('zip_uniq').AsInteger :=
      StrToInt(ajson.Items[i].GetValue<TJSONString>('zip_uniq').Value);
    for field_name in field_names do                         //フィールド配列順でコピー
    begin
      FDMemTable1.FieldByName(field_name).AsString   :=
        ajson.Items[i].GetValue<TJSONString>(field_name).Value;
    end;
  end;
  FDMemTable1.Post;
end;
```

# Amazon API Gateway

- API Gateway 「 v1/yubin/ 」 C++Builderから接続
  - devcamp33/v1/yubin/ で**JSON**取得後 **TFDMemTable**に登録

```
std::vector<UnicodeString> field_names{"zip_num","pref","city","address1"};
void __fastcall TForm1::MemTableSet(TJSONArray* ajson)
{
        TJSONObject*    a_buf{nullptr};
        FDMemTable1->Open();                                    //FDMemTableの初期設定は済前提
        for (int i = 0; i < ajson->Count; i++)                  //TJSONArrayデータをコピー
        {
                a_buf     = static_cast<TJSONObject*>(ajson->Get(i));

                FDMemTable1->Append();
                for (auto field_name : field_names )             //フィールド配列順でコピー
                {
                        FDMemTable1->FieldByName(field_name)->AsString    =
                                static_cast<TJSONString*>(a_buf->GetValue(field_name) )->Value();
                }
        }
        FDMemTable1->Post();
}
```

# OAuth 2.0 の仕組みと認証方法

認可フレームワーク

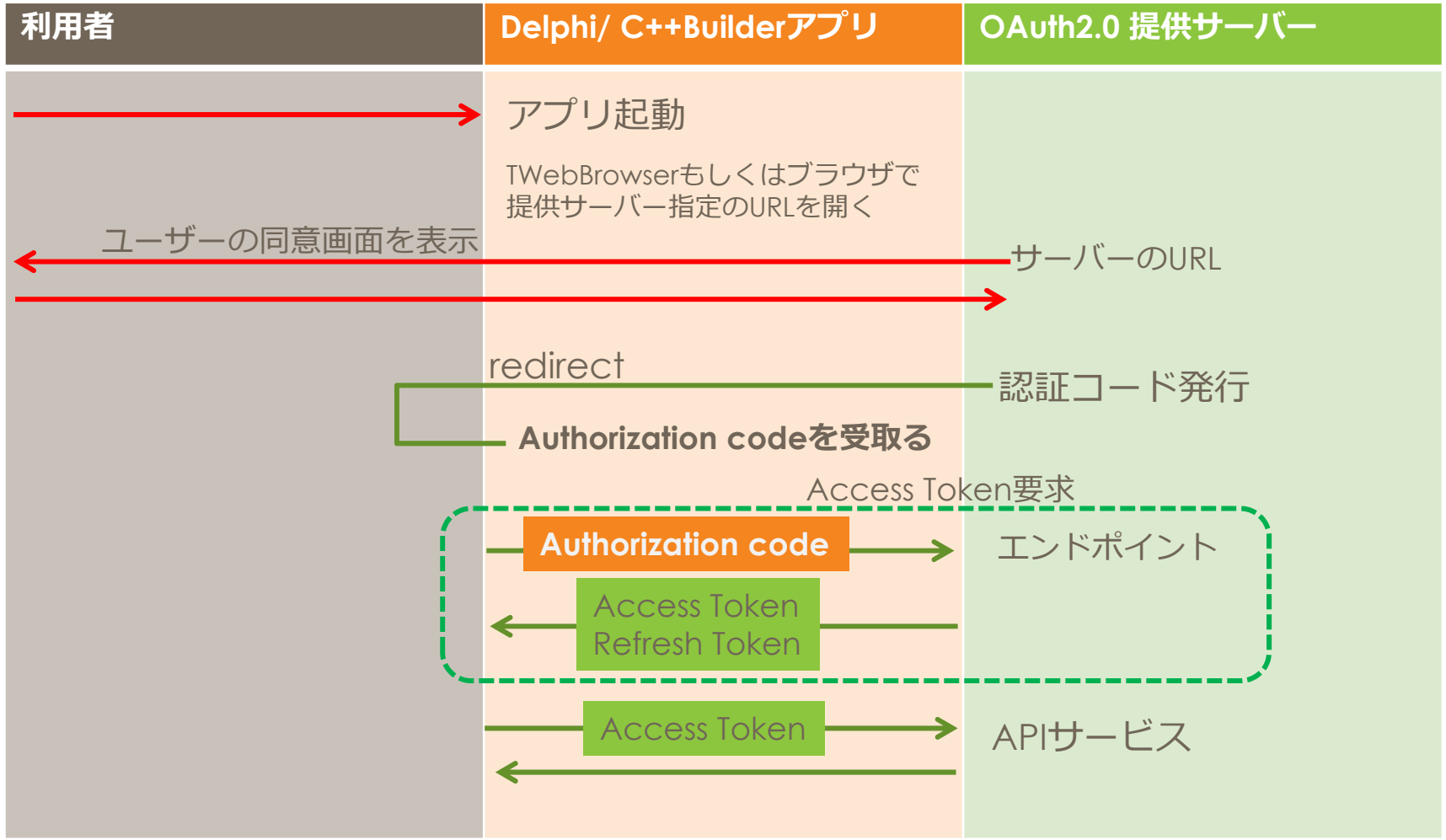
# The OAuth 2.0 Authorization Framework

- OAuth 2.0 は, サードパーティーアプリケーションによるHTTPサービスへの限定的なアクセスを可能にする認可フレームワークである。
サードパーティーアプリケーションによるアクセス権の取得には, リソースオーナーとHTTPサービスの間で同意のためのインタラクションを伴う場合もあるが, サードパーティーアプリケーション自身が自らの権限においてアクセスを許可する場合もある。

- OAuth 2.0はRFC 5849に記載されているOAuth 1.0 プロトコルを廃止し, その代替となるものである.

(https://openid-foundation-japan.github.io/rfc6749.ja.html)

# OAuth 2.0 の仕組み



# OAuth 2.0 「LINE Login」

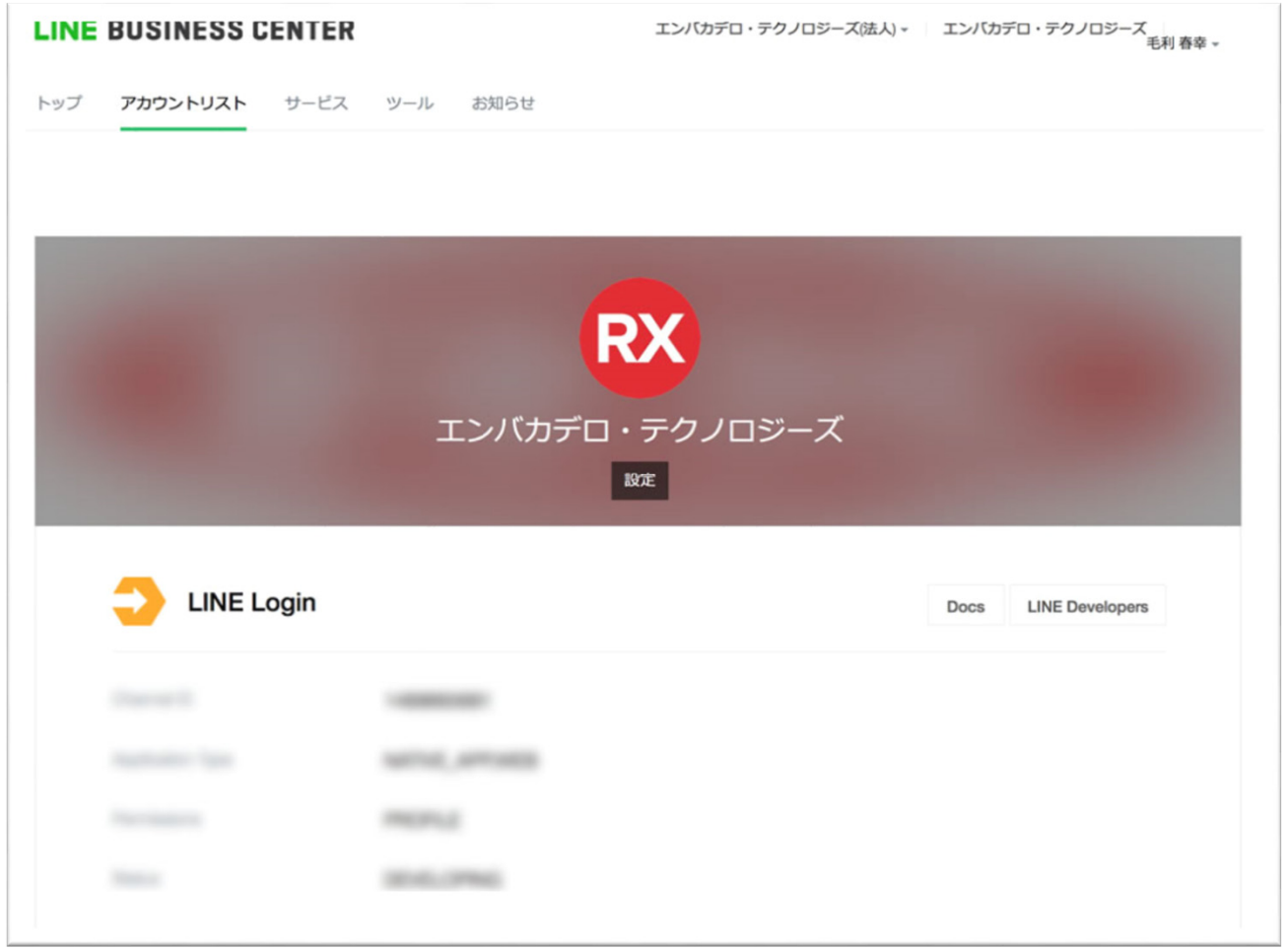
- LINE Login機能をサービスに組み込むことにより、
ユーザーがLINEで簡単にサービスにログインすることが可能となります。

- ログインしたユーザーから「表示名」、「プロフィール画像」と「ひとこと」を取得することができるため、ユーザーに登録していただく作業が必要なくなります。

LINE Login

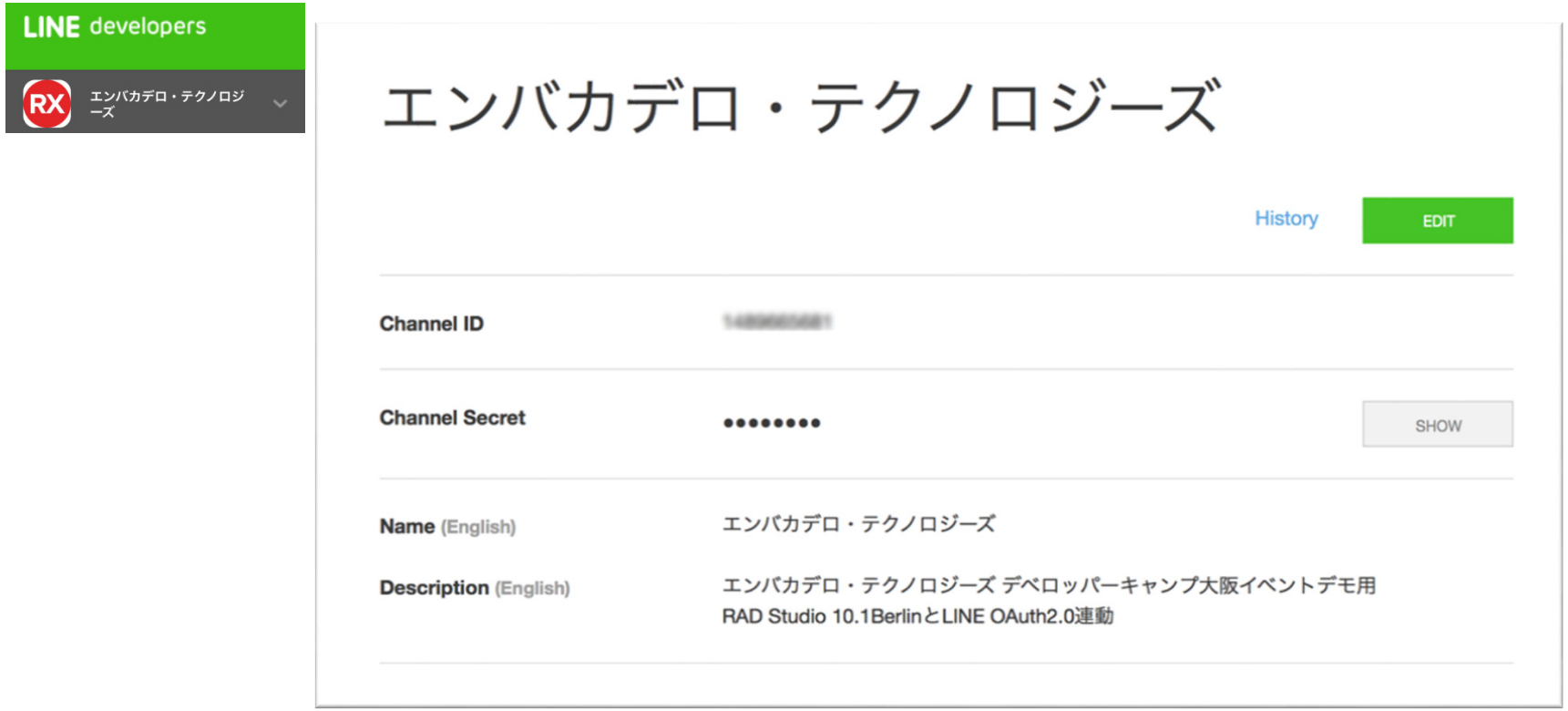
# OAuth 2.0 「LINE Login」

- LINE Login
  企業登録が必要です



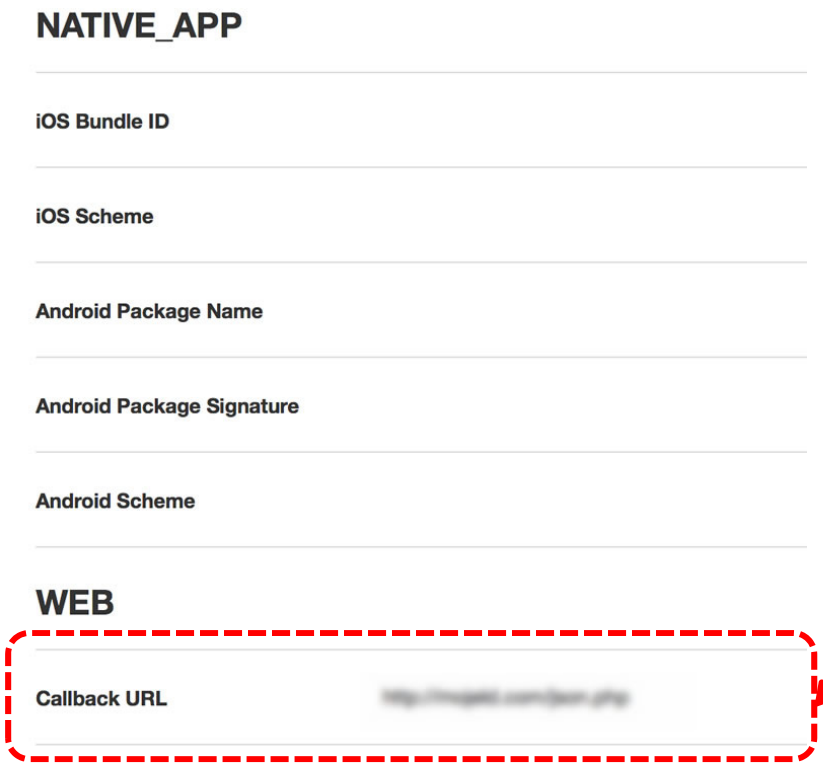

# OAuth 2.0 「LINE Login」

- LINE Login企業登録するとChannel IDとChannel Secretが取得できます




# OAuth 2.0 「LINE Login」

- 認証画面からのコールバックURLの指定が必須

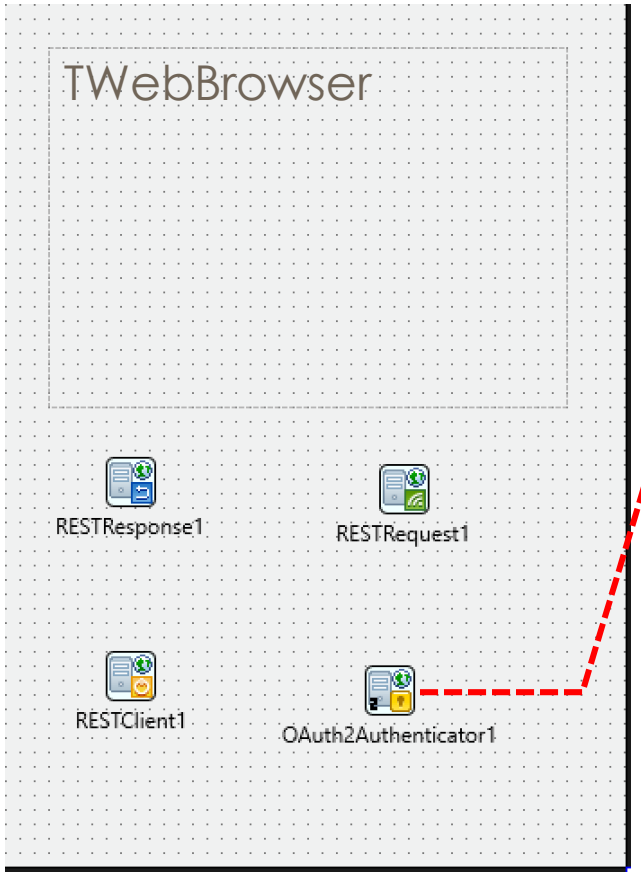


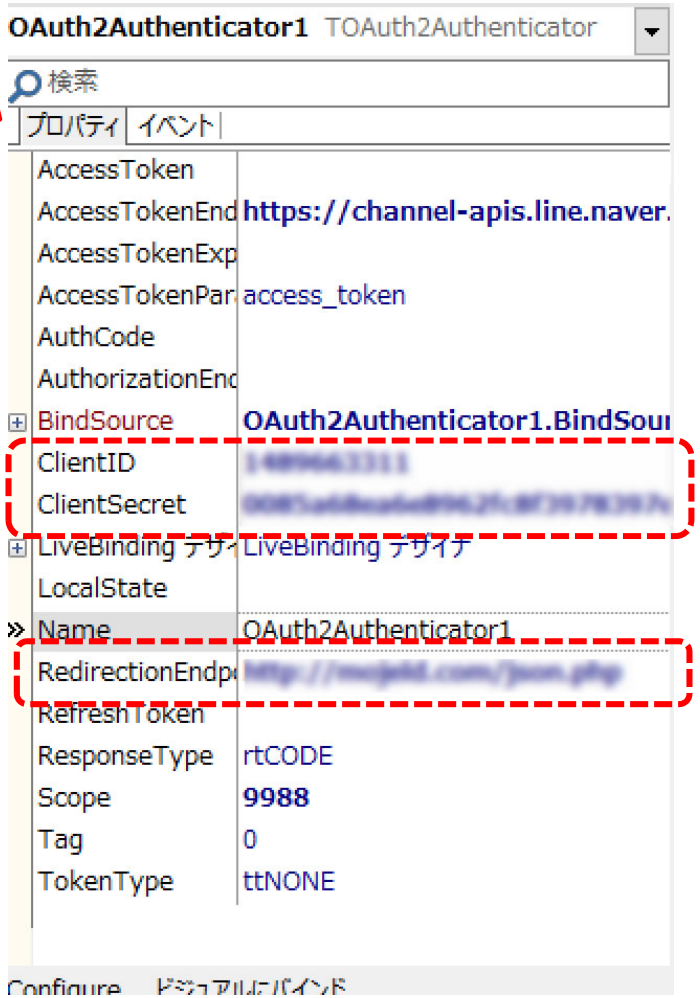

アプリ側で Schemeが設定されている場合は
iOS/Android Schemeなどの設定で良いかと思います

今回は普通のURLコールバックにしました

# OAuth 2.0 「LINE Login」

- Delphi / C++Builder から接続



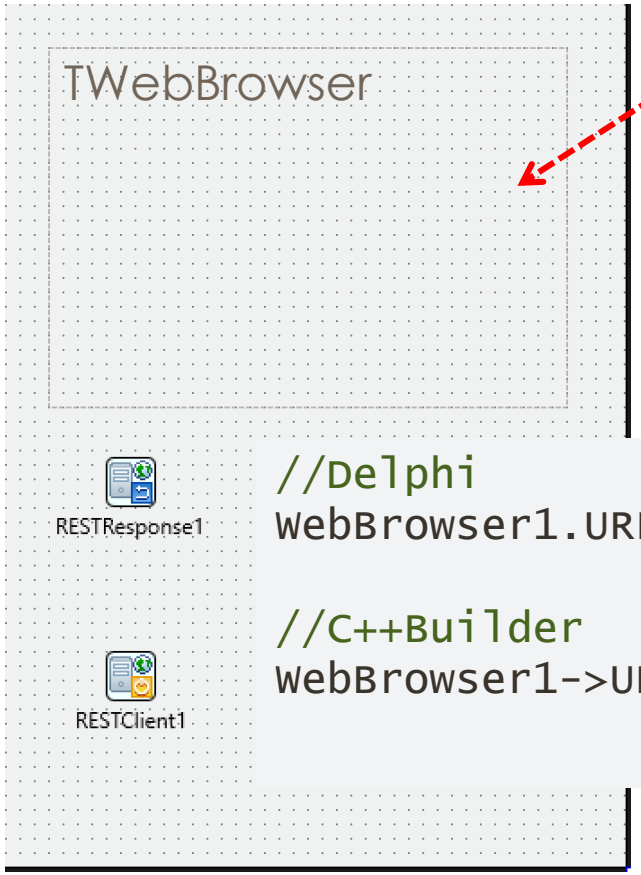




**Channel ID**
**Channel Secret**

**Redirect URL**

# OAuth 2.0 「LINE Login」

- Delphi / C++Builder から接続




LINE developer'sの仕様通りログイン画面を起動する

**Login Screen**

When a user presses the LINE Login button on your website, the user is redirected to the login screen of the LINE Platform for authentication and authorization. The URL is as follows.

```
https://access.line.me/dialog/oauth/weblogin?response_type=code&client_id={Channel ID}&redirect_uri={Callback URL}&state={State}
```

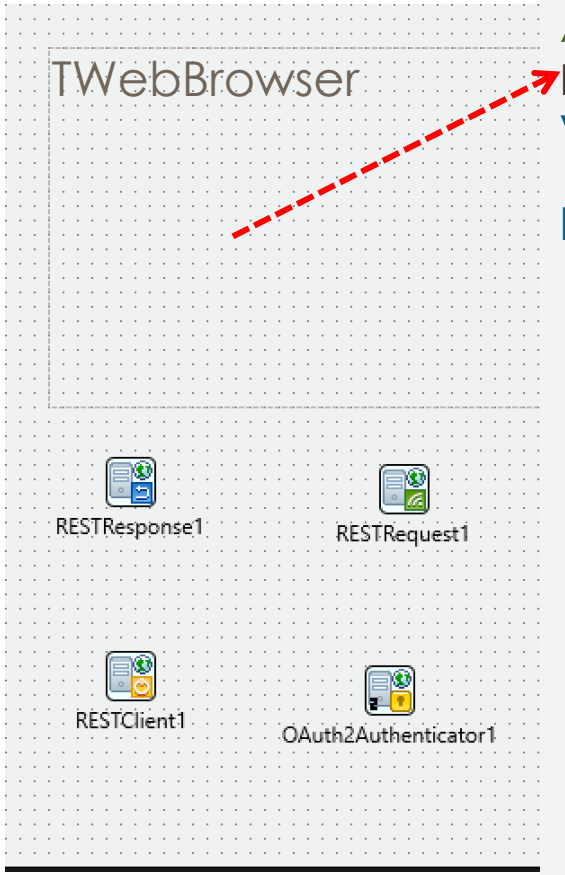
For query parameters, the following must be specified.

```
//Delphi
WebBrowser1.URL := 'https://access.line.me/dialog/oauth/weblogin…..

//C++Builder
WebBrowser1->URL = “https://access.line.me/dialog/oauth/weblogin…..
```

# OAuth 2.0 「LINE Login」

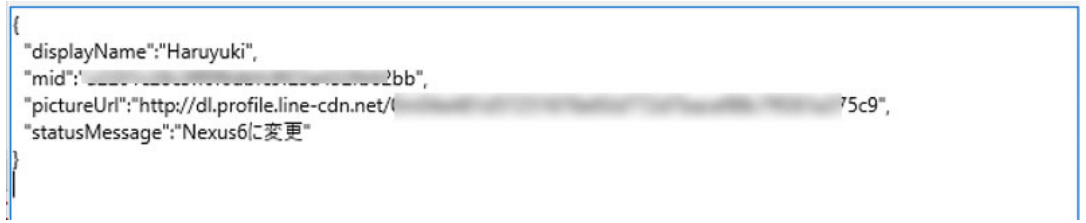
- Delphi / C++Builder から接続

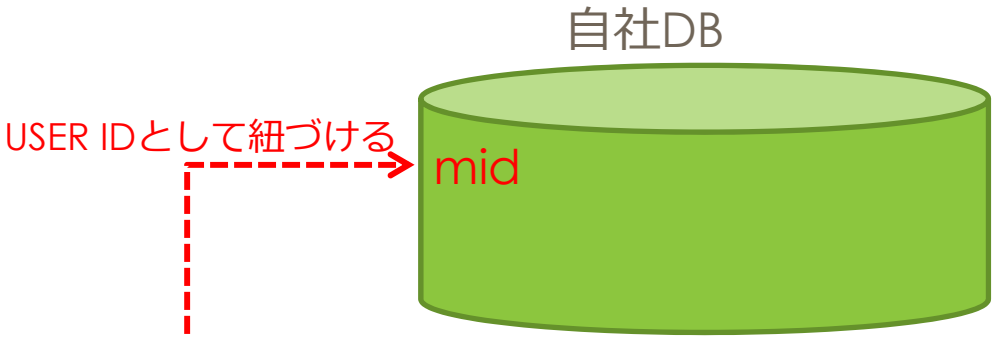



```
//Delphi
procedure TForm1.WebBrowser1DidFinishLoad(ASender: TObject);
var
  stTemp: String;
begin
  stTemp  := WebBrowser1.URL;
  if Pos('code=', stTemp) > 0 then
  begin
    stTemp  := Copy(stTemp, Pos('code=', stTemp)+5, Length(stTemp));
    stTemp  := Copy(stTemp, 1, Pos('state=', stTemp)-2);
    OAuth2Authenticator1.AuthCode := stTemp;
    OAuth2Authenticator1.ChangeAuthCodeToAccesToken;
    if Length(OAuth2Authenticator1.AccessToken) > 0 then
    begin
      RESTRequest1.Method := TRESTRequestMethod.rmGET;
      RESTClient1.BaseURL := 'https://api.line.me/v1/profile';
      RESTRequest1.Execute;   //AccessTokenが取れたのでprofile参照可能
    end;end;end;
```

# OAuth 2.0 「LINE Login」

- LINE Loginから取得したJSON




```
//JSON
{
  "displayName":"Haruyuki",
  “mid”:“xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx”,    //←ここがユニーク
  "pictureUrl":"http://dl.profile.line-cdn.net/xxxxxxxxxxxxxx",
  "statusMessage":"Nexus6に変更"
}
```

ID パスワード管理などはLINEに任せてしまう事が可能になります

# Kinveyを使ったiOSプッシュ通知

- Kinveyとはアメリカの企業であるKinvey社が提供しているBaaS(Backend as a Service)です。

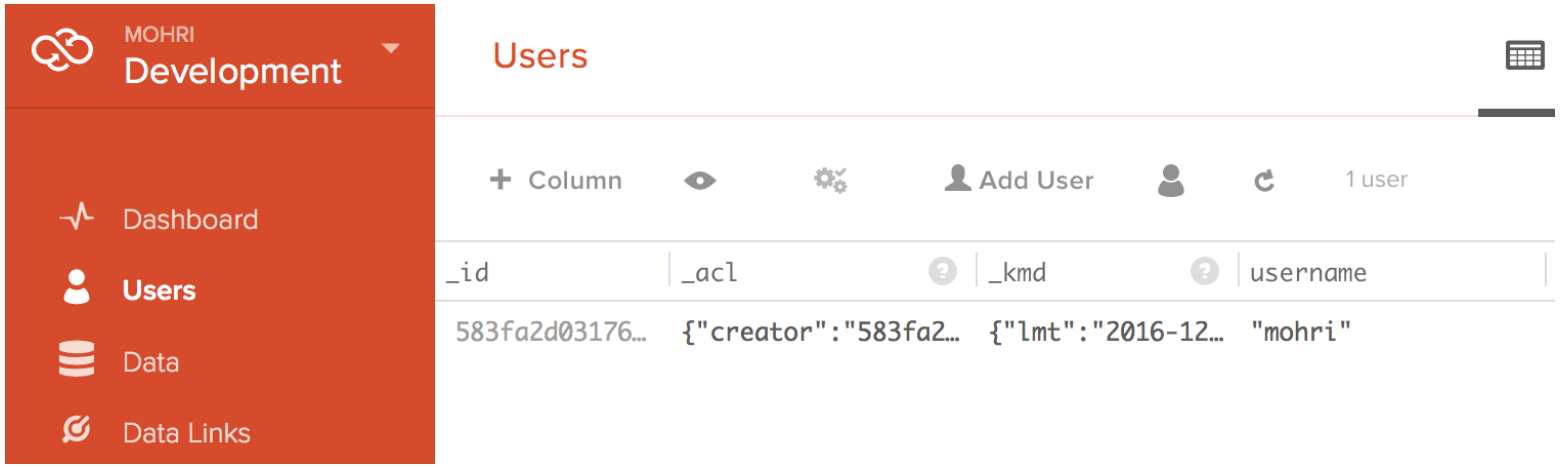
# Kinvey

- Kinveyとはアメリカの企業であるKinvey社が提供しているBaaS(Backend as a Service)です。
  - Users ユーザー管理
  - ストレージ
  - Engagement プッシュ通知
- メリット
  - サーバーを社内で構築しメンテナンスが必要ない

プッシュ通知デモの前に少しだけKinvey機能紹介致します

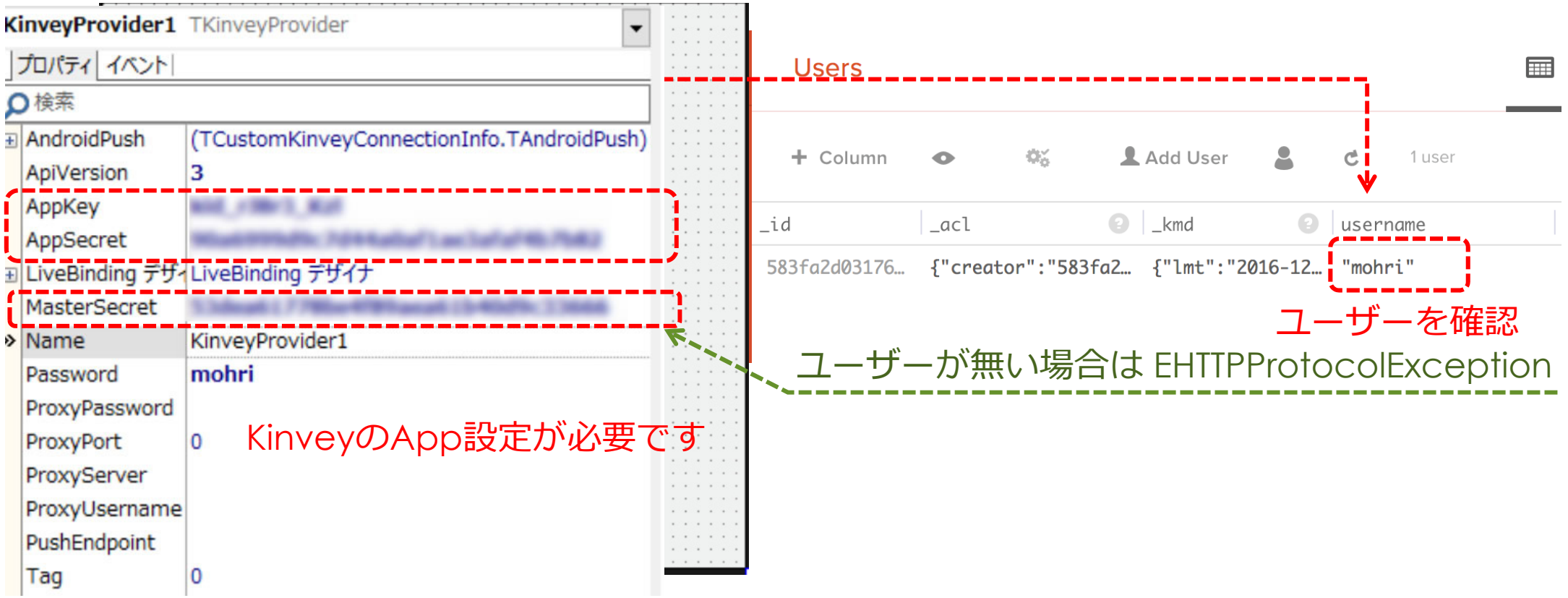
# Kinvey： Users ユーザー管理

- Users ユーザー管理
  - ユーザー追加、削除などブラウザから行う事ができます



# Kinvey： Delphi/C++Builderからのユーザーログイン

- TKinveyProvider + TBackendAuth を使ってアプリのユーザー認証を行います



# Kinvey： Delphiからのユーザーログオウト

- TKinveyProvider + TBackendAuth を使ってアプリのユーザー認証を行います

```
//Delphi
procedure TForm1.Button2Click(Sender: TObject);
begin
  try
    BackendAuth1.Logout;       //ログアウト
    Memo1.Lines.Append('BackendAuth1.Logout; OK' );
  except
    on e: exception do
      Memo1.Lines.Append('BackendAuth1.Logout; '+ e.Message );
  end;
end;
```

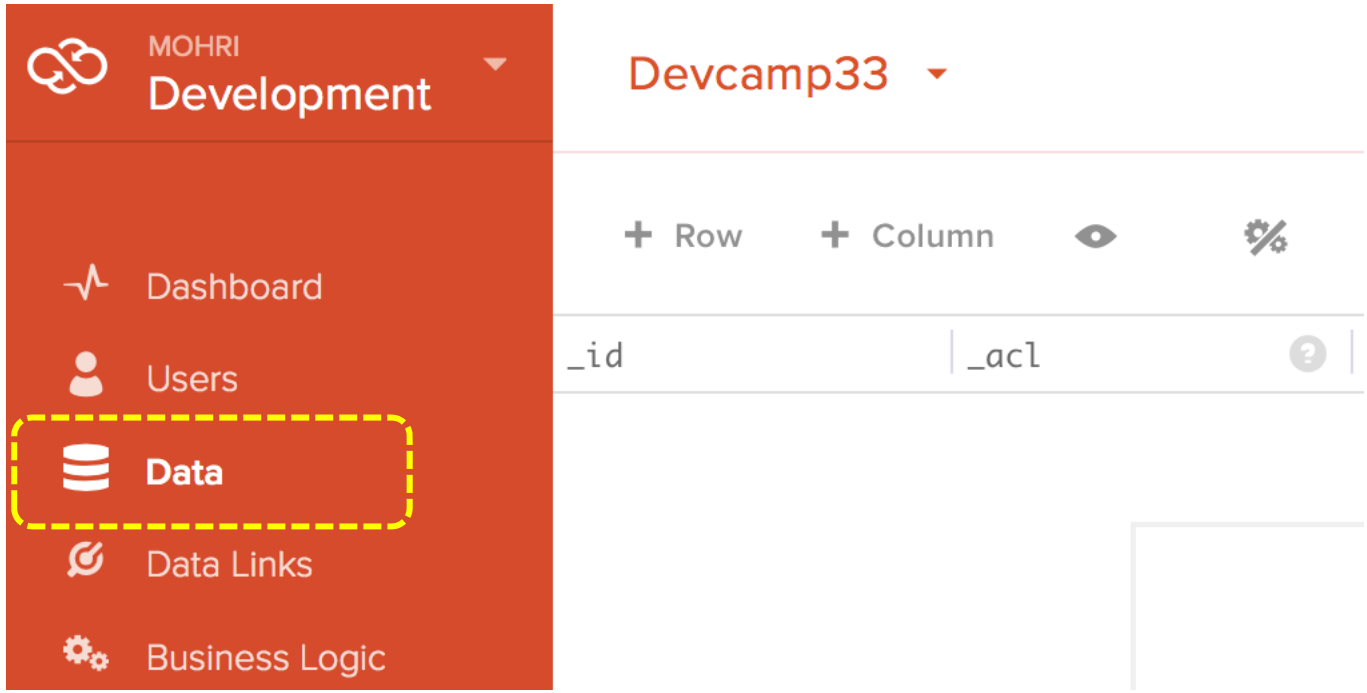
# Kinvey： C++Builderからのユーザーログオウト

- TKinveyProvider + TBackendAuth を使ってアプリのユーザー認証を行います

```
void __fastcall TForm1::Button2Click(TObject *Sender){
        UnicodeString s;
        try
        {
                BackendAuth1->Logout();                 //ログアウト
                s = "BackendAuth1->Logout(); OK";
        }
        catch(EHTTPProtocolException* e1)
        {
                s = "BackendAuth1->Logout(); " + e1->Message;
        }
        Memo1->Lines->Append(s);
}
```

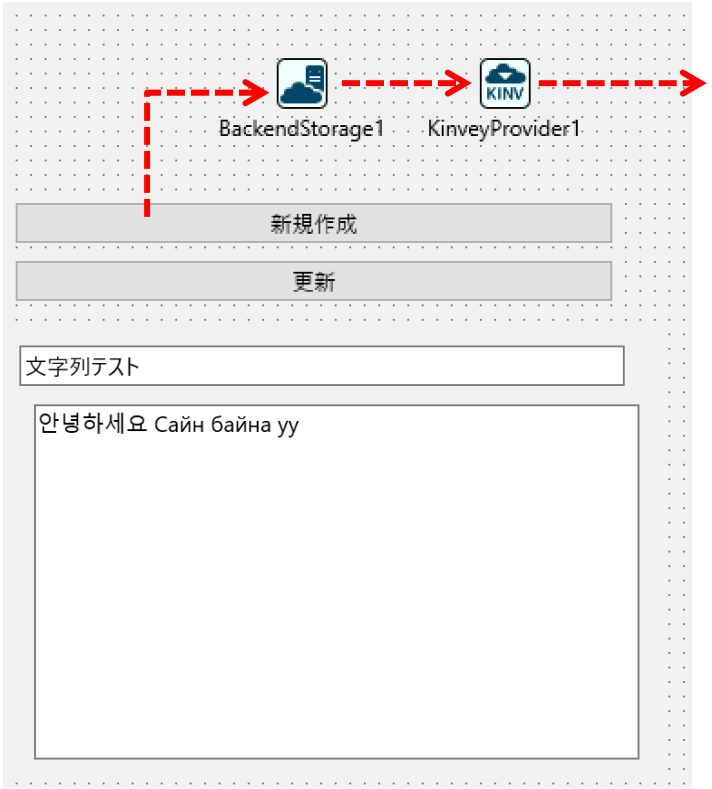
# Kinvey：ストレージ

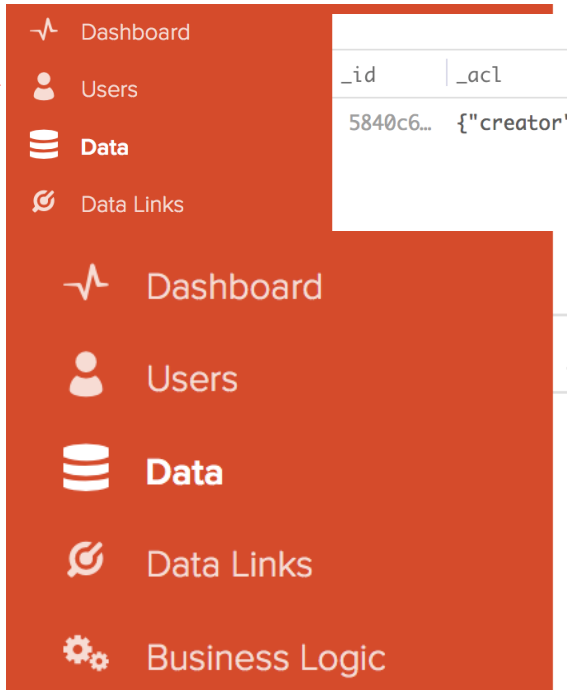
- データ保管場所を提供しています

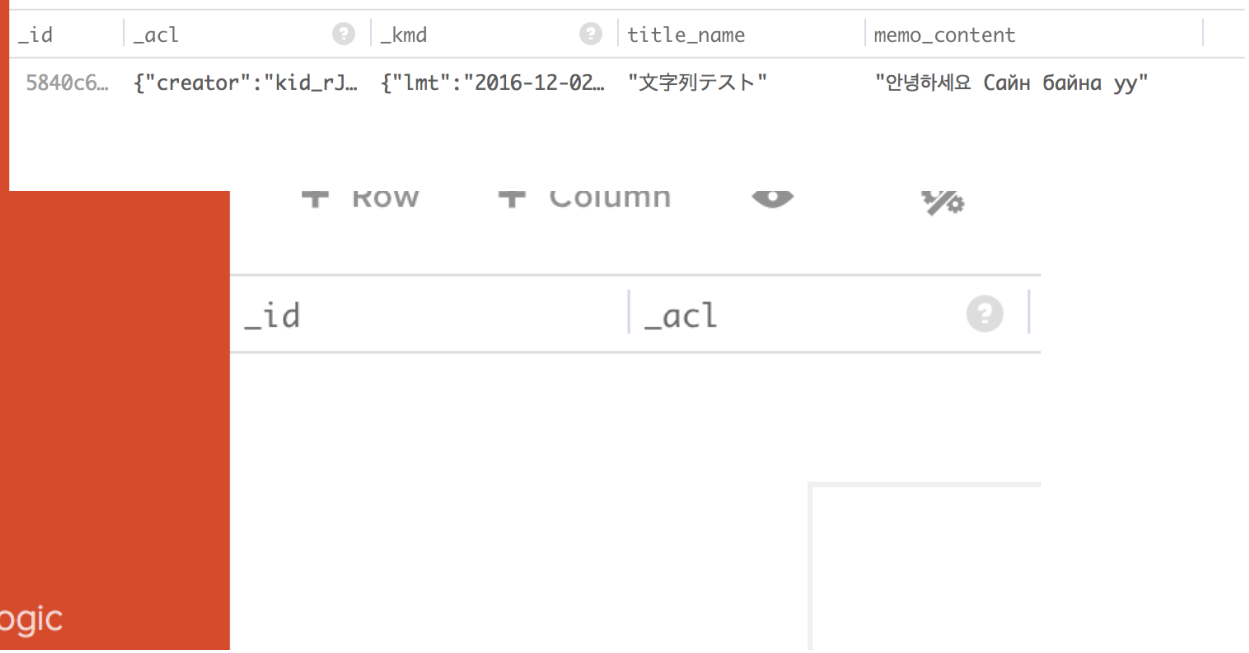


# Kinvey：ストレージ

- TBackendStorageを使って 保存や取り出しが可能です



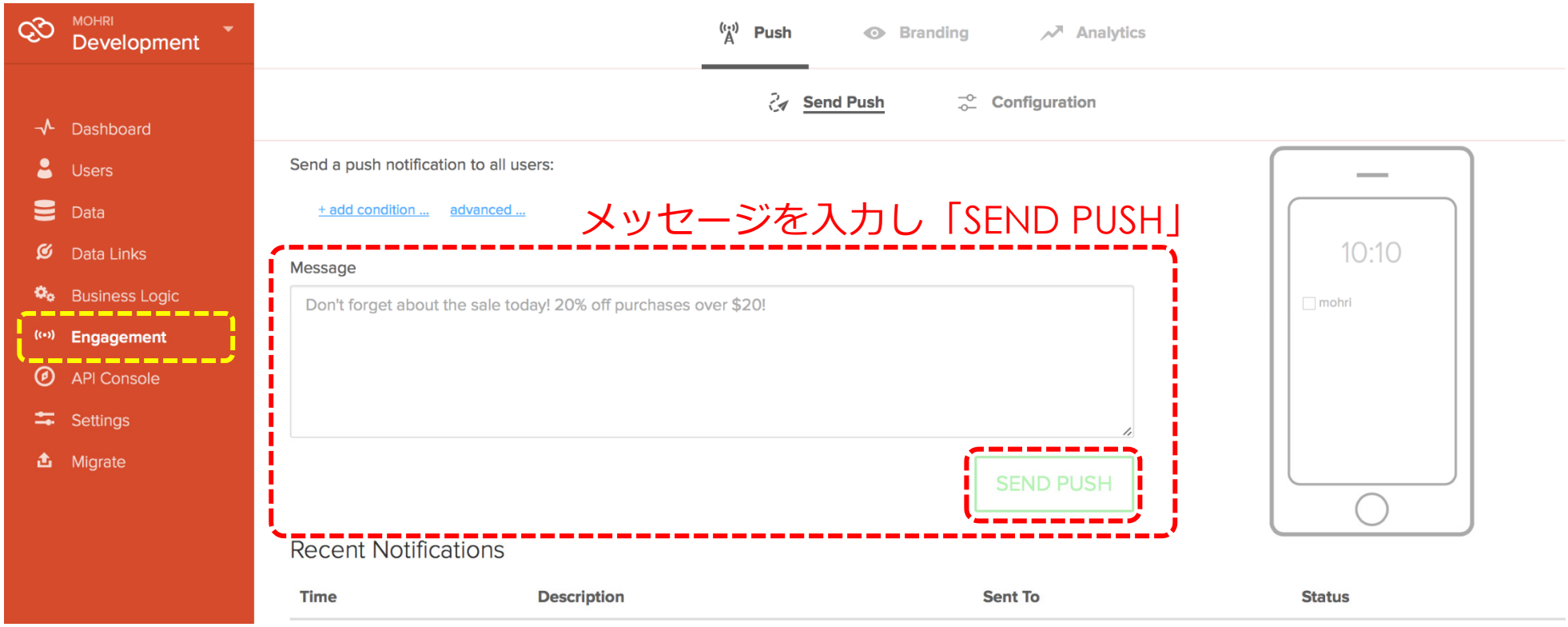
# Kinvey：ストレージ

- Delphi + TBackendStorageを使ったクラス保存コード

```
//Delphi
FBackendList: TBackendObjectList<TKinvey_memo>;
var
  obj_:  TKinvey_memo;
  entry_: TBackendEntityValue;
begin
  obj_  := TKinvey_memo.Create;
  obj_.title_name   := Edit1.Text;
  obj_.memo_content := Memo1.Lines.Text;
  BackendStorage1.Storage.CreateObject<TKinvey_memo>('Devcamp33', obj_, entry_);
  FBackendList.Add(obj_, entry_);
end;
```

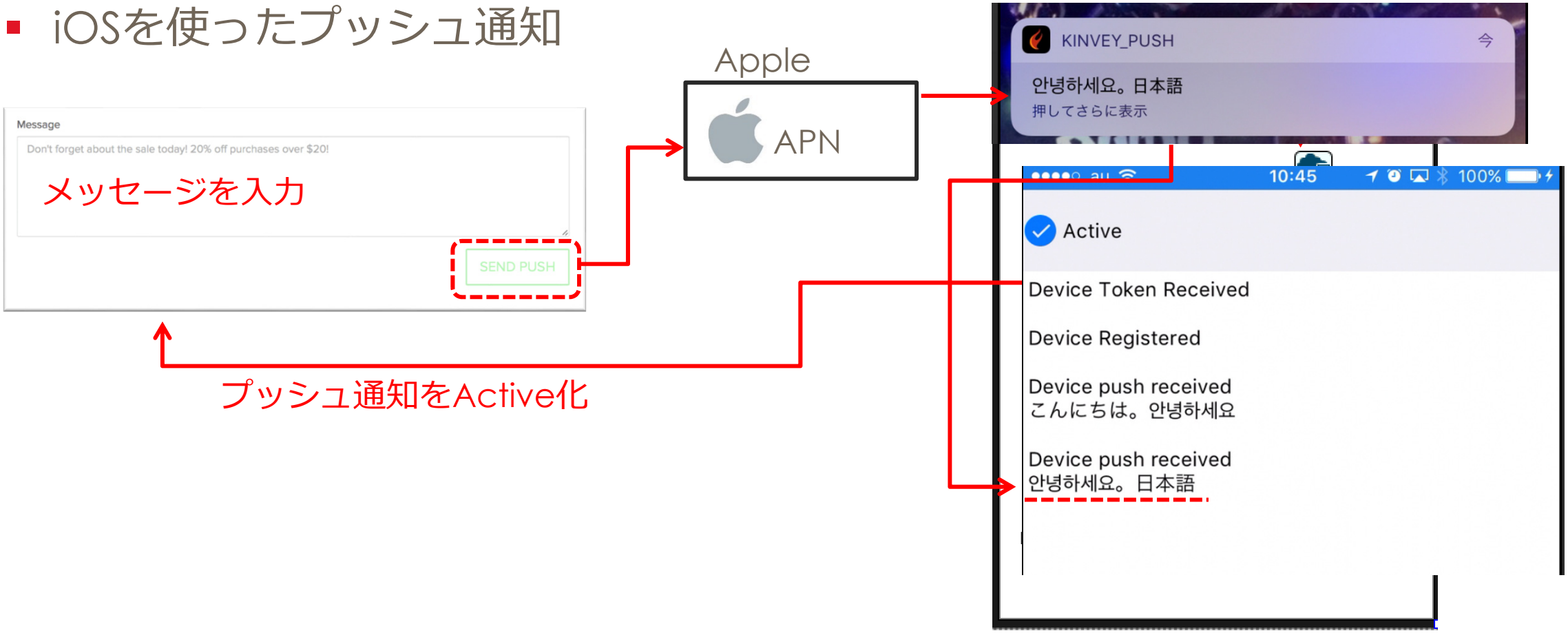
# Kinvey：プッシュ通知

- プッシュ通知送信方法




# Kinvey：プッシュ通知

- iOSを使ったプッシュ通知

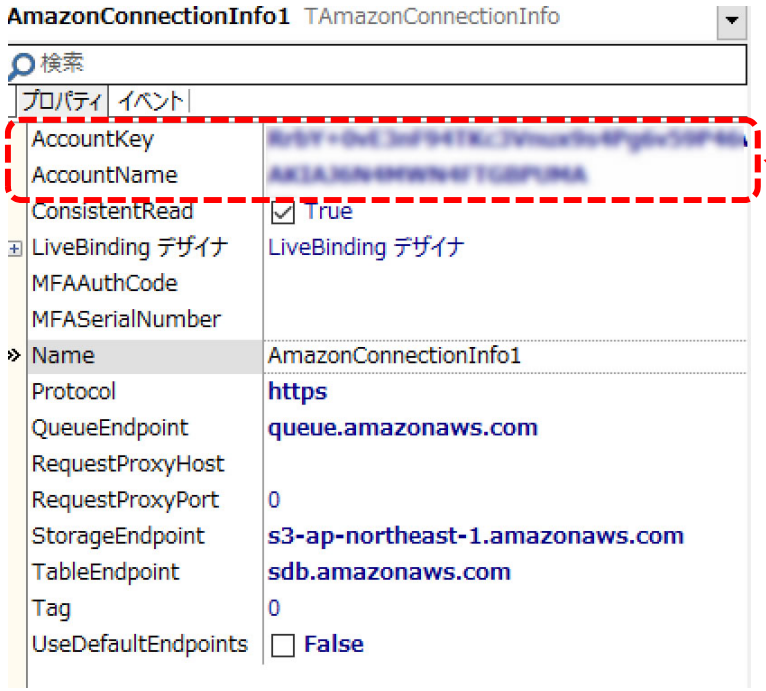


# AWS S3接続

- Amazon Simple Storage Service はインターネット用のストレージサービスです。
- ウェブスケールのコンピューティングを開発者が簡単に利用できるよう設計されています。

# AWS S3接続

- Delphi/C++ Builderは AWS接続のための TAmazonConnectionInfoコンポーネントが用意されています


IAM(Identity and Access Management)


# AWS S3接続

- Delphi/C++Builderを使ってS3接続

```
//少し省略しています
std::unique_ptr<TStringList> list_{std::make_unique<TStringList>(new TStringList()) };
std::unique_ptr<TAmazonStorageService> s3{new TAmazonStorageService(AmazonConnectionInfo1)};
TCloudResponseInfo* res_{new TCloudResponseInfo()};
try{
        TAmazonBucketResult* bs = s3->GetBucket("embarcadero-jp", list_.get(),
                                                res_,TAmazonRegion::amzrAPNortheast1);
        std::unique_ptr<TMemoryStream> mm{new TMemoryStream()};
        mm->LoadFromFile("01.jpg");
        DynamicArray<Byte> arr1;
        arr1.Length                = mm->Size;
        mm->ReadBuffer(arr1, mm->Size);
        mm->Position               = 0;
        //ファイルをアップロードする
        s3->UploadObject("embarcadero-jp", "01.jpg", arr1, true, nullptr, nullptr,
                                                TAmazonACLType::amzbaPublicRead, res_);
        mm->Clear();
        //ファイルを取得する
        s3->GetObject( "embarcadero-jp", "01.jpg", mm.get(), res_);
```

# 駅すぱあとWebサービス API 接続

- 鉄道情報を商用利用可能なAPI

# 駅すぱあとWebサービス フリープラン

- 信頼の鉄道情報を商用利用可能なAPIを0円で提供
  - https：//ekiworld.net/


アクセスキー、ドキュメント、サンプルコード
などが送られてきます

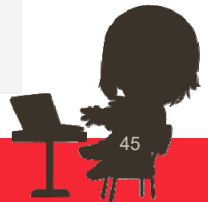
# 駅すぱあとWebサービス フリープラン + C++Builder

- 駅すぱあと Webサービス フリープランAPI

| 項目 | エンドポイント |
| --- | --- |
| 駅付加情報 | /v1/*/station/info |
| 会社情報 | /v1/*/corporation |
| 経路探索 | /v1/*/search/course/light |
| 路線情報 | /v1/*/rail |
| 線区情報 | /v1/*/line |
| 駅情報 | /v1/*/station |
| 駅簡易情報 | /v1/*/station/light |

# 駅すぱあとWebサービス フリープラン + C++Builder

```
enum TEki_apis_ {info, corporation, search_course_light, rail, line, station, station_light};
class TEki_Api : TObject
{
        (省略)
        std::map<TEki_apis_, String> FEndPoint{
                {TEki_apis_::info,                  "/station/info"},
                {TEki_apis_::corporation,           "/corporation"},
                {TEki_apis_::search_course_light,   "/search/course/light"},
                {TEki_apis_::rail,                  "/rail"},
                {TEki_apis_::line,                  "/line"},
                {TEki_apis_::station,               "/station"},
                {TEki_apis_::station_light,         "/station/light"}};
        (省略)
        __property UnicodeString EkiKey          //アクセスキー
        __property TEki_apis_ ApiChannel         //エンドポイント選択
        __property UnicodeString EndPoint        //エンドポイント名(ReadOnry)
        __property UnicodeString EkiName         //駅名
        __property TNotifyEvent  OnHttpJson      //イベント
        __property UnicodeString EkiTo           //駅 TO
        __property UnicodeString EkiFrom         //駅 FROM
        __property int EkiCode                   //駅 コード
}
```

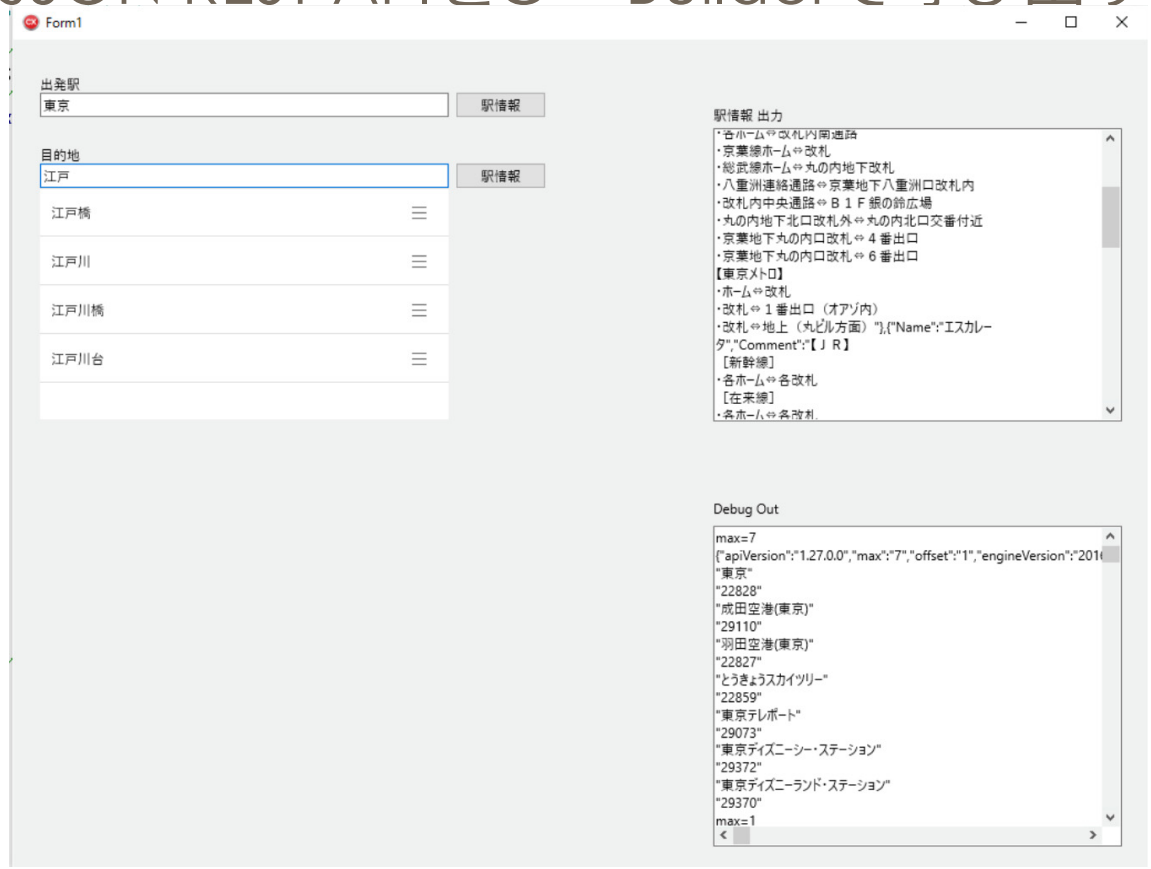


# 駅すぱあとWebサービス フリープラン + C++Builder


```
void __fastcall TForm1::OnEditChange(TObject *Sender)
{
        ListView1->Items->Clear();
        TEdit* edit_     = static_cast<TEdit*>(Sender);
        FselEdit                    = edit_;
        ListView1->Tag              = edit_->Tag;
        FEki_api->ApiChannel        = TEki_apis_::station;
        FEki_api->EkiName           = edit_->Text;     //入力した駅名
        FEki_api->OnHttpJson        = DoEditStation;   //イベント
        FEki_api->UpdateExcect();                      //API実行
}
```

# 駅すぱあとWebサービス フリープラン + C++Builder

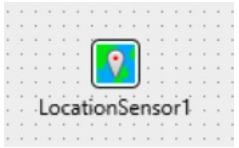
- JSON REST APIをC++Builderで呼び出す



# 駅すぱあと路線図　フリープラン + C++Builder

- Rosen JS (JavaScript)で作られたブラウザAPI

- C++Builder接続の場合
  - ロケーションセンサー(Windows, Mac, iOS, AndroidのGPS機能をカプセル化したコンポーネント)を使い位置情報取得




  - TWebBrowser(ブラウザ)からJSの関数をキックする
    - Delphi/C++Builderからブラウザ内のJavaScriptを操作可能です

# まとめ

- 各社REST サービスAPIについて
  - Amazon API Gateway
- OAuth 2.0 の仕組みと認証方法
  - TOAuth2Authenticatorコンポーネントを使って簡単OAuth2.0
  - Line, Yahoo, docomo IDなどの認証
- AWS
  - S3
- Kinveyを使ったiOSプッシュ通知
- 駅すぱあと API 接続
  - 位置情報から最寄駅検索

# THANKS!

www.embarcadero.com/jp

第33回 エンバカデロ・デベロッパーキャンプ